

Data engineering pattern in der Azure Data Factory

Stefan Kirner



Agenda

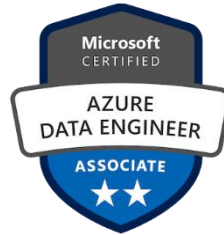
- The data challenge
- The different data engineering pattern
- The control flow
- Best practices + Q&A

About me



Stefan Kirner

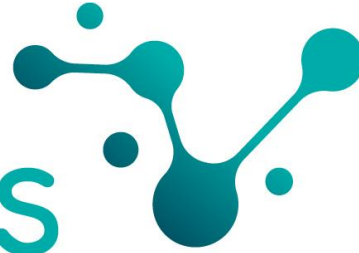
- › PASS Chapter Lead Karlsruhe ski@sqlpass.de
- › Co-Founder & Director Business Intelligence scieneers GmbH
- › Microsoft P-TSP Data Platform
- › Twitter: @KirnerKa



Wir sind

scieneers

DRIVEN BY DATA



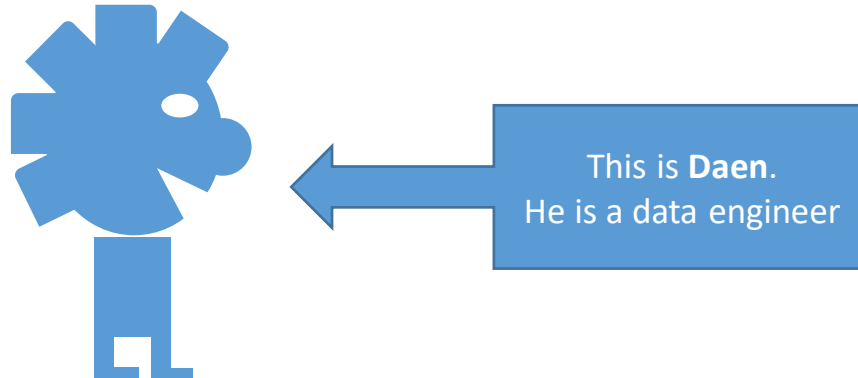
Wir gewinnen Erkenntnisse aus Daten
und schaffen damit Werte.
Für unsere Kunden, die Gesellschaft und
uns selbst.



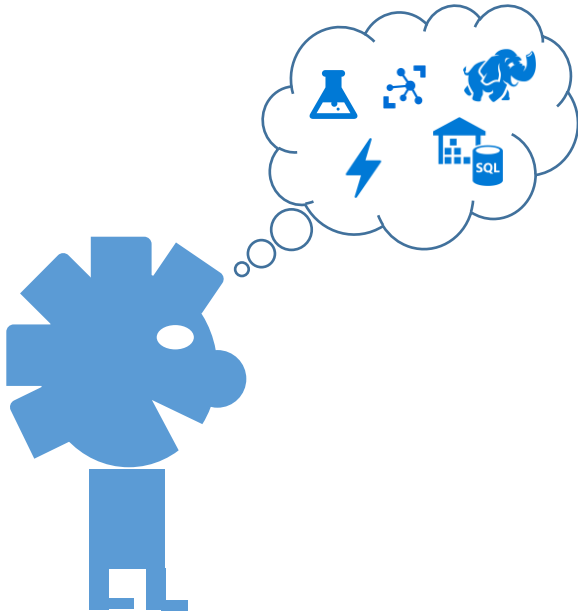
The data challenge

What is „data engineering“?

“**preparing data** for analytical or operational uses. The specific tasks [..] typically include building **data pipelines** to pull information from different source systems together; **integrating, consolidating and cleansing data**[..]”

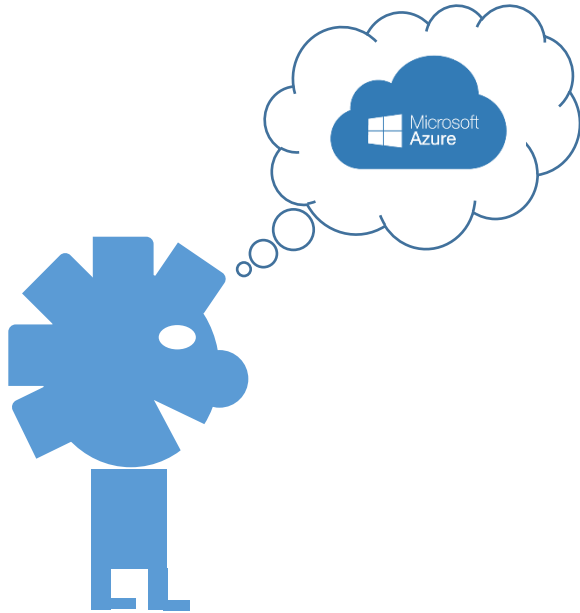


Daen thinks about a new data platform



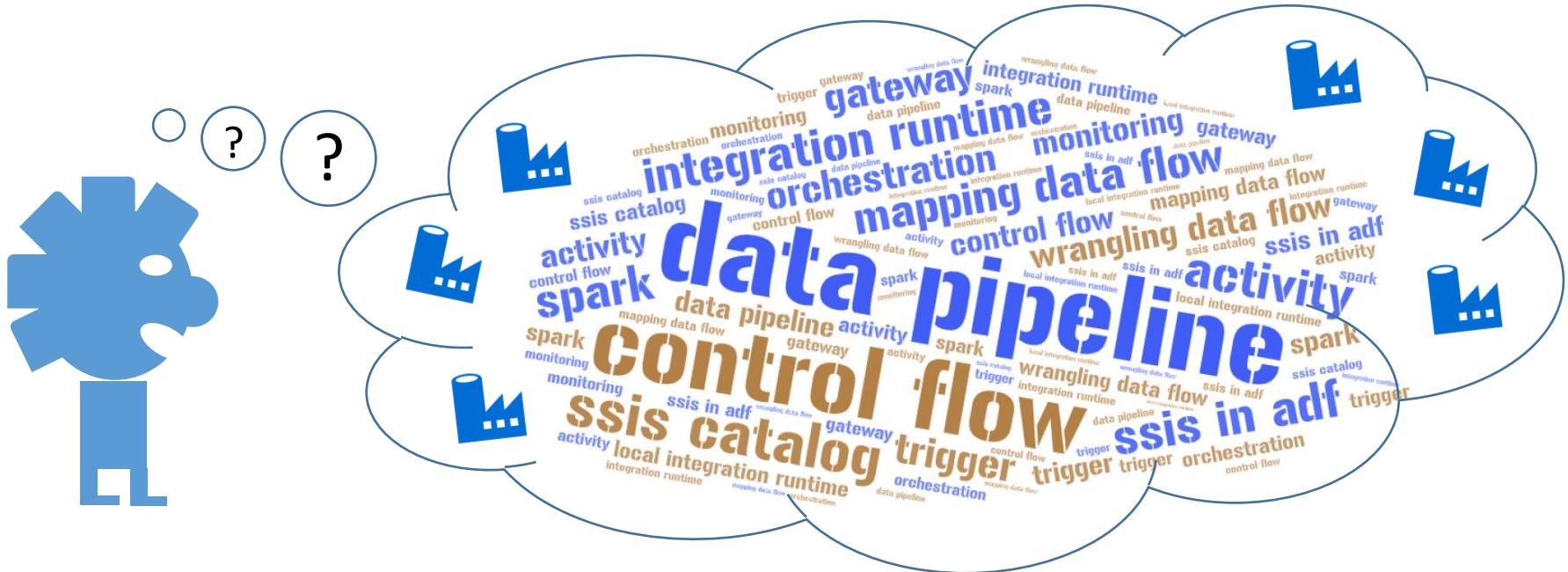
- structured & unstructured data
- mass data & messed up data
- integrate existing BI systems
- on-prem & cloud based source data
- less staff for IT system engineering

Why using managed services on a cloud platform?



- Scalability to infinity
- Cutting edge technologies
- Relaxed IT staff
- Agile Setup
- Increased reliability
- GO FAST!

Azure Data Factory



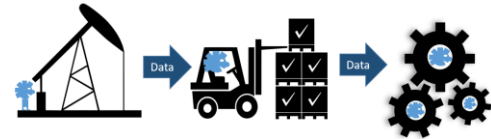
So many ways...

...which is the best?

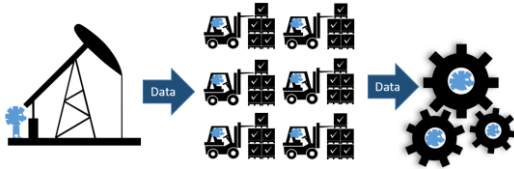
Some pattern visible in the fog



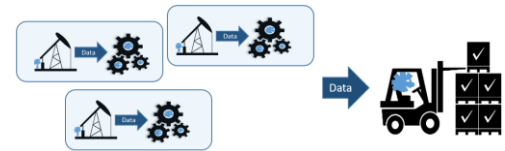
Workflow - Engine



Big Data cluster



Big DataFlow

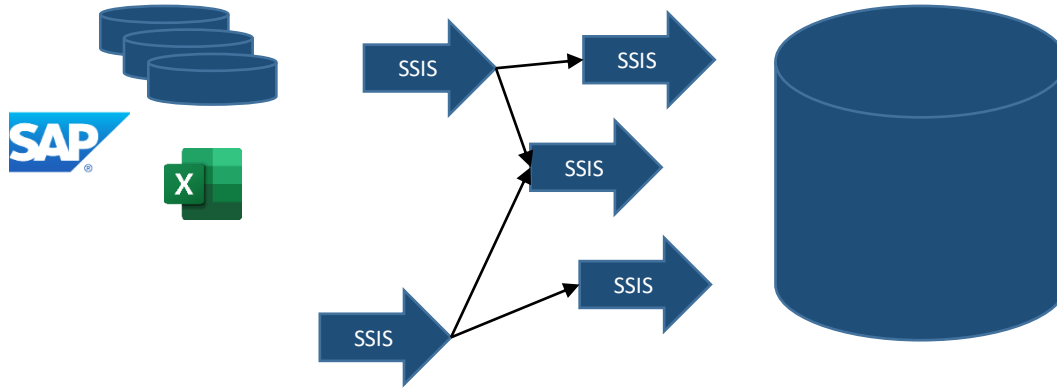


Self-Service Integration

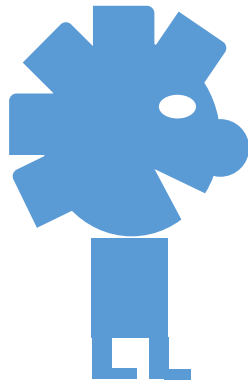


Workflow - Engine

Use Case 1: Migrate Existing Microsoft BI systems

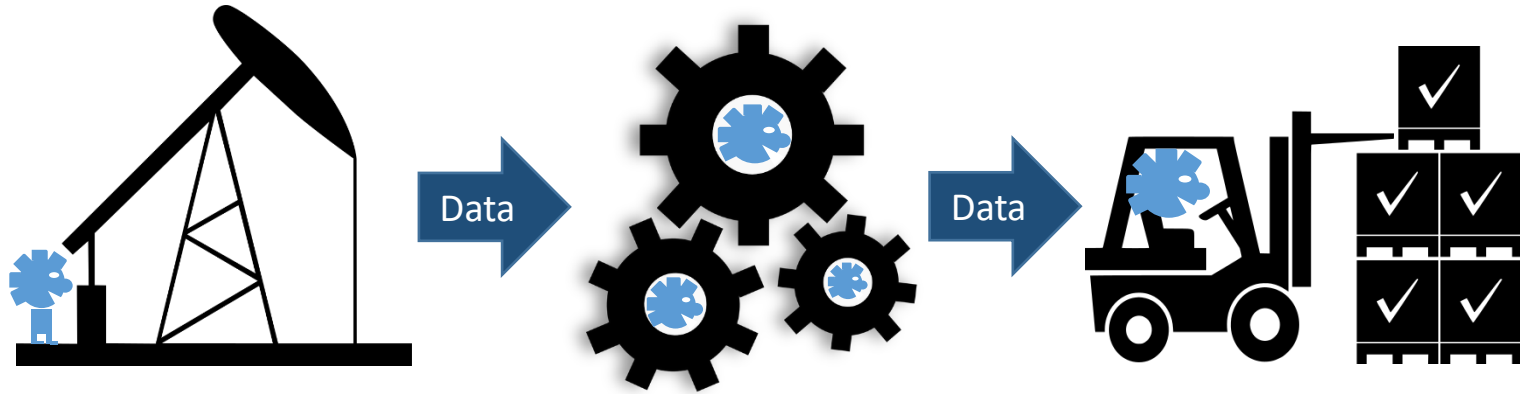


Daen has to:



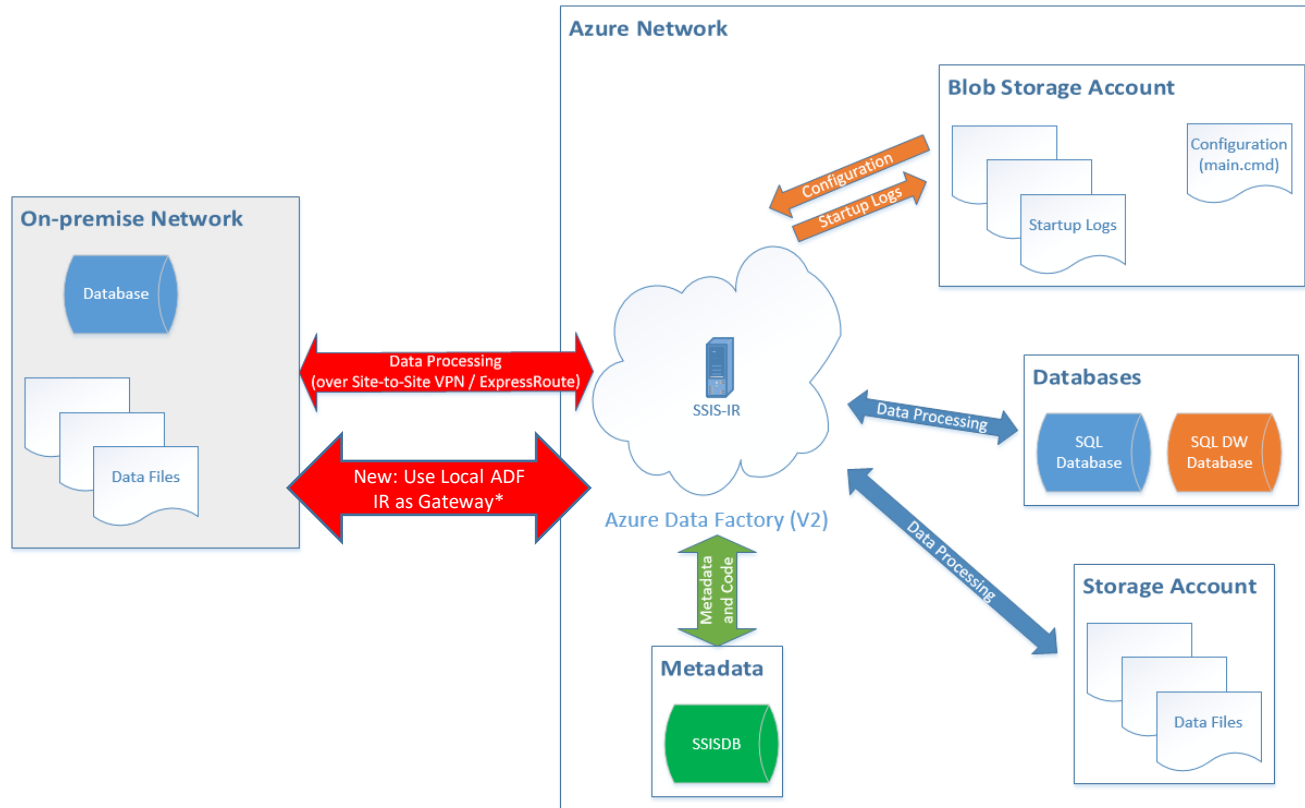
- Run logic of 1000+ SSIS packages
- Reuse deep SSIS Know-How of his team
- Bring workload to Azure with less effort

Workflow Engine = Integration Services on Azure



All logic inside Integration Services Workflow Engine
Extraction-Transformation-Load Scheme

Integration Services in ADF - components



<https://blog.westmonroepartners.com/the-new-azure-data-factory-now-with-ssis-compatibility/>

* <https://docs.microsoft.com/en-us/azure/data-factory/self-hosted-integration-runtime-proxy-ssis>

Integration Services Development

Pre-defined elements for typical tasks

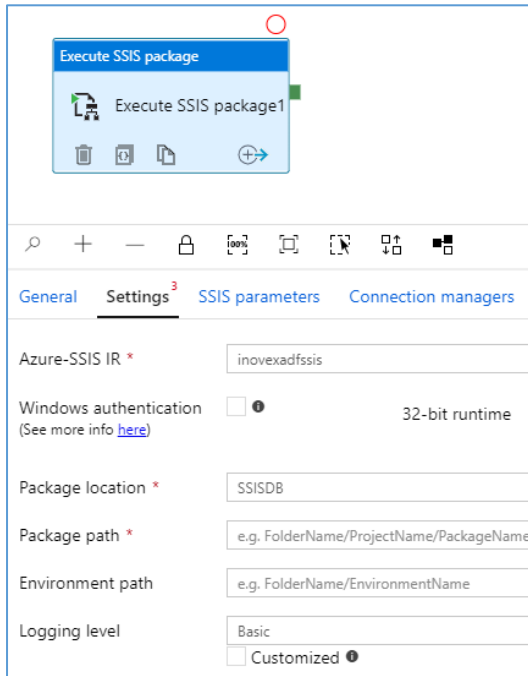
Canvas: Undo, Toolbox, Wizards, Autosave, Comments

Project-model for easy deployment

Parameter-model/Additional to variables

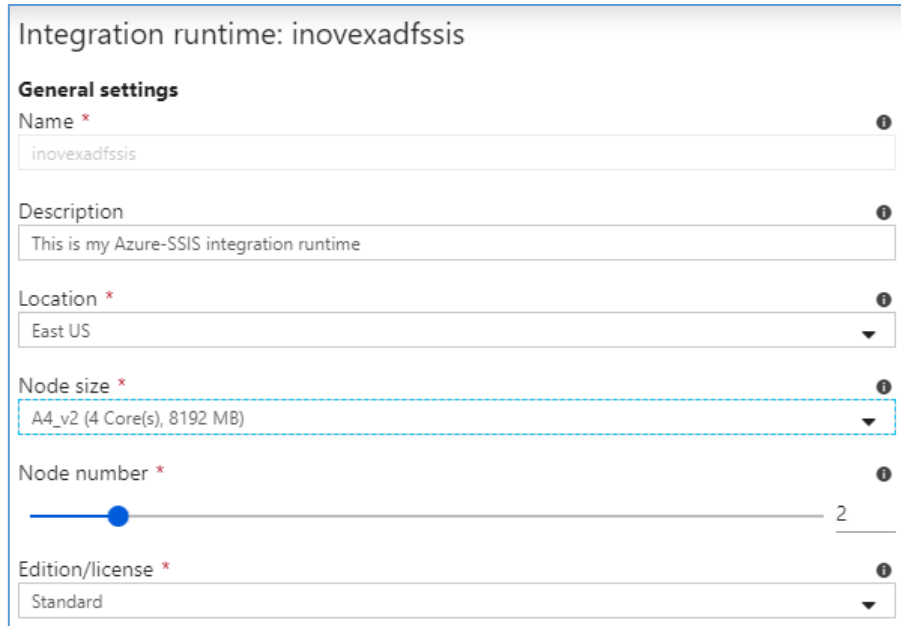
The screenshot shows the Microsoft Visual Studio IDE with an Integration Services project. The main workspace contains a workflow diagram with several tasks connected by arrows. On the left, there is a task palette with categories like 'Data Flow Task', 'Common', and 'Containers'. On the right, a project browser shows the project structure. At the bottom right, a properties window is open for the 'Fact_DivStrategy_Daily Package', showing fields for 'Description', 'ID', 'Name', and 'Name'.

Integration Services in Azure Data Factory



The screenshot shows the configuration for an 'Execute SSIS package' task. The task name is 'Execute SSIS package1'. Below the task name, there are icons for deleting, refreshing, and saving. The configuration is divided into four tabs: 'General', 'Settings', 'SSIS parameters', and 'Connection managers'. The 'Settings' tab is active, showing the following configuration:

- Azure-SSIS IR: inovexadfssis
- Windows authentication: 32-bit runtime (See more info [here](#))
- Package location: SSISDB
- Package path: e.g. FolderName/ProjectName/PackageName
- Environment path: e.g. FolderName/EnvironmentName
- Logging level: Basic Customized



The screenshot shows the configuration page for the 'Integration runtime: inovexadfssis'. The configuration is divided into several sections:

- General settings**
 - Name: inovexadfssis
 - Description: This is my Azure-SSIS integration runtime
 - Location: East US
 - Node size: A4_v2 (4 Core(s), 8192 MB)
 - Node number: 2
 - Edition/license: Standard


Good & bad about workflow engine pattern?

- Stability ★★
 - No childhood diseases (+)
 - Breaks on source data type changes (-)
- Maturity ★★★★★
 - 15 years old, much used, many skilled developers (+)
- Supported data sources & sinks ★★★★★
 - Many supported data sources and additional 3rd party components available (+)
 - Difficult to integrate semi-structured data (-)

Good & bad about workflow engine pattern?

- Lifecycle management support 
 - Visual Studio Projects for builds & source code management (+)
 - Useful environments (dev, test, prod) in SSMS (+)
 - Monitoring included in SSMS (+)
- Scalability 
 - Scale-out option only distributes packages (-)
 - Limited by resources of runtime computer (-)

Good & bad about workflow engine pattern?

- Learning curve 
 - GUI has high learn curve (+)
 - is self-documenting (+)
 - data viewers useable, but no real WYSIWYG mode (-)

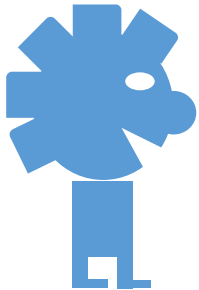
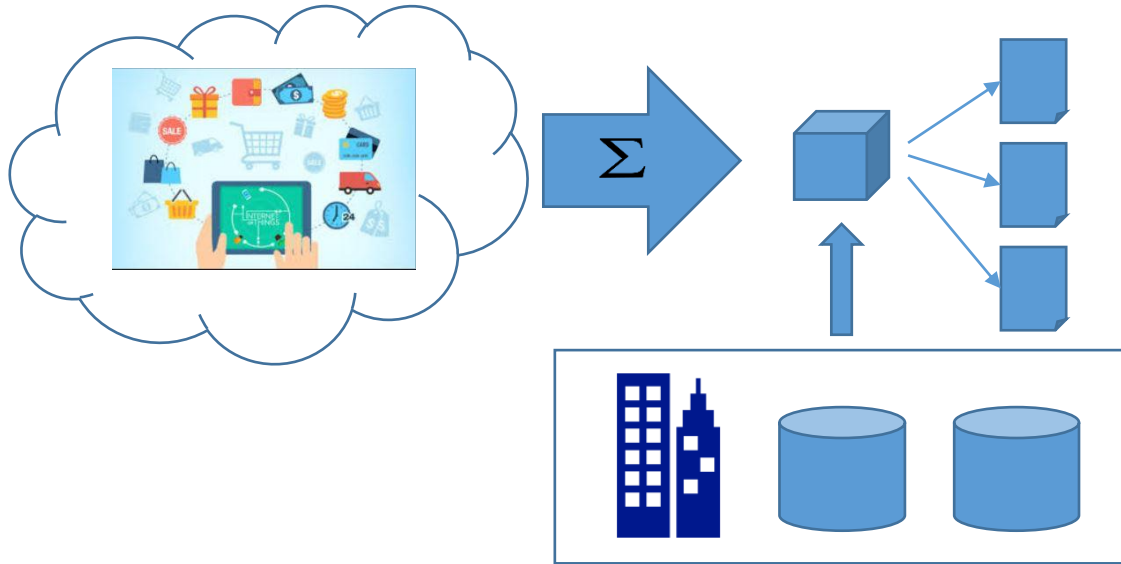
Best use cases for workflow engine

- Lift & Shift scenarios of MS BI on-prem systems using SSIS
- Small & medium data volumes
- Well structured (relational) data
- Minor schema changes
- Deterministic usage of data (e.g. for this report)



Big data cluster pattern

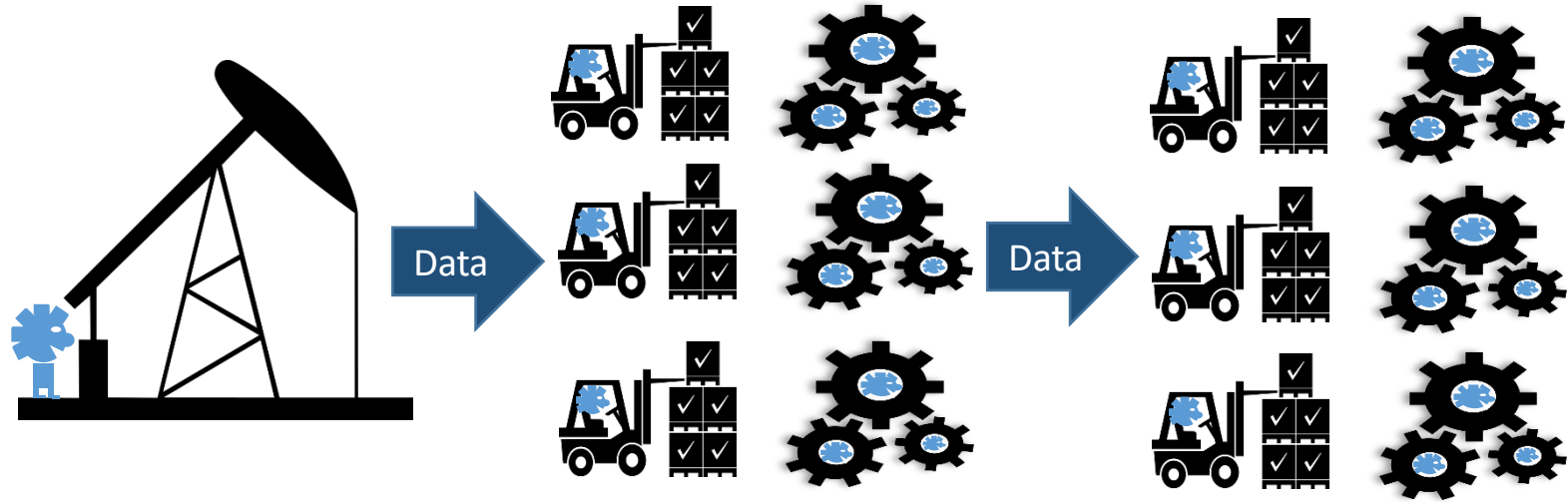
Use Case 2: cloud born mass data



Daen has to:

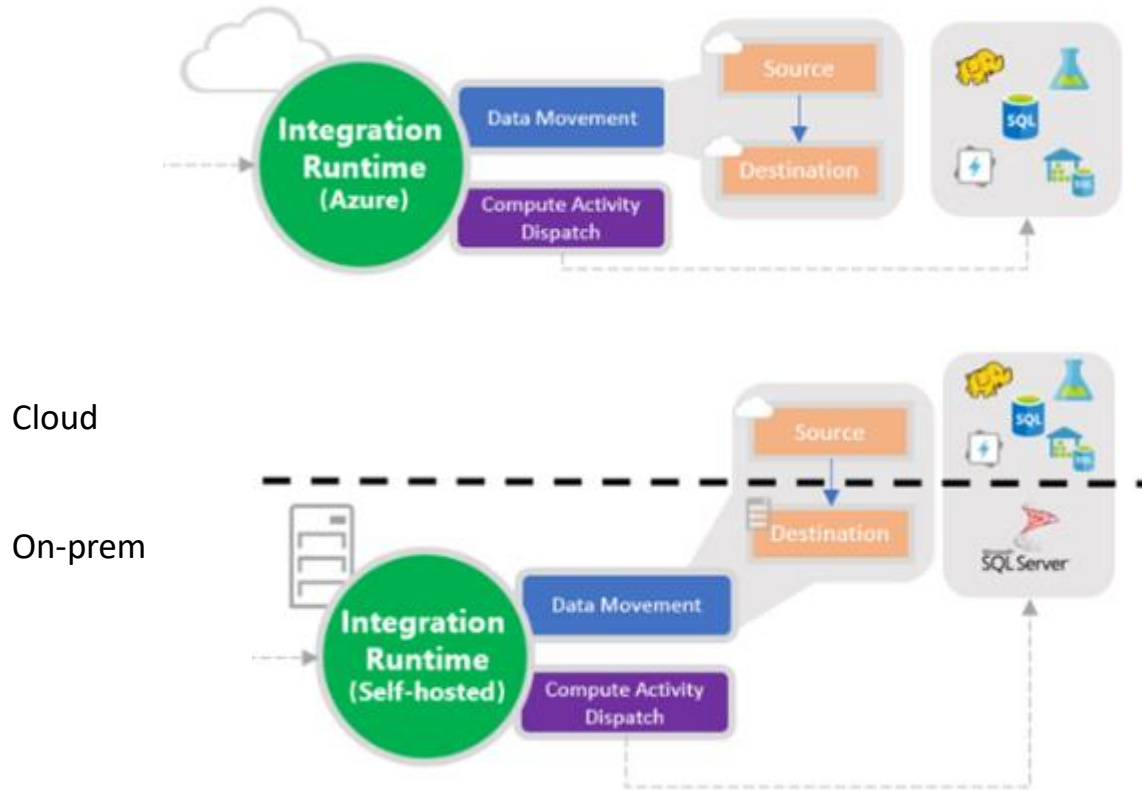
- Extract cloud-born mass data
- Aggregate data
- Combine with on-prem data
- Prepare data for analysis

Big data cluster – Let the engine do the work

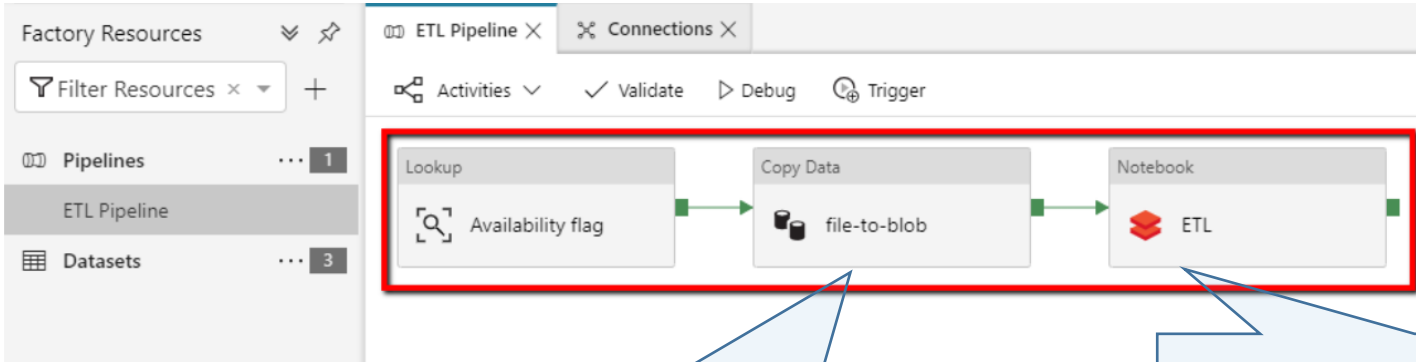


Copy to destination, Logic in external processing engine like Hadoop, Spark, SQL Server procedures...

Resources in Azure Data Factory



Big data cluster development



General Source Sink Mapping **Settings** User properties

i You will be charged # of used DIUs * copy duration * \$0.25/DIU-hour. Local currency and s

Data integration unit: 16 Edit

Degree of copy parallelism: 8 Edit

Fault tolerance: Skip incompatible rows Edit

Enable staging: [Add dynamic content \[Alt+P\]](#)

Staging settings

Staging account linked service: Select...

Create DataFrame, specify path to read file from, sort the DataFrame based on descending order of price




```
Cmd 10
1 # Create DataFrame and cleanse the data
2
3 from pyspark.sql.functions import desc
4
5 inputFile = "dbfs:/mnt/adfdata"+getArgument("input")+"/"+getArgument("filename")
6 initialDF = (spark.read # The DataFrameReader
7   .option("header", "true") # Use first line of all files as header
8   .option("inferSchema", "true") # Automatically infer data types
9   .csv(inputFile) # Creates a DataFrame from CSV after reading in the file
10 )
11
12 finalDF = (initialDF
13   .select("product_id", "category", "brand", "model", "size", "price")
14   .withColumnRenamed("product_id", "prodID")
15   .sort(desc("price"))
16 )
17
18 display(finalDF)
```

Cmd 11



Create Column with Computed value and call it doublePrice

```
Cmd 12
1 from pyspark.sql.functions import col
2 doublePriceDF = finalDF.withColumn("doublePrice", col("price") * 2)
3 doublePriceDF.show()
4 display(doublePriceDF)
```


Good & bad about big data cluster pattern?

- Stability 
 - Data type changes do not break loads - schema on read (+)
- Maturity 
 - Hadoop & spark clusters kind of commodity today (+)
 - Azure data factory copy activity well known (+)
 - Less good skilled staff available than for SSIS (-)
- Supported data sources & sinks
 - Not as many as SSIS but many (+) 

Good & bad about big data cluster pattern?

- Lifecycle management support 
 - Using different tools for copy / transform (-)
 - No visual studio support for ADF, no builds (-)
 - Code difficult to understand for operator (-)
 - More storage layers needed (-)
 - Source code integration in Azure portal (+)
- Scalability 
 - Easy scale-out for transformations (+)
 - and data copy tasks (+)

Good & bad about big data cluster pattern?

- Learning curve 
 - High for ADF Copy Tasks (+)
 - Low for programming / scripting in PySpark / Java etc. (-)
 - No WYSIWYG Editor (-)
 - Development in Notebooks could help documenting (+)

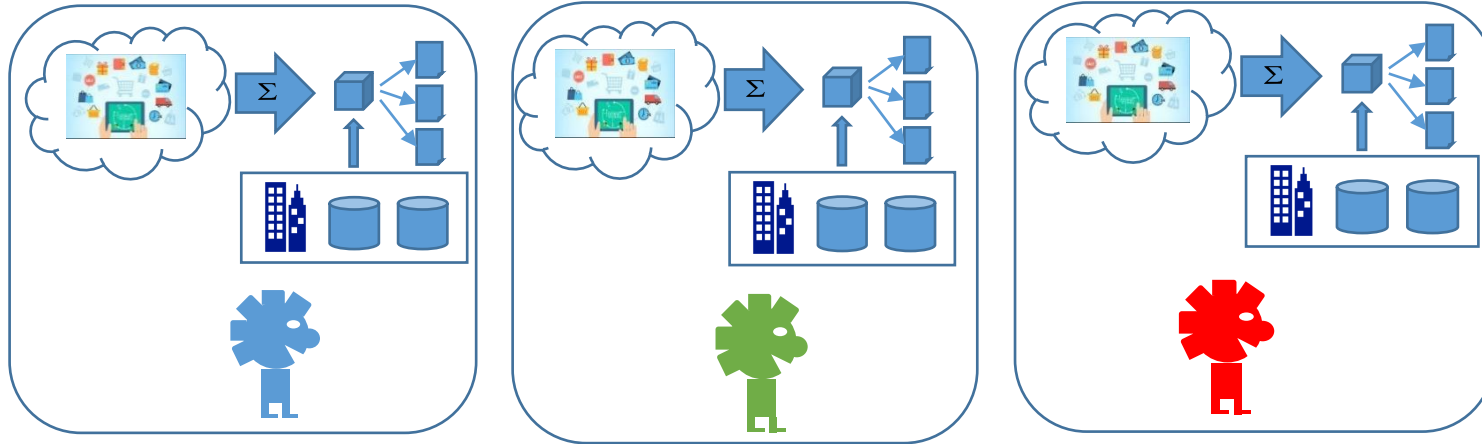
Use cases for Big Data clusters

- High volume of data
- Partitioning possible by attributes
- Complex data sources
- Many different use cases for the data
- „Agile“ source systems with often changing schemas



Big DataFlow

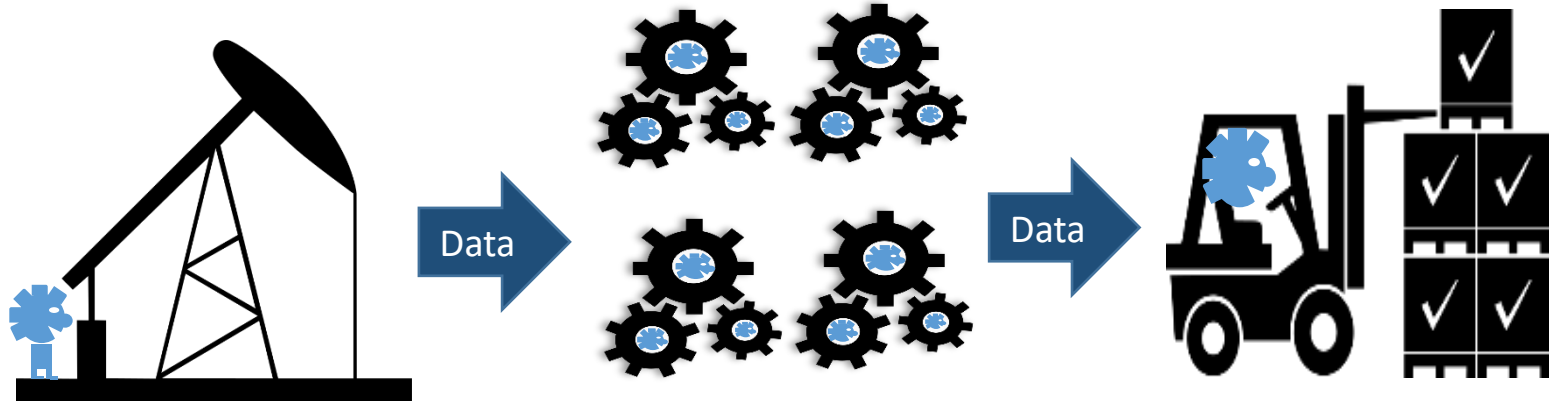
Use Case 3: **Many** cloud born mass data stores



Daen and his colleagues have to:

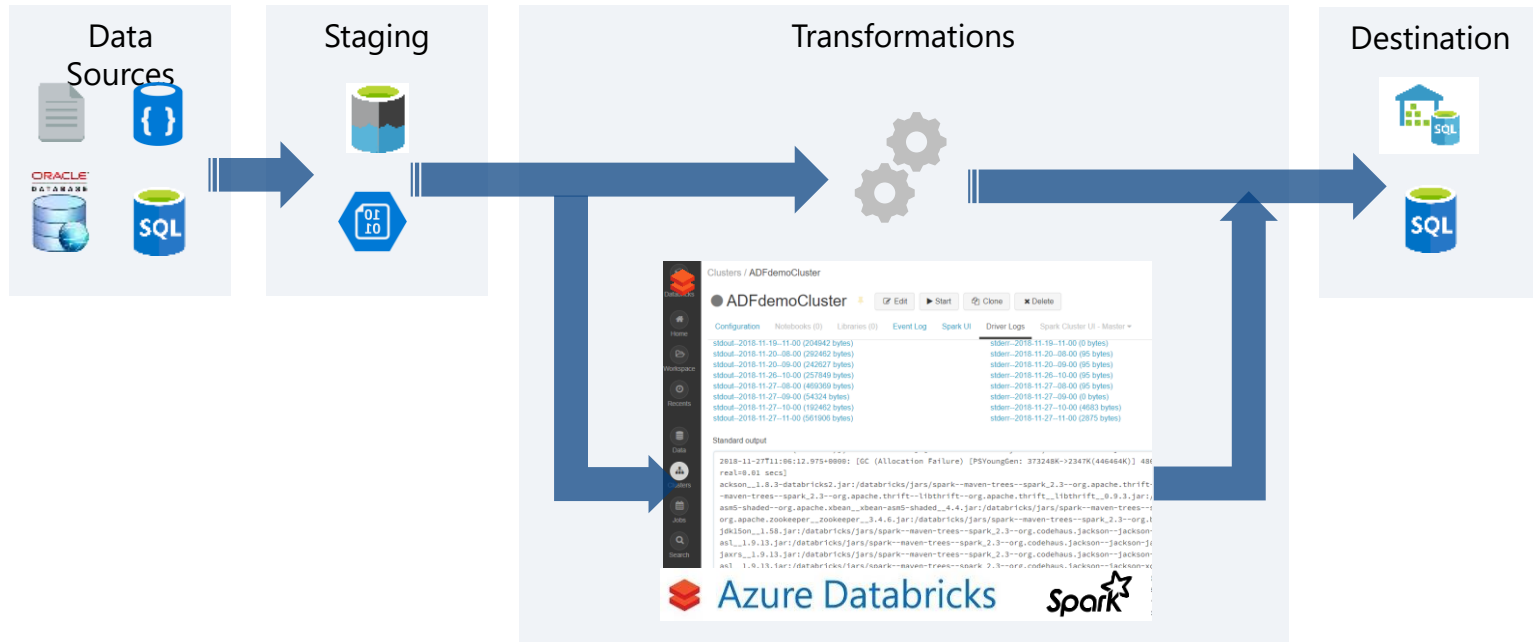
- Extract more external mass data
- Aggregate & Combine & Prepare
- **Less software engineering experience in staff**
- **Self-documenting and manageable**

Big Dataflow – Mapping dataflows in ADF



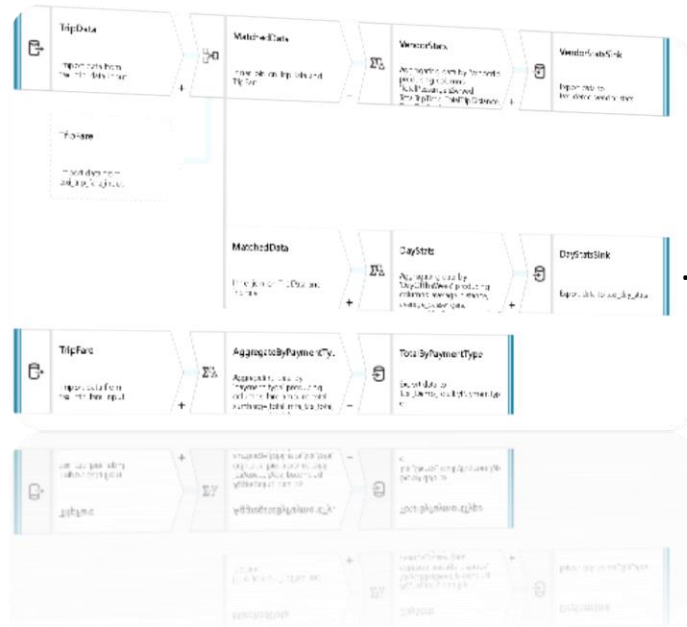
Business Logic complete in Azure Databricks external processing engine

ADF Data Flow Overview



Code-free Data Transformation At Scale

- Does not require understanding of Spark, Big Data Execution Engines, Clusters, Scala, Python ...
- Focus on building business logic and data transformation
 - Data cleansing
 - Aggregation
 - Data conversions
 - Data prep
 - Data exploration



... not ...

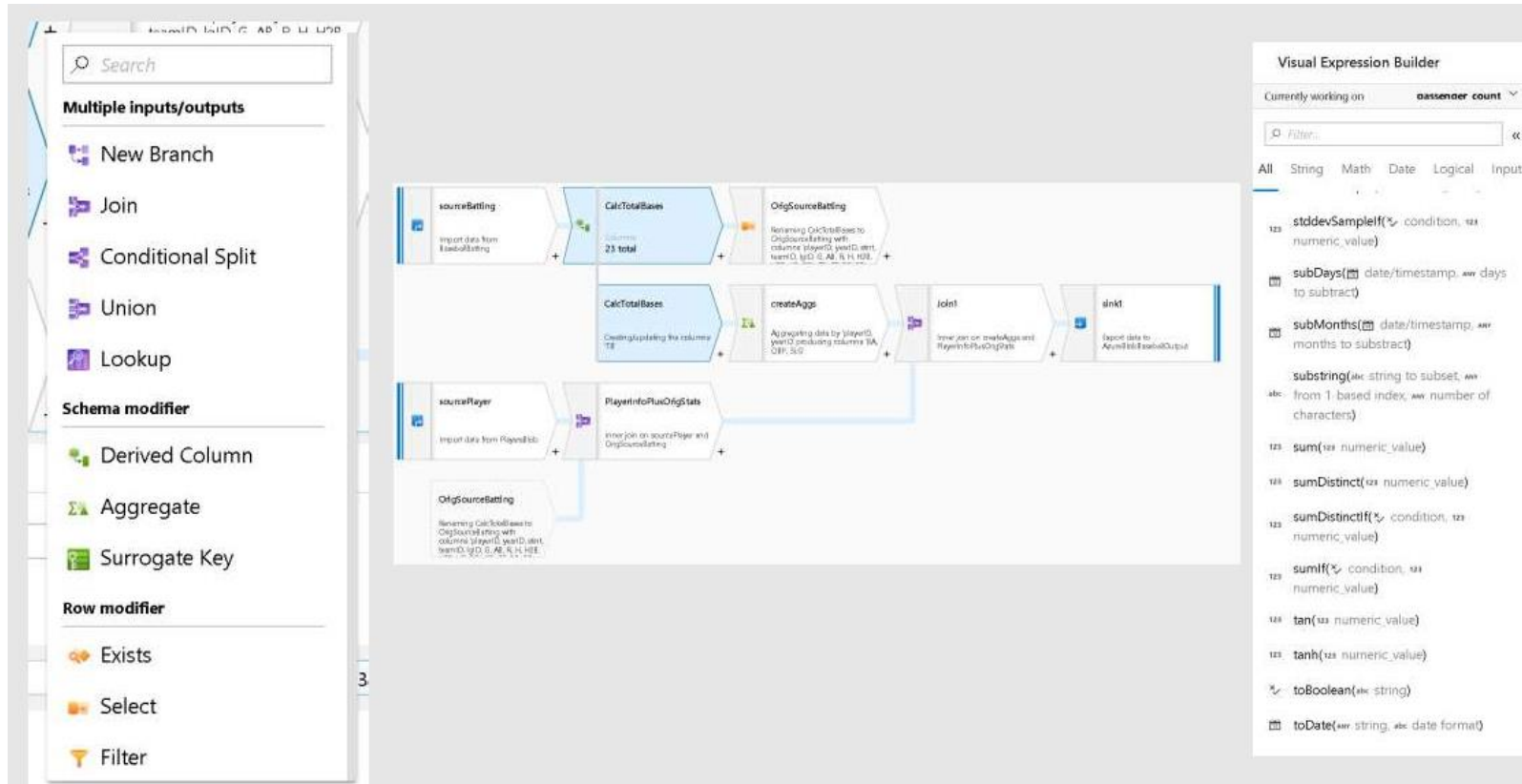
```

class MergeMatchData @scala.annotation.tailrec {
  def run(
    tipData: RDD[(String, Int)],
    matchedData: RDD[(String, Int)],
    tipFac: RDD[(String, Int)]
  ): RDD[(String, Int)] = {
    // ... Scala code for data transformation ...
  }
}

def processData(
  tipData: RDD[(String, Int)],
  matchedData: RDD[(String, Int)],
  tipFac: RDD[(String, Int)]
): RDD[(String, Int)] = {
  // ... Scala code for data transformation ...
}

object MergeMatch {
  def main(args: Array[String]): Unit = {
    // ... Scala code for main execution ...
  }
}
    
```

Development Mapping Data Flows



The screenshot displays a data flow development environment. On the left is a sidebar with various tool categories:

- Multiple inputs/outputs**
 - New Branch
 - Join
 - Conditional Split
 - Union
 - Lookup
- Schema modifier**
 - Derived Column
 - Aggregate
 - Surrogate Key
- Row modifier**
 - Exists
 - Select
 - Filter

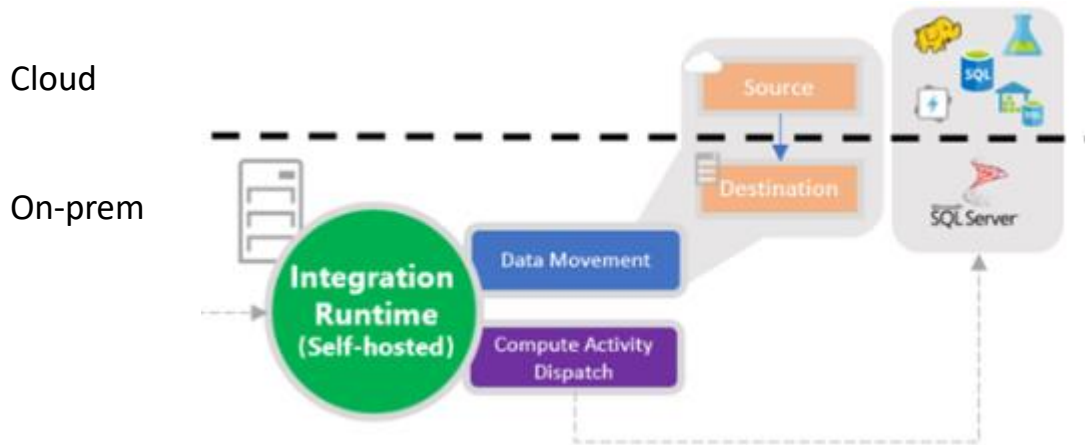
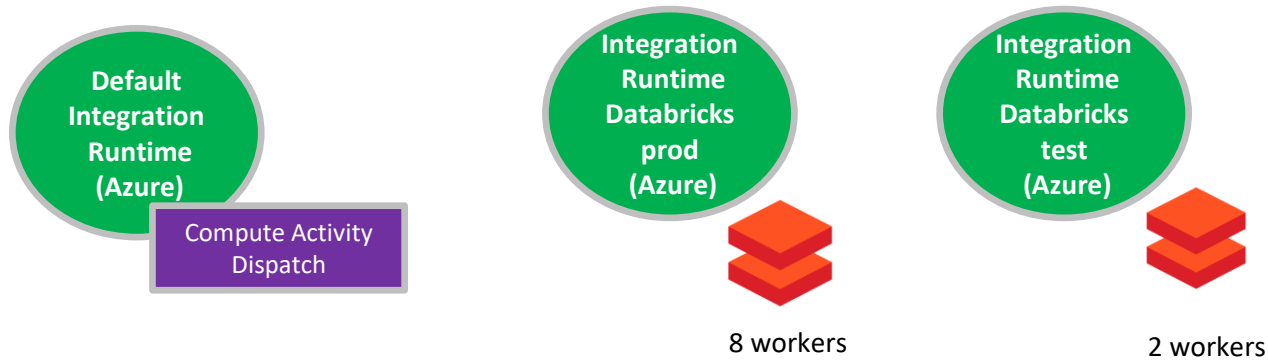
The main workspace shows a data pipeline with the following components:

- sourceBatting**: Import data from BaseballBatting
- CalcTotalBases**: Columns: 23 total
- OrigSourceBatting**: Reversing CalcTotalBases to OrigSourceBatting with columns: playerId, yearID, teamID, leagueID, AB, R, H, RBI
- sourcePlayer**: Import data from PlayerInfo
- PlayerInfoPlusOrigStats**: inner join on sourcePlayer and OrigSourceBatting
- CalcTotalBases**: Creating/Updating the columns: TB
- createAggs**: Aggregating data by (playerID, yearID) producing columns: TB, OBP, SLG
- idcol1**: inner join on metaAggs and PlayerInfoPlusOrigStats
- sink1**: Export data to MySQL:BaseballOutput




On the right, the **Visual Expression Builder** is open, showing a list of functions for the expression `passenger count`. The functions include:

- `stddevSamplef` (numeric value)
- `subDays` (date/timestamp, days to subtract)
- `subMonths` (date/timestamp, months to subtract)
- `substring` (string to subset, from 1-based index, number of characters)
- `sum` (numeric value)
- `sumDistinct` (numeric value)
- `sumDistinctf` (condition, numeric value)
- `sumif` (condition, numeric value)
- `tan` (numeric value)
- `tanh` (numeric value)
- `toBoolean` (string)
- `toDate` (string, date format)



Mapping Data Flow Resources in Azure Data Factory



Good & bad about Big DataFlow pattern?

- Stability 
 - Mechanisms to prevent break caused by data type changes (+)
 - Still some work in progress...(-)
- Maturity 
 - Pretty new in the data world – minor trust
 - Vendor lock feared
- Supported data sources & sink 
 - Not as many data inputs/outputs components available as SSIS & ADF Copy

Good & bad about Big DataFlow pattern?


- Lifecycle management support 
 - Single data-flow pipeline for copy / transform leads (+)
 - Fewer additional storage layers needed (+)
 - No visual studio support for ADF, no builds (-)
 - Source code integration in Azure portal (+)
- Scalability 
 - Easy scale out in same way integrated in ADF for copy/transform (+)
 - But could never scale as good as the optimized code of an expert (-)

Good & bad about Big DataFlow pattern?

- Learning curve 
 - WYSIWYG- GUI has high learn curve (+)
 - and is self-documenting (+)
 - No software engineering skills needed (+)

Typical use cases for Big Data workflow

- High volume of data
- Partitioning possible by attributes
- Complex data
- Many different use cases for the data
- „Agile“ source systems with often changing schemas
- Many different source systems / operators
- Less software engineering skills
- Less IT engineering skills for clusters

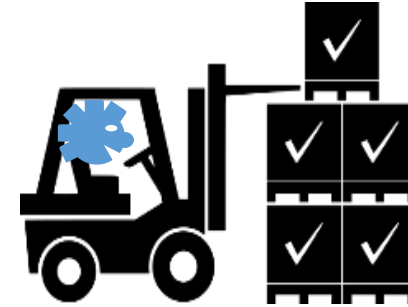
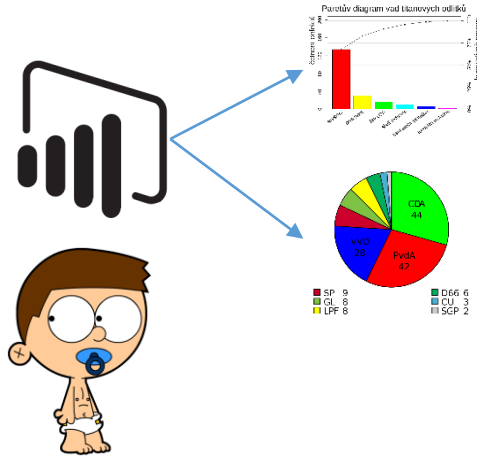


Same as big
data cluster
pattern



Self-Service Integration

Use Case 4: Reuse existing Power Query Code



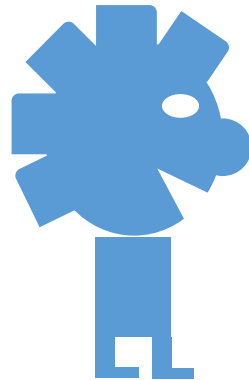
Daen has to:

Reuse students Power Query work

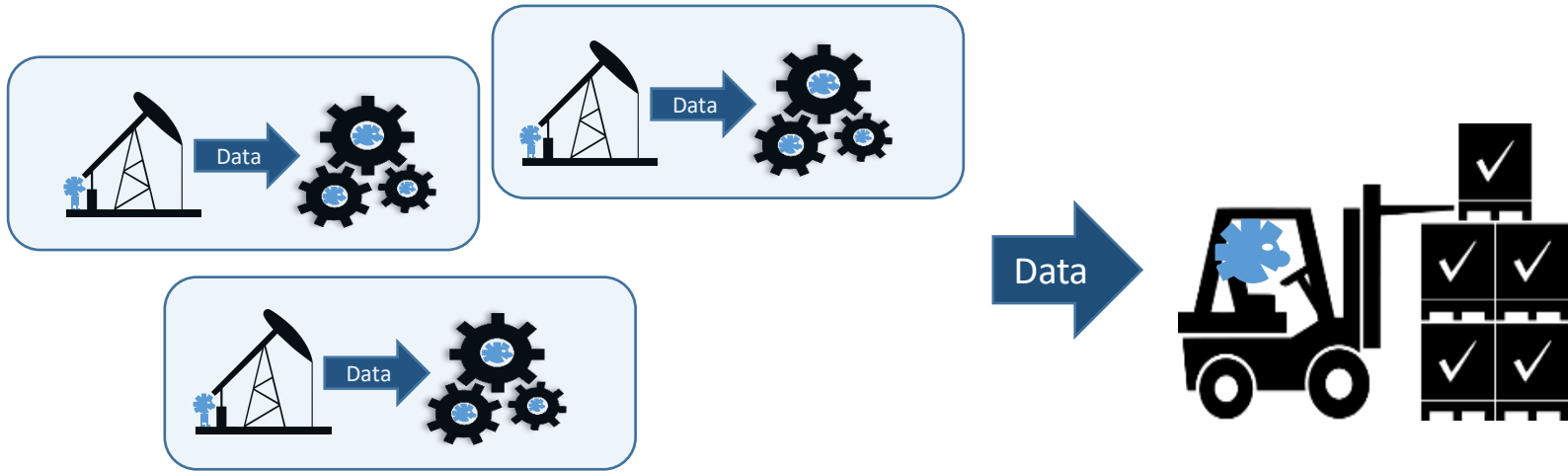
Without rewriting code (complex)

Run this regularly & managed

Integrate data on data platform

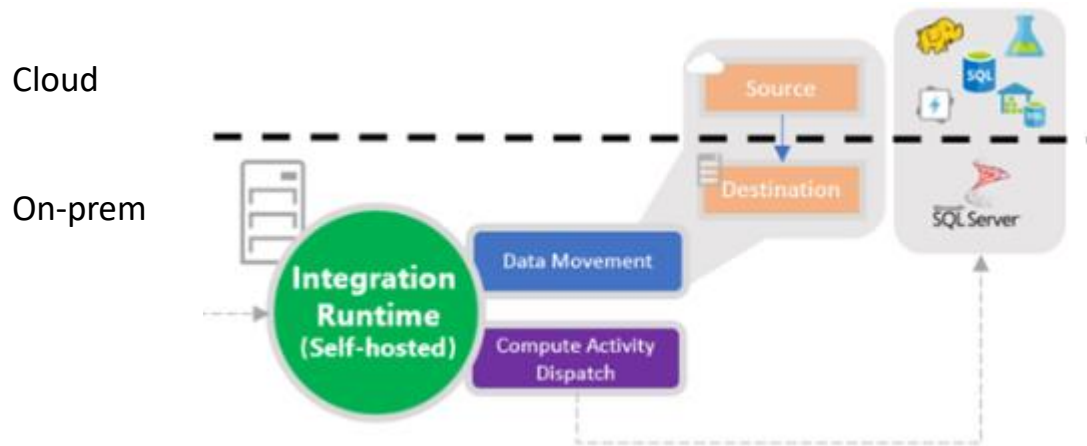
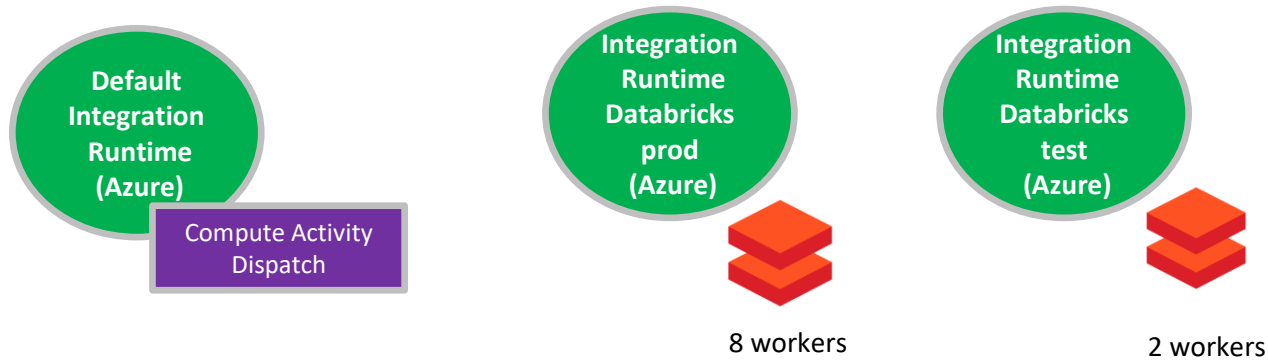


Self-Service Integration – Wrangling Dataflows

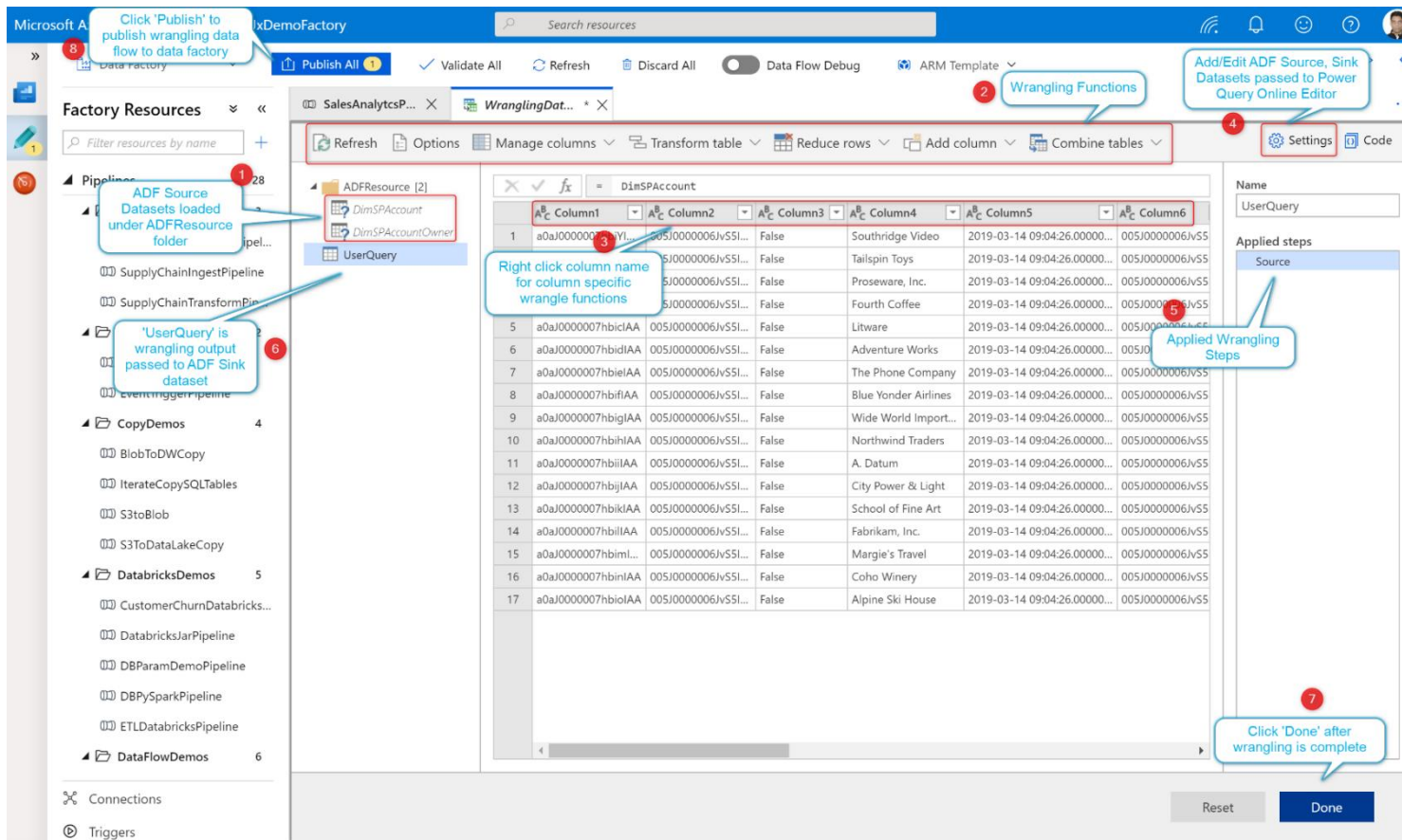


Logic in Azure Databricks external processing engine

Wrangling Data Flow Resources in Azure Data Factory



Development Wrangling Data Flows



1. Click 'Publish' to publish wrangling data flow to data factory

2. Wrangling Functions

3. Right click column name for column specific wrangle functions

4. Add/Edit ADF Source, Sink Datasets passed to Power Query Online Editor




5. Applied Wrangling Steps

6. 'UserQuery' is wrangling output passed to ADF Sink dataset



7. Click 'Done' after wrangling is complete

	Column1	Column2	Column3	Column4	Column5	Column6
1	a0aJ0000007h...	005J0000006jv...	False	Southridge Video	2019-03-14 09:04:26.00000...	005J0000006jvSS
2	a0aJ0000007h...	005J0000006jv...	False	Tailspin Toys	2019-03-14 09:04:26.00000...	005J0000006jvSS
3	a0aJ0000007h...	005J0000006jv...	False	Proseware, Inc.	2019-03-14 09:04:26.00000...	005J0000006jvSS
4	a0aJ0000007h...	005J0000006jv...	False	Fourth Coffee	2019-03-14 09:04:26.00000...	005J0000006jvSS
5	a0aJ0000007h...	005J0000006jv...	False	Litware	2019-03-14 09:04:26.00000...	005J0000006jvSS
6	a0aJ0000007h...	005J0000006jv...	False	Adventure Works	2019-03-14 09:04:26.00000...	005J0000006jvSS
7	a0aJ0000007h...	005J0000006jv...	False	The Phone Company	2019-03-14 09:04:26.00000...	005J0000006jvSS
8	a0aJ0000007h...	005J0000006jv...	False	Blue Yonder Airlines	2019-03-14 09:04:26.00000...	005J0000006jvSS
9	a0aJ0000007h...	005J0000006jv...	False	Wide World Import...	2019-03-14 09:04:26.00000...	005J0000006jvSS
10	a0aJ0000007h...	005J0000006jv...	False	Northwind Traders	2019-03-14 09:04:26.00000...	005J0000006jvSS
11	a0aJ0000007h...	005J0000006jv...	False	A. Datum	2019-03-14 09:04:26.00000...	005J0000006jvSS
12	a0aJ0000007h...	005J0000006jv...	False	City Power & Light	2019-03-14 09:04:26.00000...	005J0000006jvSS
13	a0aJ0000007h...	005J0000006jv...	False	School of Fine Art	2019-03-14 09:04:26.00000...	005J0000006jvSS
14	a0aJ0000007h...	005J0000006jv...	False	Fabrikam, Inc.	2019-03-14 09:04:26.00000...	005J0000006jvSS
15	a0aJ0000007h...	005J0000006jv...	False	Margie's Travel	2019-03-14 09:04:26.00000...	005J0000006jvSS
16	a0aJ0000007h...	005J0000006jv...	False	Coho Winery	2019-03-14 09:04:26.00000...	005J0000006jvSS
17	a0aJ0000007h...	005J0000006jv...	False	Alpine Ski House	2019-03-14 09:04:26.00000...	005J0000006jvSS


Good & bad about self-service integration pattern?

- Stability 
 - Less mechanisms to prevent break caused by data type changes (-)
 - Still very much work in progress...(-)
- Maturity 
 - Still in preview (-)
 - Youngest option in ADF – minor trust (-)
 - Vendor lock feared (-)
- Supported data sources & sink 
 - Minor data sources (-)

Good & bad about self-service integration pattern?

- Lifecycle management support 
 - *Same as Big DataFlow*
 - Self-service very focused approach e.g. build a report – not always suitable for a data platform with multiple purposes for data (-)
 - Less expertise of users could lead to bad queries and minor data quality (-)
- Scalability 
 - *Same as Big DataFlow*
 - Double-interpreted -> M to Mapping Flow, Mapping Flow to Scala (-)

Good & bad about self-service integration pattern?

- Learning curve 
 - *Same as Big DataFlow*
 - Re-use skills from Power BI solutions (+)
 - Re-Use existing knowlegde of business users & analysts (+)

Typical self-service integration use cases

- Someone build a very complex Power Query
 - fastest way to integrate this to a managed production data platform
- Integration of data exploration

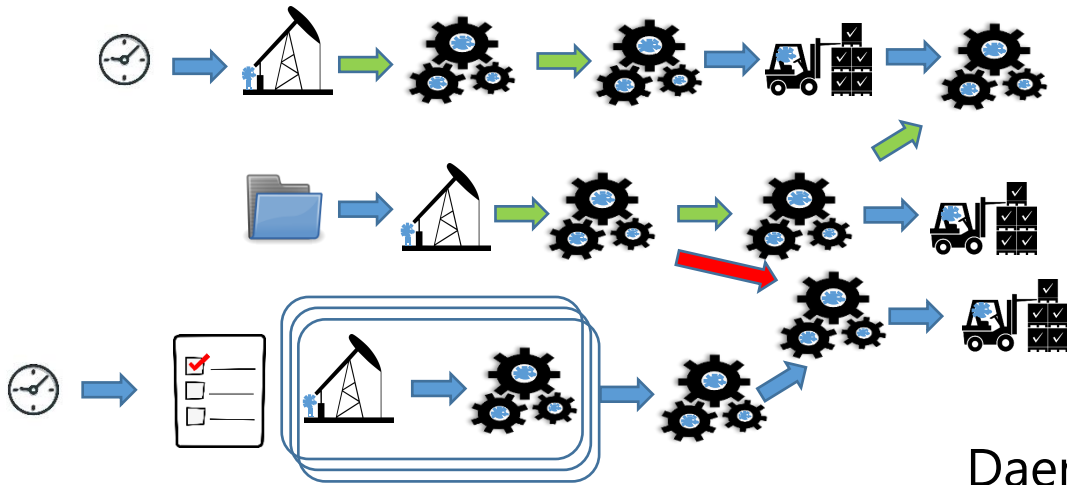
Results in overview

Criterion	Workflow engine	Big Data cluster	Big DataFlow	Self-Service
Stability	★ ★	★ ★ ★ ★	★ ★	★
Maturity	★ ★ ★ ★	★ ★ ★	★ ★	★
Supported data sources & sinks	★ ★ ★ ★	★ ★ ★	★ ★	★
Lifecycle management support	★ ★ ★ ★	★ ★ ★	★ ★	★
Scalability	★ ★	★ ★ ★ ★	★ ★ ★	★ ★
Learning curve	★ ★ ★	★ ★	★ ★ ★	★ ★ ★ ★



Control Flow

Use Case 5: Run & Keep control



Daen has to:

Model dependencies

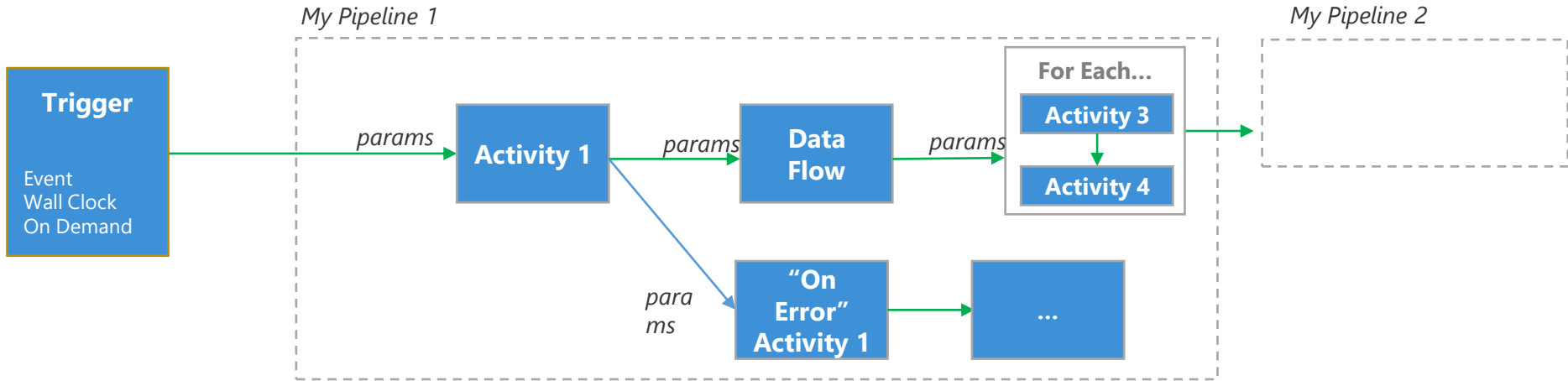
Implement error handling

Trigger loads by schedule & events

Run pipelines in a loop

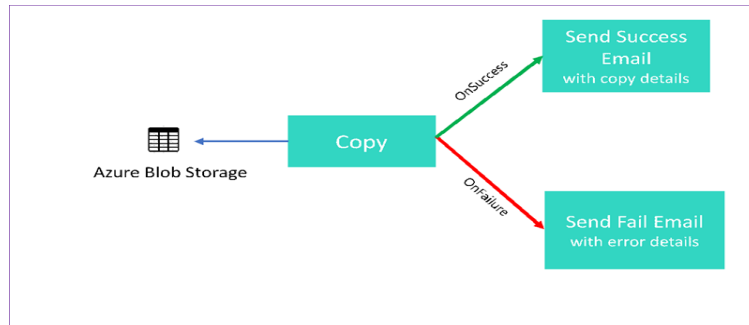
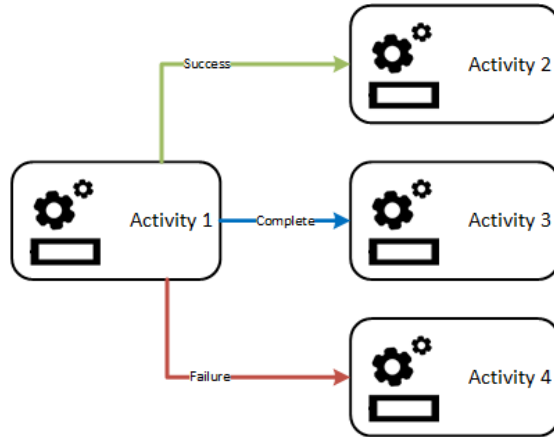
Monitor & Debug data pipelines

Control Flow – data pipelines, activities, triggers



Activities

Concepts



Branching

Dependencies of activities in a pipeline

Possible constraints:

- On success
- On failure
- On completion

Also custom 'if' conditions will be available for branching based expressions

Triggers

How do pipelines get started?

- on-demand
- Wall-clock Schedule
- Tumbling Window (aka time-slices in v1)
- Event on Blob Store

Example Control Flow

Get all data of a system by metadata

Pipeline check metadata



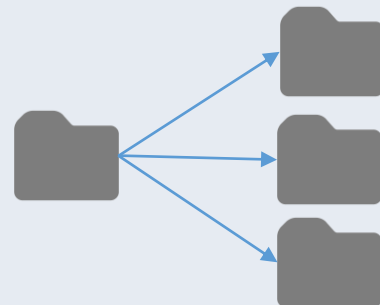
Activity: For each table
in source

exec

exec

exec

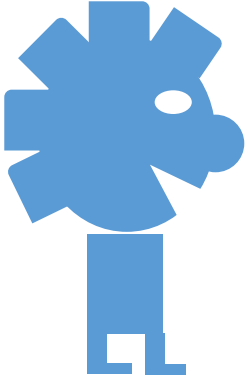
Activity: Copy
Data to Data Lake /
Blob Store





Best practices + Q&A

Best practices: Work, don't play!




Daen has to:

- Re-use logic in ADF
- Use secure mechanisms for credentials
- Deploy code to different stages w/o manual interference
- Develop with his team


Use Templates

DataMonsterDf X DataMonsterD... X **Template gallery** X


All **My Templates** Copy SSIS Transform




Bulk Copy from Database
Use this template to copy data in bulk from database using external control table to store partition list of source tables.
...
by Microsoft



Copy data from Amazon S3 to Azure Data Lake Store
Use this template to copy data from Amazon S3 to Azure Data Lake Storage.
...
by Microsoft



Copy data from Netezza to Azure Data Lake Store
Use this template to copy data from Netezza source...



Copy data from on premise SQL Server to SQL Azure
Use this template to copy data from on premise SQL...

Use Azure Key Vault for Configs

Edit linked service (Azure SQL Database)

Name *

AzSQLDB

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

Connection string

Azure Key Vault

AKV linked service *

InovexKeyVault

[Edit connection](#)

Secret name *

dbconnection

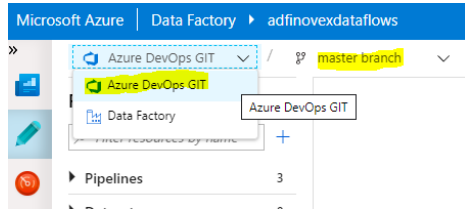
Secret version


Use the latest version if left blank

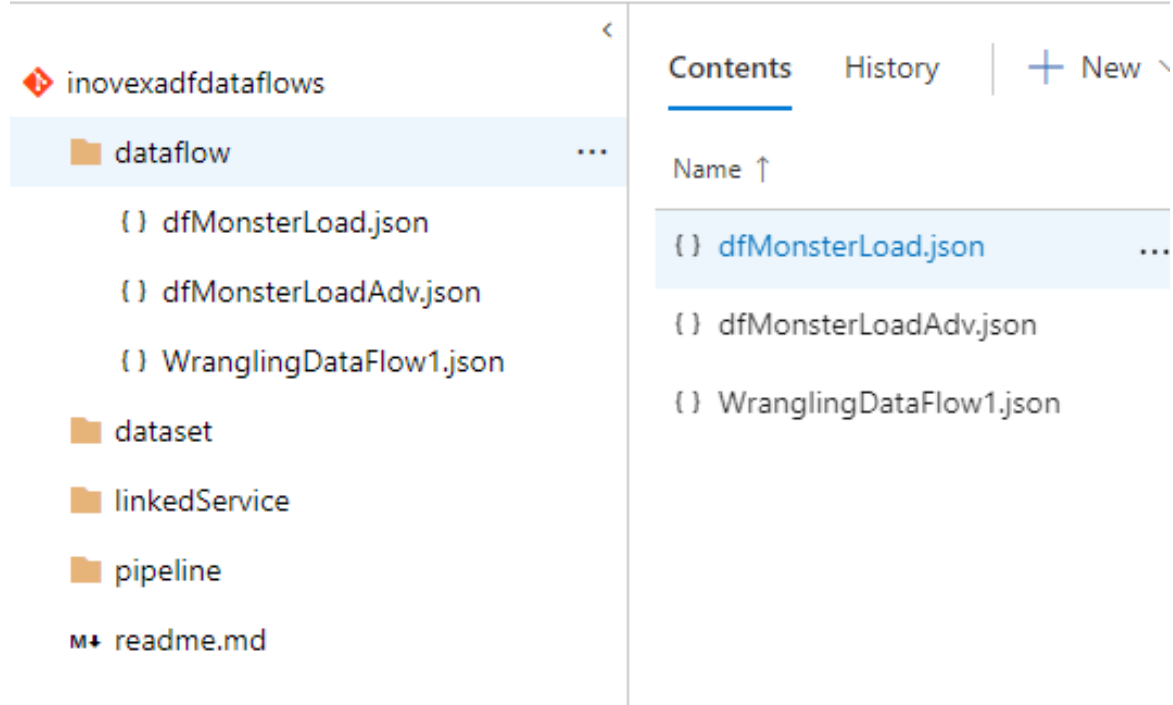
Authentication type *

Sql Authentication or Managed Identity

Use a repository for your solution



 master ▼ inovexadfdatabflows / **dataflow**



inovexadfdatabflows

- dataflow
 - dfMonsterLoad.json
 - dfMonsterLoadAdv.json
 - WranglingDataFlow1.json
- dataset
- linkedService
- pipeline
- readme.md

Contents | History | + New

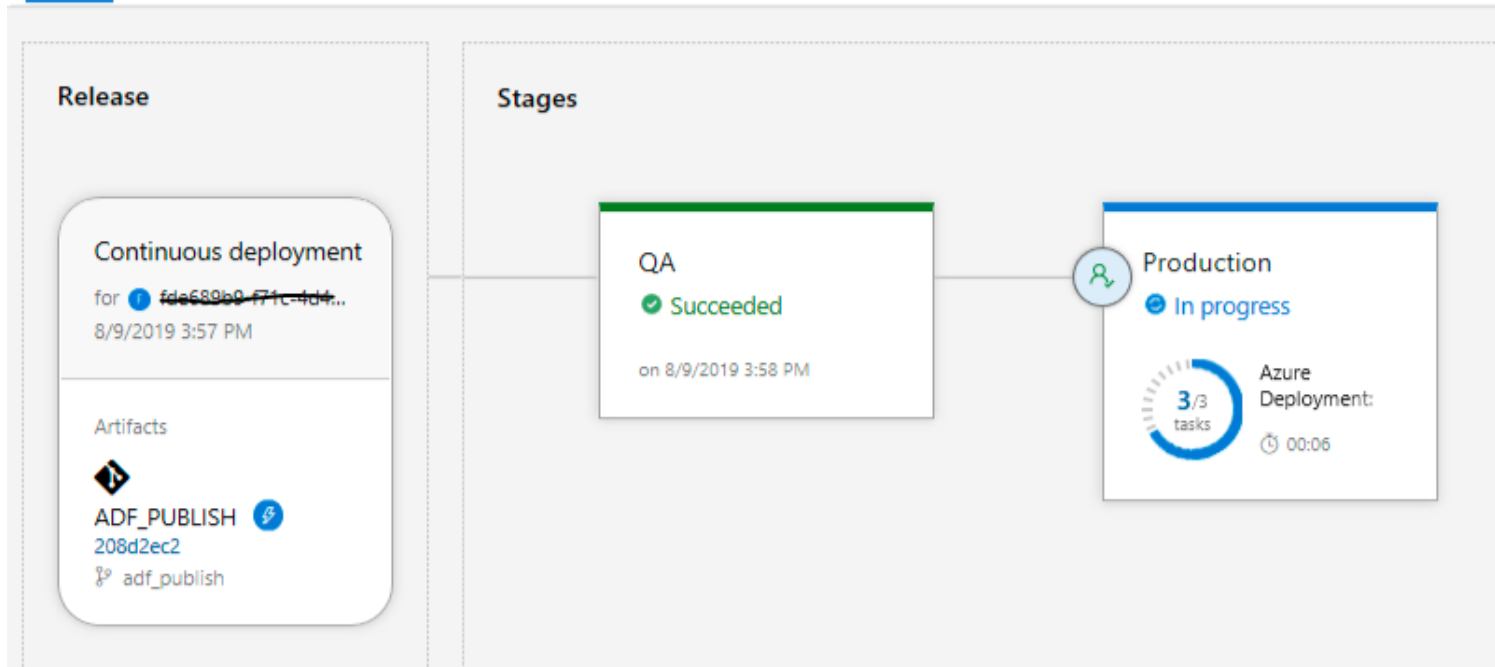
Name ↑

- dfMonsterLoad.json
- dfMonsterLoadAdv.json
- WranglingDataFlow1.json

Use Build & Release pipelines in Azure DevOps

ADFv2DemoPipeline1 > Release-2

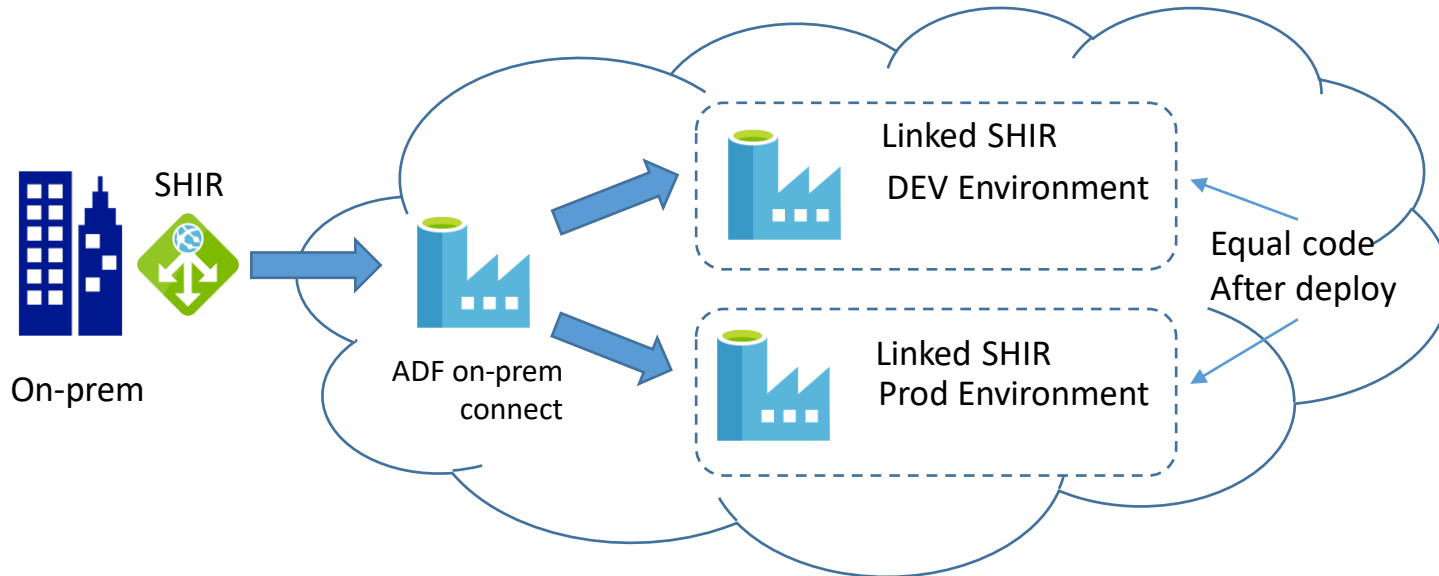
Pipeline Variables History | + Deploy Cancel Refresh Edit ...



The screenshot displays the Azure DevOps Release pipeline interface. On the left, the 'Release' section shows a 'Continuous deployment' artifact for a specific build, with a timestamp of 8/9/2019 3:57 PM. Below this, an artifact named 'ADF_PUBLISH' is listed with a file icon and a download link. The main 'Stages' section shows a sequence of two stages: 'QA', which has 'Succeeded' status, and 'Production', which is 'In progress'. The 'Production' stage includes a progress indicator showing '3/3 tasks' and a timer for 'Azure Deployment' at '00:06'.

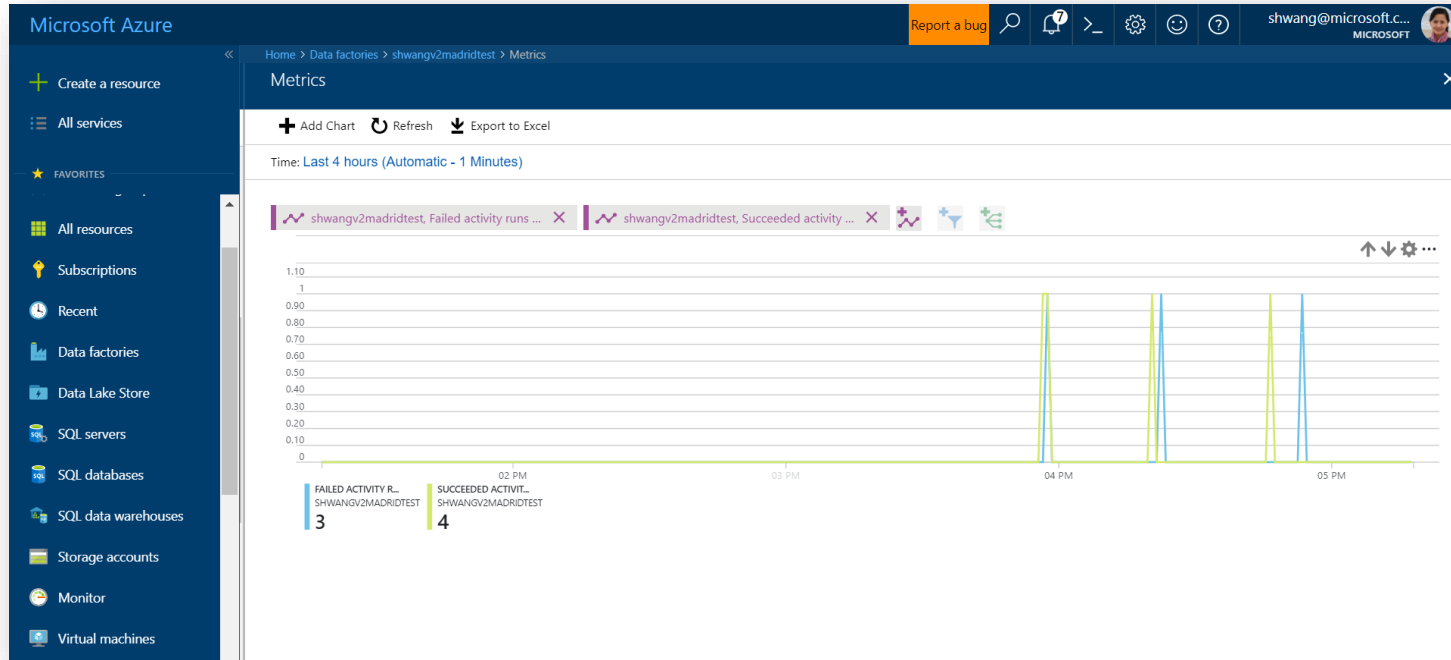
Using Linked Self-Hosted Integration Runtimes

Keep structure of data factories similar for different environments



Azure Monitoring integration

- Aggregated Metrics
- Alerts and actions
- Detailed diagnostic logs



Links and further informations

1. Microsoft documentation:

<https://docs.microsoft.com/en-us/azure/data-factory/>

2. Azure Data Factory – data flows preview documentation

<https://github.com/kromerm/adfdataflowdocs>

3. Cool screencasts about data flows

<https://github.com/kromerm/adfdataflowdocs/tree/master/videos>

4. Another good blogpost about ADF Data Flows

<https://visualbi.com/blogs/microsoft/azure/azure-data-factory-data-flow-activity/>

5. Comparison ADF Data Flows vs. SSIS vs. T-SQL

<https://sqlplayer.net/2018/12/azure-data-factory-v2-and-its-available-components-in-data-flows/>

Thank you for listening!

Daen is happy now and can better decide which pattern to use

