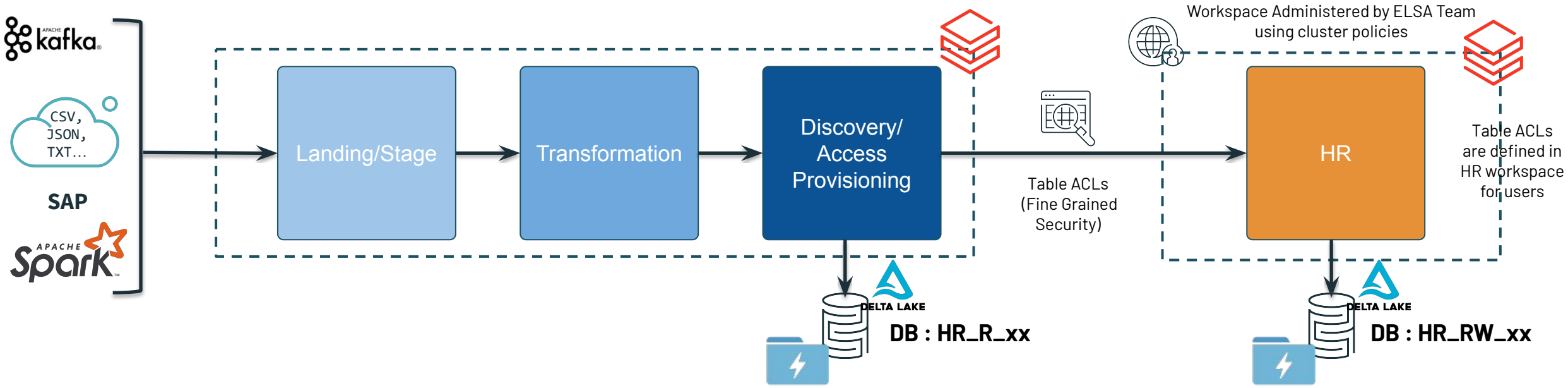
ELSA Data Encryption Concept

Team ELSA

Overview

- Business requirements: Already fulfilled with the current role design
- However, Bayer team feels “more comfortable” with data encryption in place
- Data at rest needs to be encrypted
  - Persisted tables in ADLS
  - For now, it is sufficient to encrypt ‘HR’ data (~70 tables)
- Data in motion: A first version of ‘data masking’ is already implemented
- Basically, there are two different options for encrypting data at rest:
  - Implementation with native Azure resources
  - Implementation within Databricks

Project Workspace for department(HR)



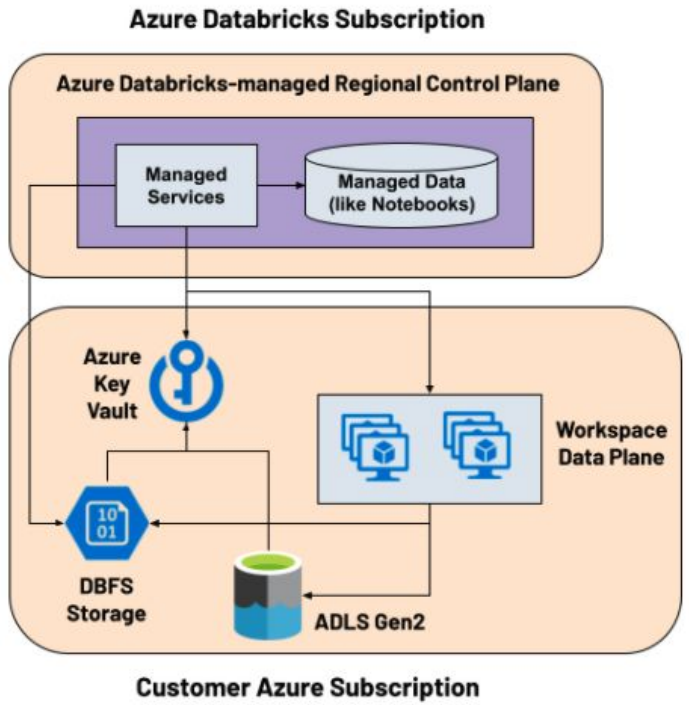
DB : HR_R_xx	DB : HR_RW_xx
• Database HR_R_xx is provisioned for HR Users with relevant views including row/column security • HR users will only have READ access over the tables in this database • ELSA Central Team will be responsible for provisioning & maintenance of views in this database • If HR needs some data from Finance, a view pointing to Finance data can be created under HR_R_xx	• Database HR_RW_xx is provisioned for HR Users by ELSA central team where they can create tables & write computed data for their use-case • HR users will only have READ & WRITE access over the tables in this database • HR use-case teams will be responsible for maintaining data/tables/views etc. in this database • Deprovisioning of data/tables/views etc. needs to be governed by standard service operations process with necessary approvals. (To mitigate impact on downstream teams)
• xx can be replaced by a combination of department, use-case and/or a business related term to make the naming convention more meaningful. • Common Metastore can be used to manage ddls & views.	

databricks

Solution 1 - Encryption based on Azure services

- **Default: Data in a storage account is encrypted with Microsoft-managed keys**
- It is possible to manage encryption of data with your own key:
  - https://docs.microsoft.com/en-us/azure/storage/common/customer-managed-keys-overview
  - https://databricks.com/blog/2021/05/21/customer-managed-key-cmk-public-previews-for-databricks-on-azure-and-aws.html
- **Encryption/Decryption process is transparent**
  - If an identity is allowed to read, data will be encrypted automaticall

Use Case	Status
CMK Managed Services (notebooks, queries and secrets stored in control plane)	Public Preview
CMK Workspace Storage (DBFS)	Generally Available (GA)
CMK for your own data sources	Already works seamlessly



Solution 2 - Encryption with Databricks

- Data can also be encrypted within Databricks
  - This solution only makes sense when a **subset of columns of the data** needs to be encrypted
  - The decryption process will significantly influence query performance on encrypted tables
  - This solution has additional drawbacks, which will be discussed on the next slide
- There is already a documented solution for column-level encryption within Databricks
  - But: This solution uses a scala UDF for decryption of the data, which currently is blocked in SQL-endpoints and hence no option for us
- For encryption/decryption we can leverage the new 'aes_encrypt' and aes_decrypt' functions of the latest DBR Runtime 10.3
  - Both functions need in the input an encryption key
- We store this encryption key in a Databricks secret with an Azure Key Vault backed scope
- Access to the secret can be controlled with Secret ACLs
  - Secrets can be accessed within a Notebook/Job, either via 'dbutils' or via 'spark configuration'
  - Currently, we do not know if it is also possible from SQL-endpoints (needs to be checked)
- Where do we apply encryption and decryption?
  - Encryption: Everytime when we persist tables in ADLS in the ETL workflow from Staging -> Transformation -> Discovery
  - Decryption: Decryption call is part of the view creation, which eventually are consumed by the users (e.g. HR users)

Solution 2 - Encryption with Databricks

- Drawbacks of this solution:
  - Significant influence on performance
  - Data types will be lost for encrypted data
    - We need to store the actual table schema in addition to the data and after decryption we need to cast the data to correct types (e.g. store data in json format and add data type as field)
  - Currently, DBSQL might not yet support DBX Runtime 10.3 and therefore the encrypt/decrypt functions are not available for the views
    - We need to check this, if the solution is considered for implementation
    - We also need to check if we can access the secret with the encryption key from SQL-endpoints

Summary & Recommendations

- Data at rest is encrypted per default with Microsoft owned keys
- We can provide our own key, if the Bayer team wants to manage the encryption keys on their own
  - However, we should discuss the use cases where this really helps us
- If we would like to encrypt only a subset of the data by encrypting on columnar level, we should consider solution 2
  - Here we would need to check
    - A) If it is possible to decrypt via SQL-endpoints
    - B) If the performance is still acceptable
- **Overall, it might be better/simpler to implement a more fine-grained access pattern for the central workspace based on the existing role design**