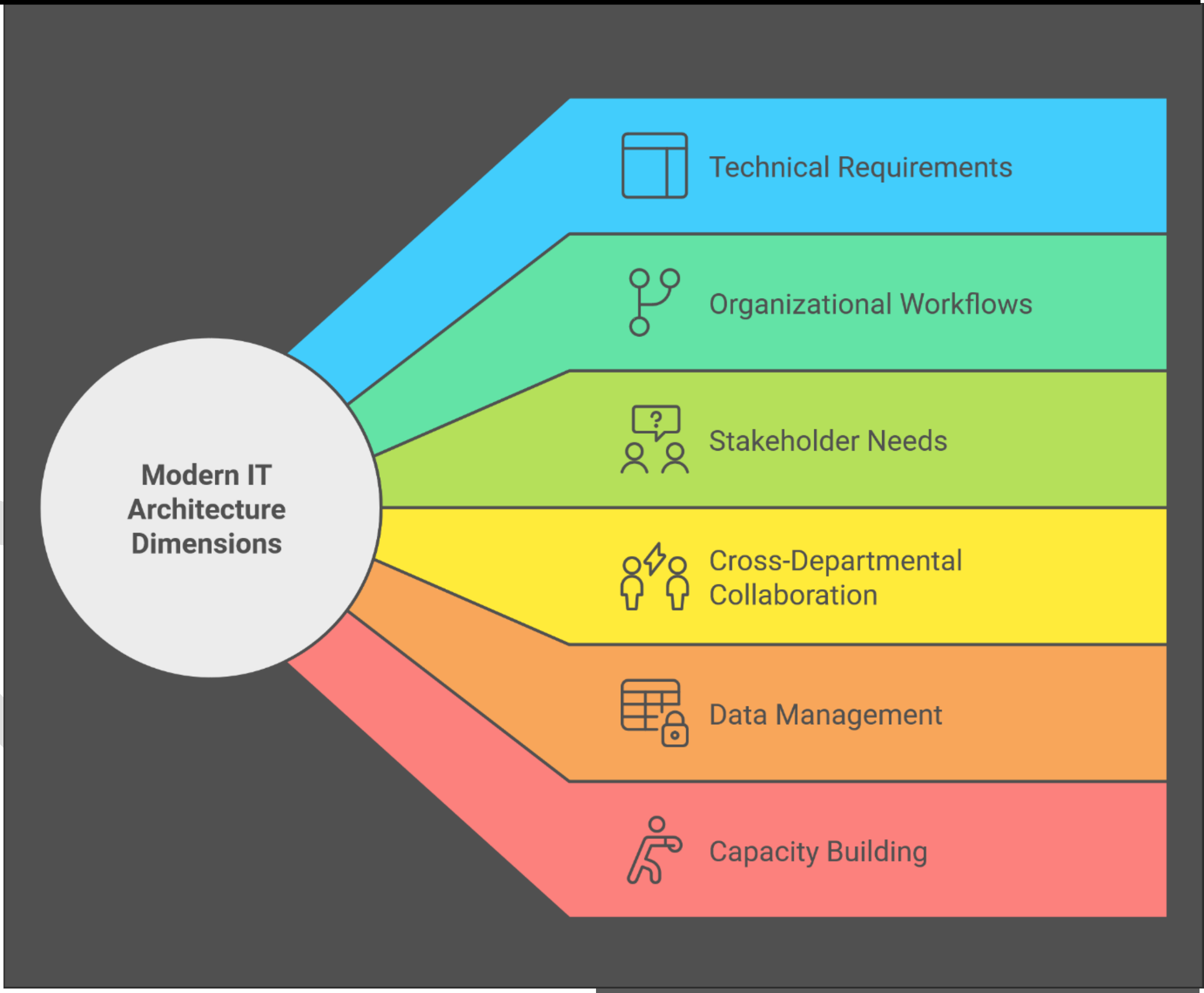
2025

Technical Document to Modernize IT architecture in NSO/NSS using open-source technology stack



6/18/2025

ABOUT THIS DOCUMENT:

This document is meant to be a living document that will change along with changes in an ever-evolving IT world. This document is developed by UN Statistics Division (UNSD) under the "Data For Now" and the "Collaborative on the use of Administrative Data for statistics". It consolidates concepts and tools to provide a basis to assess feasibility for modernizing IT architecture to incorporate new data sources at national statistical offices (NSOs) using free and open-source technology stacks and to help implement practical deployment of select components of a minimum data lake technology stack. We have tried to make this a practical document to support IT teams in NSOs, keeping it short and to the point - with the potential risk that some information is omitted or not researched well enough. Please note this space is fast evolving and therefore this document should be used in conjunction with additional resources to meet your organizations' needs.

While statistical processes such as data cleaning, modeling, anonymization, quality assurance, standardization, and record linkage, along with statistical methodologies, are crucial components of the statistics data lifecycle, they are not the focus of this document.

The views expressed in this wiki document are those of the authors and do not necessarily represent the views of UNSD, the United Nations, or any of its affiliated organizations. For list of any errors or omissions, please contact statistics@un.org.

ACKNOWLEDGEMENTS:

Document lead: Samrat Maskey

Document authors: Samrat Maskey, Thomas Aristide, Luis Gerardo Gonzalez Morales

Overall guidance: Luis Gerardo Gonzalez Morales, Sean Lovell, Vibeke Oestreich Nielsen, Faryal Ahmed, Eric Aloysius Jacobus Johannes Deeben …

Content refined: Generative AI tools including ChatGPT, Microsoft Copilot, Claud

Version: Draft 1

Table of Contents

List of Figures

Executive Summary

This guide supports National Statistical Offices (NSOs) in modernizing their IT architecture using a modular, open-source data lake technology stack. The framework was developed under the "Data for Now" initiative along with experience shared through the “Collaborative on use of Admin Data for statistics” to help NSOs integrate innovative data sources alongside traditional statistical data, enabling the production of timely, disaggregated, and high-quality statistics for sustainable development.

The Challenge
Today’s statistical offices are expected to do more than just run censuses and surveys. They need to tap into new/existing data sources like admin data, mobile phone records, satellite images, social media, and more to use them for statistics. To handle this volume, variety, and veracity (3V’s) of data, NSOs need a modern, secure, and interoperable IT infrastructure that supports the efficient processing, integration, and analysis of multiple data types.

Proposed Solution
The document outlines a modular, scalable data lake architecture built on Free and Open-Source Software (FOSS) principles. The minimum technology stack used includes:

- **Apache NiFi** for data ingestion and pipeline management
- **MinIO** for secure, scalable object storage
- **JupyterHub** for collaborative data processing and analysis
- **Trino** for flexible data virtualization
- **Kubernetes** for container orchestration and scalability
- **Active Directory** (if already in use) integration with the tools for identity and access management

Implementation Experience
The framework has been successfully tested and deployed across multiple NSOs including:

- Colombia (DANE) and Senegal (ANSD) - initial assessments and requirements identification for data lake platform (DANE using Hadoop; ANSD using proposed open-source stack)
- Vietnam (GSO), Namibia (NSA), Tunisia (INS) - Kubernetes deployments
- Maldives (MBS) - direct server deployment of storage

Key Benefits
<u>Technical Advantages</u>

- Cost-effective: Prioritizes FOSS solutions while remaining open to complementary proprietary tools
- Scalable: Architecture grows from single-node to multi-node clusters based on organizational needs
- Flexible: Supports both structured and unstructured data formats
- Secure: Implements robust authentication, authorization, and encryption mechanisms

<u>Organizational Impact</u>

- Enhanced Data Processing: Improves efficiency through compressed storage formats like Parquet, ORC, etc.
- Collaborative Environment: Enables cross-team collaboration through shared notebook environments.
- Capacity Building: Strengthens in-house IT team capabilities for long-term sustainability.

Implementation Requirements

Infrastructure Specifications

Minimum hardware requirements scale based on concurrent users:

- Small deployment (5 users): 16 CPU cores, 32GB RAM, 500GB storage
- Medium deployment (25 users): 32 CPU cores, 128GB RAM, 1.5TB storage
- Large deployment (100 users): 128 CPU cores, 512GB RAM, 4TB storage

Essential Skills

- System administration (Linux/Ubuntu)
- Networking and security
- Infrastructure management and containerization
- Scripting and automation
- Data management and version control

Implementation Approach

A phased deployment strategy is recommended, allowing NSOs to gradually adopt components based on priorities and capacity. The document provides:

- Current state assessment using "Data4Now: IT Guiding Questionnaire" (Annex) and identify hardware, software, skillset requirements.
- Target architecture vision and verification if proposed IT architecture meets the needs.
- Detailed technical implementation guide with checklist.
- Skills development recommendations
- Good practices for data organization and access management

Future Considerations

As organizations mature, the architecture can incorporate additional components including:

- Data discovery and cataloging tools
- Advanced observability and monitoring
- Workflow management systems
- Artificial Intelligence and Machine Learning platforms

Conclusion

This IT modernization framework provides NSOs with a practical, tested approach to building modern data infrastructure. By prioritizing open-source solutions, emphasizing capacity building, and maintaining flexibility for future growth, the framework enables statistical offices to meet evolving data demands while maintaining operational efficiency and data security. The successful implementations across multiple countries demonstrate the framework's adaptability to diverse organizational contexts and requirements.

1 Introduction

1.1 Background

In this modern data age, the role of statistical offices has evolved significantly with the rapid growth of digital technologies and the emergence of innovative data sources. Modern IT infrastructure has become essential for statistical offices to collect, process, analyze, and disseminate both traditional (census, survey, etc.) and non-traditional (administrative data, earth observation, mobile phone, social media, sensors, etc.) data, which usually come in different sizes and formats.

Guided by the 'Data for Now' initiative, we developed and tested a modular IT architecture based on a minimum data lake technology stack, with the aim to support the implementation of a platform (like data innovation lab) to process innovative data and methods to produce statistical indicators. It is based on experience from technical implementation of the Data for Now initiative (Data4Now) and Development Accounts 13 (DA-13) in different countries. The work has also benefited from the discussions with partners in task-team 3 (Technical interoperability and linking) of the Collaborative on administrative data (CAD), UN Global Platform and the UN Economic and Social Commission for Asia and the Pacific (UNESCAP).

The work began under Data4Now initiative, which supports members of the National Statistical Systems (NSS) in participating countries to leverage innovative sources, technologies and methods for the streamlined production and dissemination of better, more timely and disaggregated data for sustainable development. During initial assessments to support the NSOs of Colombia (DANE) and Senegal (ANSD) on the technology front, both countries independently identified similar cross-cutting IT requirements, including the need for a data lake platform capable of storing diverse set of datasets irrespective of their maturity level and technology specification. In parallel, work on IT Architecture workstream under CAD compiled use-case of deployed IT Architecture along with tools and technology used for collecting administrative data in NSOs from Mexico (INEGI), Uruguay (INE), Colombia (DANE), Norway (Statistics Norway), Namibia (NSA) helped understand actual IT architectures implemented at different NSOs. Building on these findings, a data lake technology stack was proposed to new Data4Now participating NSOs, with deployment platforms and technology stacks tailored to their specific requirement. To facilitate this process, a 'Data4Now: IT Guiding Questionnaire' was developed to Identify country-specific requirements and challenges, particularly focusing on data flows within the technology landscape. This document, along with the questionnaire provides a basis to assess feasibility for new NSO aspiring to deploy a data lake architecture.

A key consideration in deploying this architecture is aligning it with the organizations' IT strategic roadmap and establishing a skilled IT team to manage the administrative and platform operations. This architecture not only strengthens the data engineering capabilities of IT teams within NSO but also provides a Data Innovation Lab with robust and flexible foundation for integrating innovative data sources and advanced data science methods into the production of official statistics.

The proposed data lake configuration represents a minimum setup, designed to balance deployment and management efforts while remaining open to integration of additional tools and components based on specific needs. The select technology stack was initially implemented in NSO of Senegal (ANSD) using Docker Compose and later using Kubernetes in NSOs of Viet Nam (GSO), Namibia (NSA), and Tunisia (INS), offering enhanced scalability, maintainability, and resilience while remaining adaptable to specific needs of each NSO. However, in the case of NSO of Maldives (MBS), one of the components was deployed directly in the server. There were gradual enhancements in the implementation platform from docker to Kubernetes to multi-node Kubernetes cluster to manage additional resources.

To ensure sustainability, Free and Open-Source Software (FOSS) principle is prioritized for its cost-effectiveness, adaptability, and flexibility to evolve with technological advancements. For instance, **Apache NiFi** is used for data ingestion as it provides a visual platform to create data pipelines with minimal manual coding. **MinIO** provides a secure, scalable, and efficient data storage platform that enables storage of new and innovative data formats while being simple to deploy and maintain. **JupyterHub** facilitates advanced data processing workflows through collaborative notebook environments using python and R, and **Trino** supports flexible data virtualization. 'Figure 1-1 Minimum data lake technology stack used in Data4Now' highlights each stage of data flow and respective technology stacks. We will explore each of these stages in coming sections.

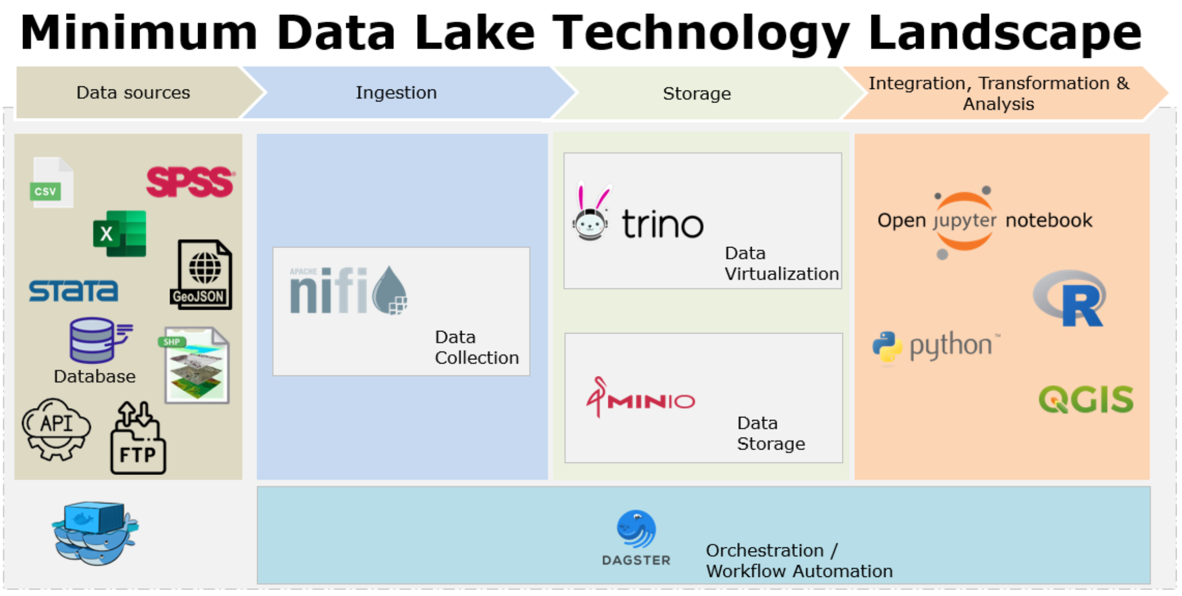


FIGURE 1-1 MINIMUM DATA LAKE TECHNOLOGY STACK USED IN DATA4NOW

While FOSS forms the backbone of the architecture, proprietary solutions developed by private vendors are not excluded when they align with specific needs. For example, **Active Directory (AD)**, a widely used proprietary identity and access management system, reinforces security for many NSOs. They can choose to use existing AD with these tools and reduce the burden of having to manage separate identity and access management systems. FOSS and proprietary solutions should be seen as complementary, with their advantages and limitations carefully evaluated based on the unique context and operational requirements of each NSO.

Brief overview of these tools and additional FOSS alternatives can be found in ‘Few components of open technology stack for data’. ‘Basic requirements for minimum data lake technology stack used in Data4Now’ outlines the platform (hardware, software) and skill-set requirements for deploying and managing the selected minimum data lake technology stack. Finally, ‘Technical implementation guide for minimum data lake technology stack used in Data4Now’ provides hands-on guidance for deployment of the proposed architecture tested by Data4Now team.

Before moving to the next section, it is important to highlight that effective modernization extends beyond technical infrastructure to encompass organizational workflows, stakeholder requirements, and interdepartmental collaboration. Aligning the proposed architecture with an organization’s overall IT strategy enhances data management, capacity building, governance, and long-term scalability. This is also highlighted in the figure below.

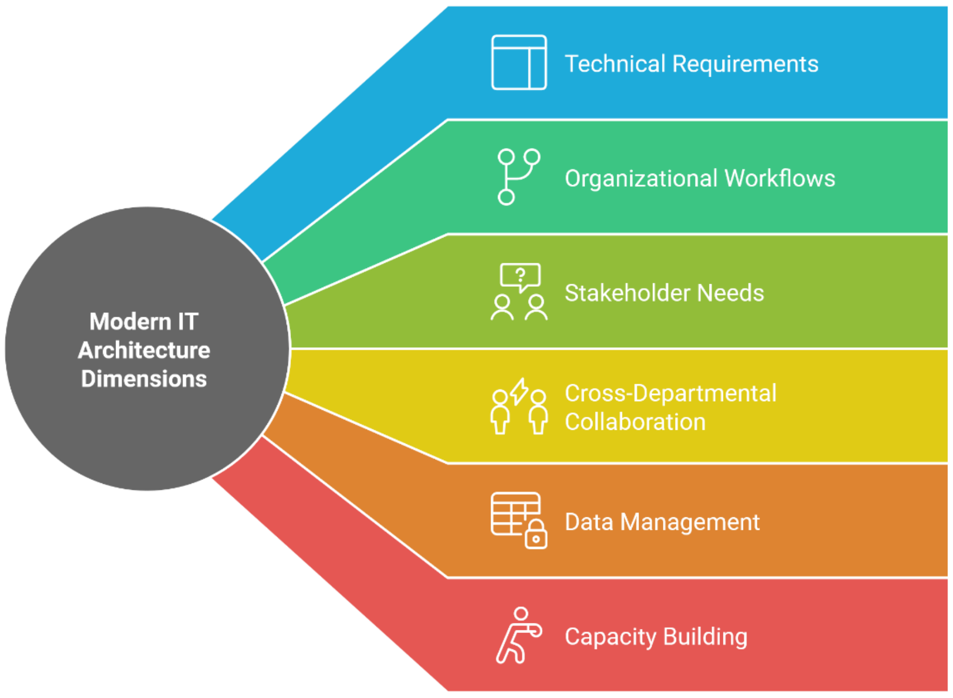


FIGURE 1-2 UNVEILING MODERN IT ARCHITECTURE DIMENSIONS

Implementing mature and stable technologies with a focus on automation, reliability, and efficient scaling is important and needs to be planned from the beginning. Continuous enhancement, monitoring, and testing of these tools to integrate into the overall solution is essential to ensure smooth transitions as we modernize our infrastructure.

1.2 Goals and objectives

The primary goal of this document is to support the modernization of IT architecture at NSO to incorporate new data sources by providing information on the design and implementation of modular, scalable, and resilient data lake architecture. This IT architecture is designed to integrate innovative data sources supporting both structured and unstructured data formats while improving data processing efficiency by utilizing storage formats that store compressed data. It also serves as a data innovation lab to support implementing advanced data science methodologies, enabling the production of timely, disaggregated, and high-quality data for sustainable development. Grounded in the principles of sustainability, inclusivity, and innovation**,** the proposed architecture prioritizes the use of FOSS while remaining open to complementary proprietary tools where necessary, ensuring efficient processing, management, and dissemination of diverse data sources.

Specific Objectives:

1. **Consolidate foundational knowledge:** The document provides an overview of tools and technologies in the data lifecycle, from data ingestion and storage to processing and dissemination with a focus on using innovative data sources. It highlights the benefits and limitations of few of these tools, offering practical insights to guide informed decision-making.
2. **Build institutional capacity**: Emphasis is placed on fostering the skills and self-reliance of in-house IT teams to manage and evolve these systems, ensuring long-term sustainability and resilience. This document highlights hardware, software and skillset requirements needed to deploy, manage, and use the modern IT architecture. It also documents the deployment process of the minimum data lake technology stack so the IT team can follow the process and evolve as needed.
3. **Aligning with strategic goals**: Document focuses on aligning IT systems with national strategies, enhancing workflows, fostering interdepartmental collaboration, and addressing global and national development priorities.
4. **Incorporate lessons and good practices**: Practical examples drawn from initiatives such as Data4Now and DA-13 are provided to support scalable and context-specific implementations for NSO, aligning with global efforts to promote innovation, efficiency, and collaboration.

1.3 Guiding principles for the modernization of IT architecture

The following Data4Now principles guide IT modernization:

1. **Sustainability:** Leverage free and open-source technologies supported by active communities for long-term viability and adaptability while reducing reliance on proprietary solutions. Train IT staff on managing and using the deployed tools.
2. **Holistic perspective:** Adopt a comprehensive approach to IT modernization that promotes innovation, integrating data workflows, tools, and processes to create a cohesive and efficient statistical system.

3. **Production-ready solutions:** Implement robust and reliable open-source tools that are capable of handling real-world workloads, ensuring scalability and operational efficiency.
4. **Privacy and confidentiality:** Ensure compliance with data protection standards to safeguard sensitive information, enabling secure data analysis and sharing.

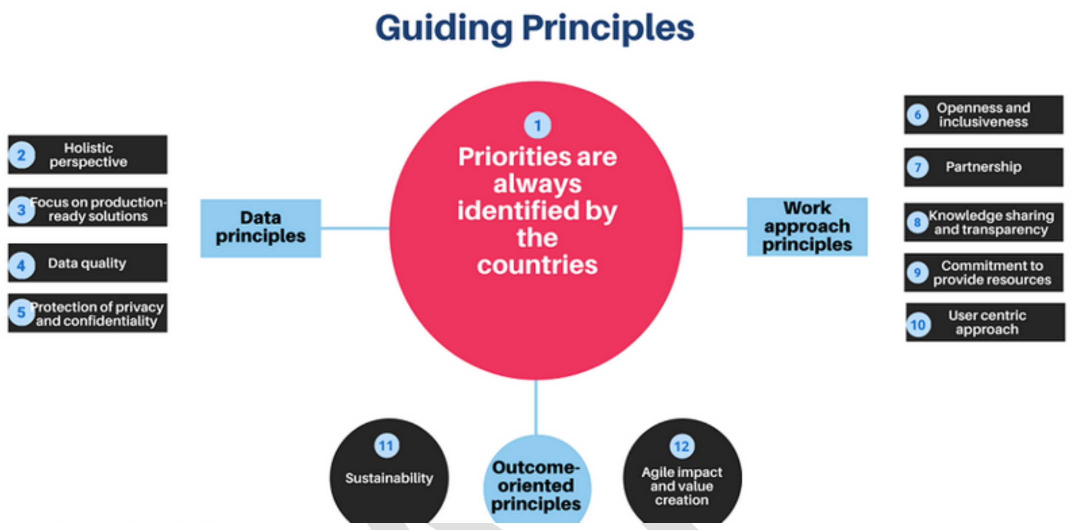

Guiding Principles
1 Priorities are always identified by the countries
Data principles
2 Holistic perspective
3 Focus on production-ready solutions
4 Data quality
5 Protection of privacy and confidentiality
Work approach principles
6 Openness and inclusiveness
7 Partnership
8 Knowledge sharing and transparency
9 Commitment to provide resources
10 User centric approach
Outcome-oriented principles
11 Sustainability
12 Agile impact and value creation

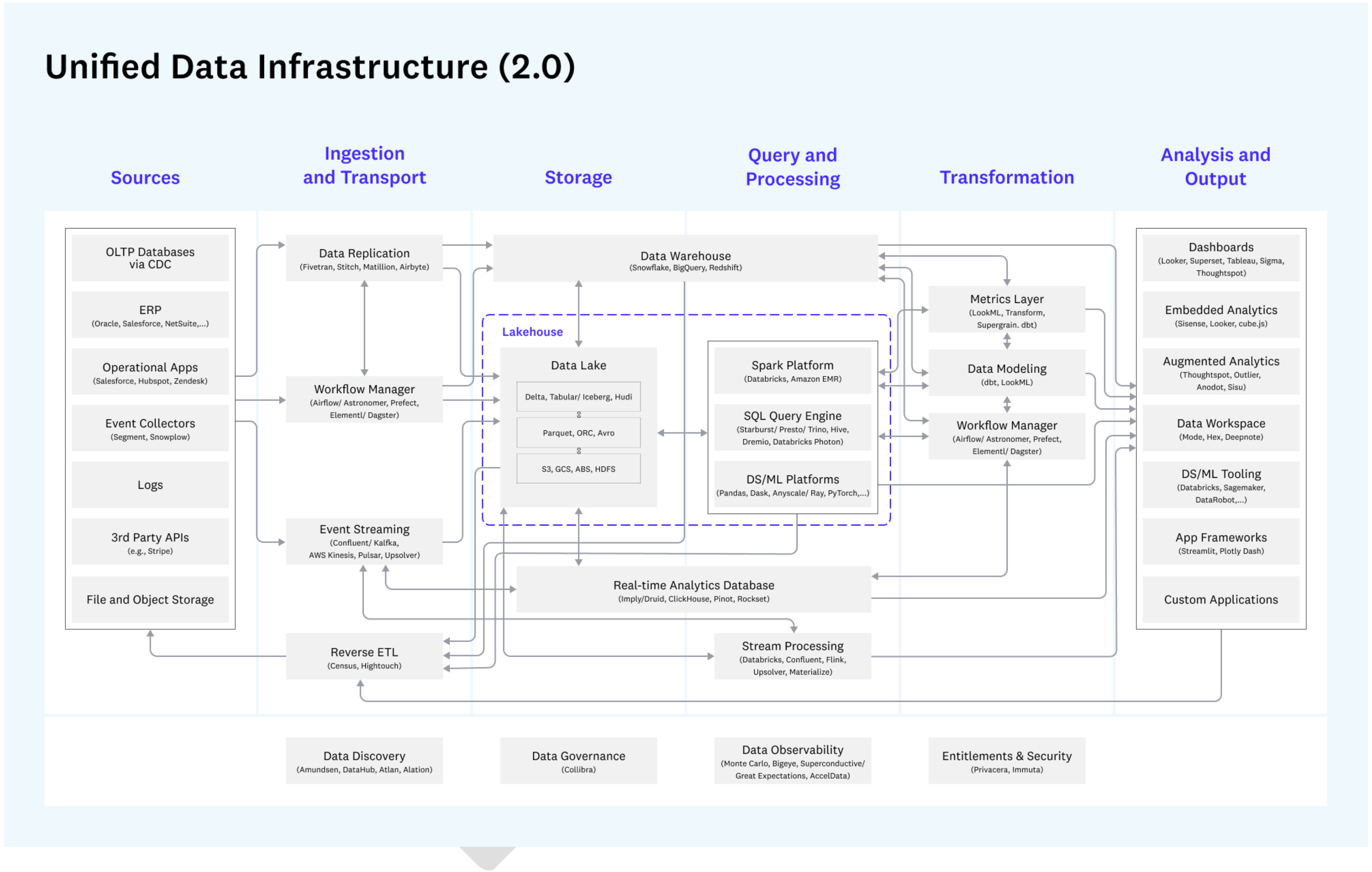
FIGURE 1-3 DATA FOR NOW GUIDING PRINCIPLE

1.4 Reference architecture for data innovation

The rapid evolution of IT architecture, driven by innovative data sources, technological advancements, and changing organizational requirements, highlights the importance of establishing a reference architecture for data innovation. Such architecture provides a strategic framework for designing workflows that encompass data collection, storage, processing, analysis, and dissemination. It promotes flexibility and scalability to address emerging challenges, such as handling different formats of data along with the shift from traditional Extract, Transform, Load (ETL) processes to more adaptive Extract, Load, Transform (ELT) models.[1]

The updated unified data infrastructure diagram (shown below) provides an overview of organizational data flows and incorporates practical recommendations on tools and platforms utilized by leading data organizations. While not all elements may be implemented at NSOs, the framework offers an adaptable foundation for the modernization efforts, particularly within initiatives like Data4Now. As it progresses, it could then incorporate Artificial Intelligence (AI) and Machine Learning (ML) capabilities.

[1] This perspective draws on insights from "Emerging Architectures for Modern Data Infrastructure," authored by Matt Bornstein, Martin Casado, and Jennifer Li at Andreessen Horowitz. The authors provide an overview of trends and best practices shaping modern data infrastructure.

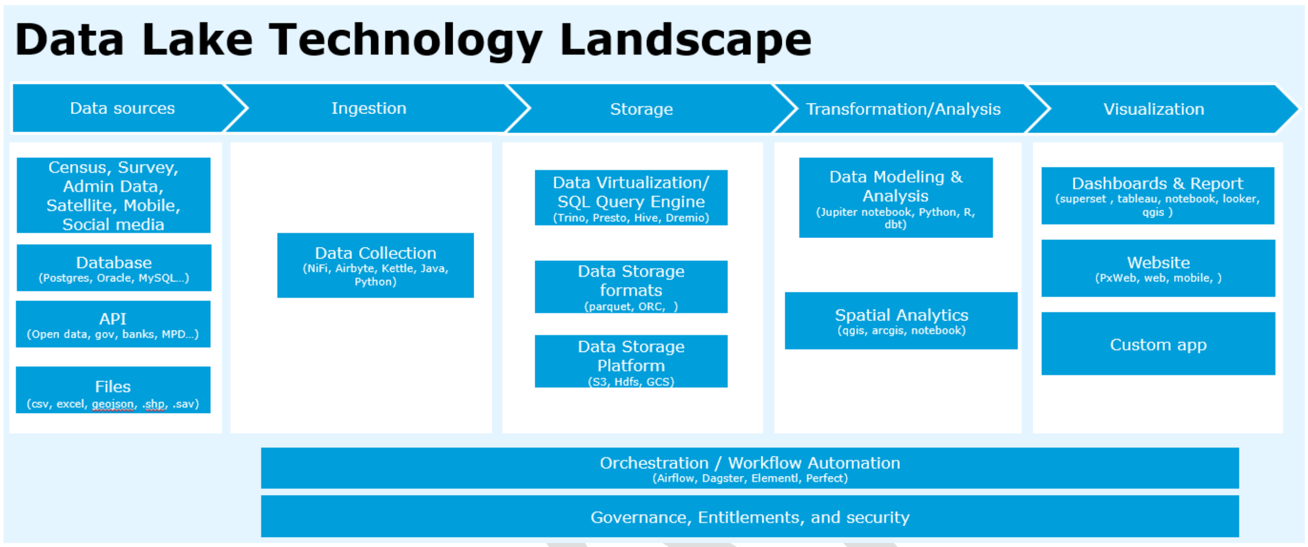


Unified Data Infrastructure (2.0)
Sources
Ingestion and Transport
Storage
Query and Processing
Transformation
Analysis and Output
OLTP Databases via CDC
ERP (Oracle, Salesforce, NetSuite,...)
Operational Apps (Salesforce, Hubspot, Zendesk)
Event Collectors (Segment, Snowplow)
Logs
3rd Party APIs (e.g., Stripe)
File and Object Storage
Data Replication (Fivetran, Stitch, Matillion, Airbyte)
Workflow Manager (Airflow/ Astronomer, Prefect, Elementl/ Dagster)
Event Streaming (Confluent/ Kalfka, AWS Kinesis, Pulsar, Upsolver)
Reverse ETL (Census, Hightouch)
Data Warehouse (Snowflake, BigQuery, Redshift)
Lakehouse
Data Lake
Delta, Tabular/ Iceberg, Hudi
Parquet, ORC, Avro
S3, GCS, ABS, HDFS
Spark Platform (Databricks, Amazon EMR)
SQL Query Engine (Starburst/ Presto/ Trino, Hive, Dremio, Databricks Photon)
DS/ML Platforms (Pandas, Dask, Anyscale/ Ray, PyTorch,...)
Real-time Analytics Database (Imply/Druid, ClickHouse, Pinot, Rockset)
Stream Processing (Databricks, Confluent, Flink, Upsolver, Materialize)
Metrics Layer (LookML, Transform, Supergrain. dbt)
Data Modeling (dbt, LookML)
Workflow Manager (Airflow/ Astronomer, Prefect, Elementl/ Dagster)
Dashboards (Looker, Superset, Tableau, Sigma, Thoughtspot)
Embedded Analytics (Sisense, Looker, cube.js)
Augmented Analytics (Thoughtspot, Outlier, Anodot, Sisu)
Data Workspace (Mode, Hex, Deepnote)
DS/ML Tooling (Databricks, Sagemaker, DataRobot,...)
App Frameworks (Streamlit, Plotly Dash)
Custom Applications
Data Discovery (Amundsen, DataHub, Atlan, Alation)
Data Governance (Collibra)
Data Observability (Monte Carlo, Bigeye, Superconductive/ Great Expectations, AccelData)
Entitlements & Security (Privacera, Immuta)

FIGURE 1-4 UNIFIED DATA INFRASTRUCTURE ARCHITECTURE 2.0 - ANDREESSEN HOROWITZ

Core Architecture: The Data Lake

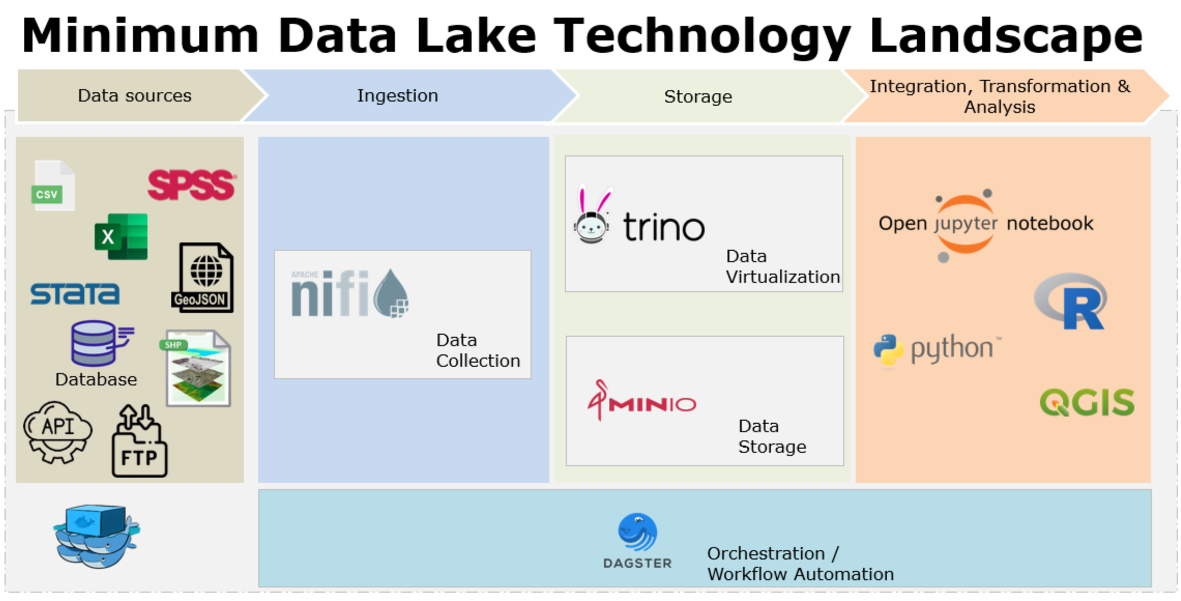
To support NSO, a simplified reference data lake architecture has been proposed below in 'Figure 1-5 Simplified reference data lake Architecture', comprising logical layers and technical landscape focused on scalability and user-centric design. This framework prioritizes the integration of FOSS while recognizing the importance of strengthening existing capabilities with proprietary tools where necessary. While this architecture remains under development, specific attention is being directed toward security and access control to ensure operational readiness.



Data Lake Technology Landscape
Data sources
Ingestion
Storage
Transformation/Analysis
Visualization
Census, Survey, Admin Data, Satellite, Mobile, Social media
Database (Postgres, Oracle, MySQL…)
API (Open data, gov, banks, MPD…)
Files (csv, excel, geojson, .shp, .sav)
Data Collection (NiFi, Airbyte, Kettle, Java, Python)
Data Virtualization/ SQL Query Engine (Trino, Presto, Hive, Dremio)
Data Storage formats (parquet, ORC,)
Data Storage Platform (S3, Hdfs, GCS)
Data Modeling & Analysis (Jupiter notebook, Python, R, dbt)
Spatial Analytics (qgis, arcgis, notebook)
Dashboards & Report (superset , tableau, notebook, looker, qgis)
Website (PxWeb, web, mobile,)
Custom app
Orchestration / Workflow Automation (Airflow, Dagster, Elementl, Perfect)
Governance, Entitlements, and security

FIGURE 1-5 SIMPLIFIED REFERENCE DATA LAKE ARCHITECTURE

"Figure 1-1 Minimum data lake technology stack used in Data4Now" serves as an example with selected tools to design this data lake architecture while exploring customizations based on organizations requirements. We will explore in following chapters some of the requirements to deploy these tools including the importance of a strong IT team and practical deployment steps.



This architecture (Figure 1.4-2) divides the data lifecycle into five interconnected stages:

1. **Data source**

A "data source" can refer both to the original point of data generation and to the system where the data is subsequently stored or made available. In the statistical context, "data source" often refers to the instrument or system used to generate data. These sources include traditional statistical instruments like censuses and sample surveys, as well as administrative records, business registers, and non-traditional sources such as mobile phone data, Earth observation (satellite data), social media, e-commerce records, remote sensor data, etc. In this document, we will classify data sources based on the format in which the data is accessed, such as database connections, data files or API connections, and frequently refer to the specific formats in which the data is serialized (e.g., JSON, XML, CSV, DAT, Parquet, etc.).

2. **Data ingestion**

Data ingestion is the process of importing, acquiring, or transferring data from various data sources into a system where it can be processed and analyzed. It involves connecting to diverse sources of data discussed above using the format in which the data is accessed. This may involve using APIs, database connections, file uploads, or other methods and could use real time data using streaming or schedule data upload using batch. Additionally, the data may come in various formats like JSON, CSV, Parquet, etc. Some popular tools for data ingestion include Apache NiFi, Airflow, Pentaho PDI, etc. along with code options like java, python script, etc.

3. **Data storage**

Data storage is the process of storing collected data in a secure and organized manner to allow for efficient retrieval, management, and future use. Data stored should be capable of managing large volumes of structured and unstructured data. The choice of storage depends on the data format, volume, and accessibility requirements. Options include relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Cassandra), or file-based systems like Hadoop Distributed File System (HDFS) or MinIO. Even in file-based systems, large datasets can be stored in compressed row or column-oriented storage like Parquet, ORC, etc. and further virtualized using tools like Hive, Trino, Presto, etc.

4. **Data processing**

Data processing refers to the transformation of raw data into meaningful and usable forms for analysis, modeling, or reporting. It often involves cleaning, validating, anonymizing, linking, aggregating, or transforming data collected from various sources. Processing pipelines vary depending on whether the data is structured, semi-structured, or unstructured. Common steps include deduplication, missing data handling, integration from multiple sources, and harmonization into standardized formats. Some of the common tools used include Python libraries (pandas, NumPy, etc.), R, STATA, SPSS, Apache Spark, QGIS, etc.

5. **Data dissemination**

Data dissemination is the process of sharing processed data or statistical outputs with stakeholders, end-users, or the public in a format that is interpretable, accessible, and actionable. Dissemination ensures that the collected and processed data is available for decision-making, research, or public use. This involves making datasets, dashboards, or reports available via platforms like websites, APIs, interactive portals, or file distribution systems. Security and privacy considerations, such as anonymizing sensitive data, are key to dissemination strategies. Apache superset, PowerBI, Looker studio, Metabase, etc. are some of the popular dashboards and PxWeb, CKAN, GeoServer, etc. are few among many of the dissemination tools.

Conclusion

Each phase of this reference architecture for data innovation is crucial to building a modern statistical framework that ensures seamless data flow from sources to actionable insights or dissemination. This approach enhances efficiency, adaptability, and relevance, supporting the mandate of NSO to provide high-quality, timely, and reliable statistics for sustainable development. In addition to this, there are components like data-discovery, observability, workflow manager, AI/ML platform, etc. that should be integrated as the system becomes mature.

2 Few components of open technology stack for data

2.1 Overview

The open technology stack for data comprises a suite of open-source tools and open-standard technologies designed to support the innovative data lifecycle, from collection and processing to dissemination and reuse. It facilitates organizations to modernize their data infrastructure by enhancing tool integration, promoting interoperability, and reducing reliance on proprietary solutions thereby mitigating vendor lock-in. The modular design of the stack fosters sustainability, strengthens local capacity, and ensures long-term viability for NSO.

A modern open technology stack for data aligns closely with the principles of Data4Now, which emphasize open standards, modular design, and better integration. This alignment makes open technology stack for data as an ideal technical foundation for implementing efficient, interoperable, and scalable data infrastructure tailored to the unique requirements of NSO. However, building a robust and effective open technology stack for data presents significant challenges, particularly given vast and fragmented landscape of tools and technologies with overlapping functionalities. In the below '<u>Figure 2-1 Overview of select tools discussed in different components of the data value chain</u>', we have grouped and highlighted few of these FOSS tools.

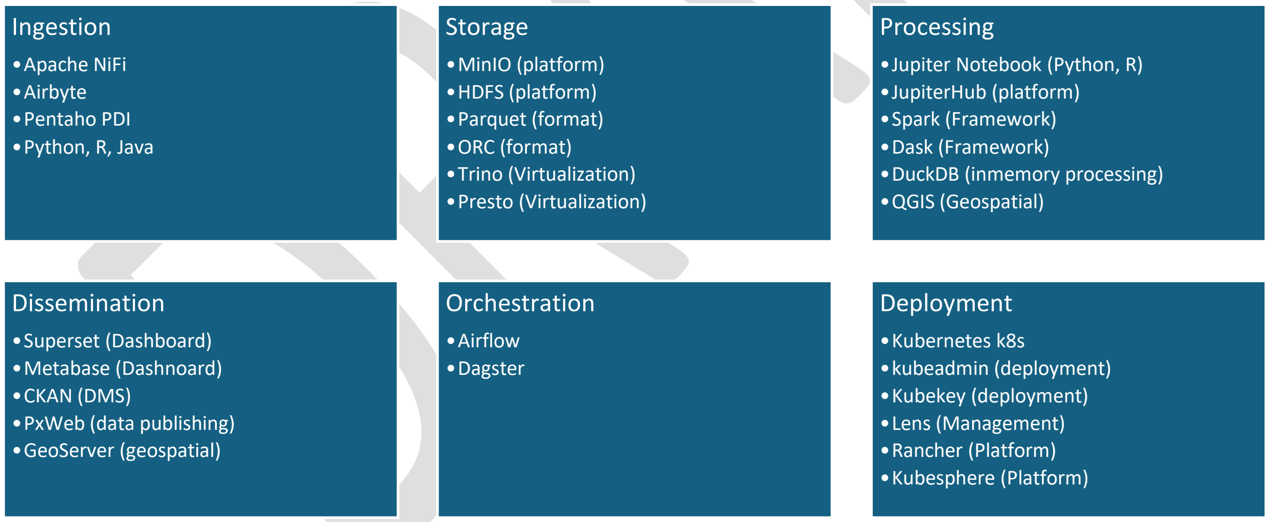


Figure 2-1 Overview of select tools discussed in different components of the data value chain

In the following sections, we will examine few select components along with tools shown in the above figure, categorized according to their function within the data value chain.

2.2 Data ingestion/collection

Data ingestion is the process of importing, acquiring, or transferring data from various sources. This data is then processed and analyzed within a system. It involves connecting to diverse sources of data such as surveys, administrative records, or non-traditional data sources like mobile phone or satellite data and retrieving the data. Few considerations for this layer, along with relevant tools, are discussed below.

2.2.1 Few considerations:

The diagram below highlights few considerations for data collection.

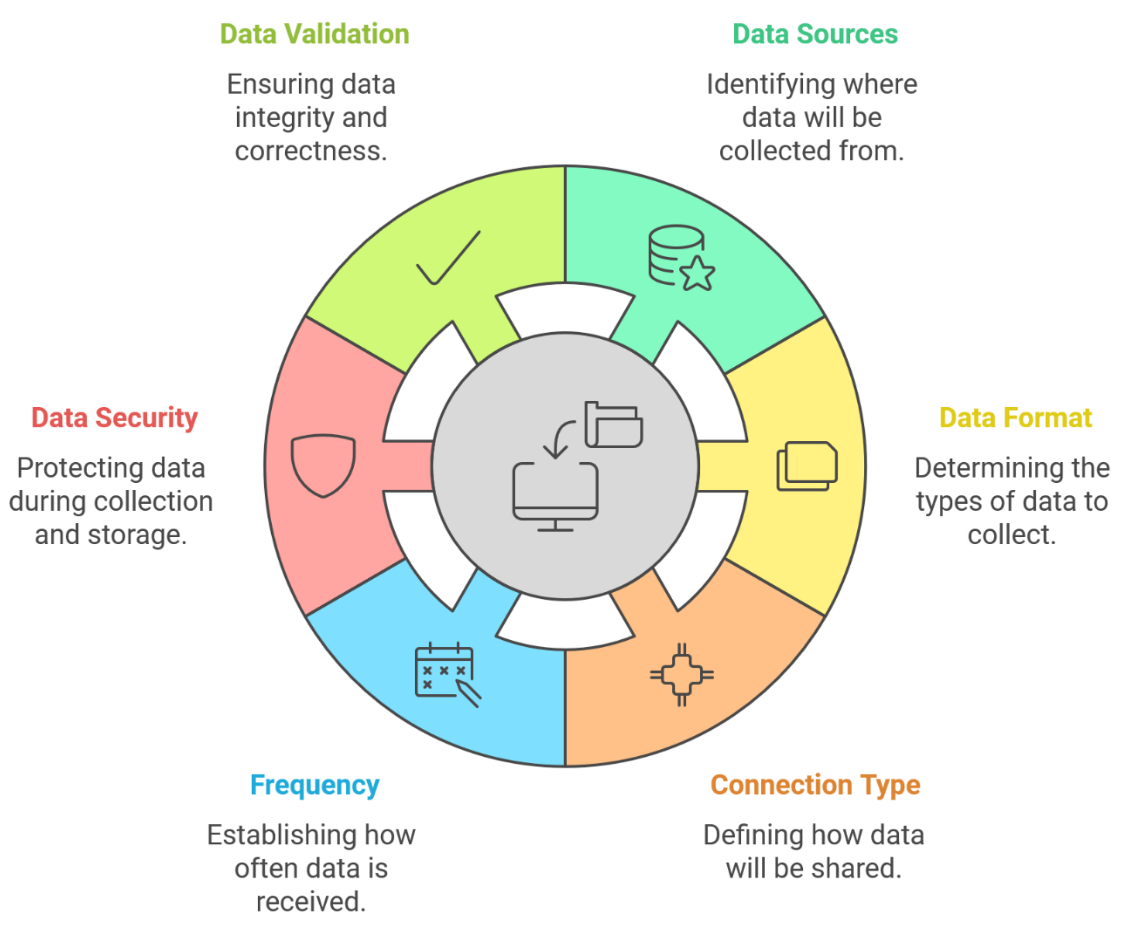

Data Validation
Ensuring data integrity and correctness.
Data Sources
Identifying where data will be collected from.
Data Security
Protecting data during collection and storage.
Data Format
Determining the types of data to collect.
Frequency
Establishing how often data is received.
Connection Type
Defining how data will be shared.

FIGURE 2-2 DATA INGESTION CONSIDERATIONS

1. **Data Sources**: Identifying the sources from which the data will be collected is crucial. Some examples of data sources are census, surveys, admin-data, citizen data, news and social media, satellite, mobile phone data, e-commerce data, and more. Establishing Memorandums of Understanding (MOUs) with data providers may be necessary to ensure legal and ethical data sharing.
2. **Data Format**: Determining the types of data to collect is important, such as structured data (e.g., tables, spreadsheets) or unstructured data (e.g., text, images, audio, video). For large datasets, efficient formats like Parquet and OCR should be considered. For statistical data,

SDMX (Statistical Data and Metadata eXchange) format is recommended for data exchange along with DDI (the Data Documentation Initiative) for capturing metadata.

3. **Connection Type**: Defining how data will be shared is equally important. Common connection types include sFTP, database connections, data APIs, application APIs, web upload, email, and HTTP (web scraping).
4. **Frequency**: Determining how often data is collected, is important. Data may be received daily, weekly, monthly, quarterly, annually, or as a one-time process.
    a. **Batch Processing**: Data is collected, processed, and delivered in predefined batches at scheduled intervals. This is suitable for handling large volumes of data and performing resource-intensive operations.
    b. **Real-time Streaming**: Data is continuously ingested and processed as it arrives. This is vital for applications requiring real-time analytics or monitoring, such as disaster response.
5. **Data Security**: Implementing security measures is vital to protect the data during collection, transmission, and storage. It includes encryption protocols (e.g., SSL/TLS with HTTPS, SFTP, SSH) encrypted tunnels (e.g. VPNs) and an identity verification system to prevent unauthorized access.
6. **Data Validation**: Establishing validation processes to ensure data integrity and accuracy is crucial to avoid errors and inaccuracies.

2.2.2 Data ingestion tools

In the context of data ingestion, low-code tools and traditional coding languages play important roles in simplifying the process of moving and processing data from various sources into a data pipeline.

Low-code tools: These platforms enable users to create data pipelines and workflows with minimal manual coding. They provide visual interfaces and pre-built components, making it easier to design, configure, and automate processes. Low-code tools are particularly useful for users without extensive programming knowledge but those who need to manage data workflows effectively. Examples of FOSS tools used by various NSO for data ingestion include:

1. **Apache NiFi**: A user-friendly tool for real-time and batch data flows, offering flexible connectors and transformations.
2. **Airbyte**: A modern ELT platform with an extensive library of prebuilt connectors.
3. **Pentaho PDI**: A data integration tool that enables ETL from a variety of sources.

A quick comparison of few of these tools is shown in the table below:

Feature	Apache NiFi	Pentaho PDI	Airbyte
License	Open Source (Apache)	Commercial (Enterprise) / Open Source (Community)	Open Source (MIT)
UI Experience	Web-based flow designer with drag and drop	Desktop application with drag and drop	Web-based configuration UI

Feature	Apache NiFi	Pentaho PDI	Airbyte
Learning Curve	Moderate to steep	Moderate	Low
Connectors/Sources	300+ processors	100+	300+ connectors
Transformation Capabilities	Moderate	Excellent	Limited (relies on destination)
Scalability	Highly scalable	Good	Good
Scheduling	Built-in	Built-in	Built-in
Cloud Support	Yes	Yes	Yes (cloud-native)
Community Activity	Very active	Moderate	Very active

Custom code: Writing custom code provides greater flexibility and customization for data ingestion processes. While it requires technical expertise, it allows for tailored solutions to meet specific requirements. Common languages used for data ingestion include:

1. **Custom Python/R Scripts**: Flexible and widely used for bespoke data process.
2. **Custom Java/C#/etc.**: Suitable for building custom data ingestion pipelines with high performance and scalability.

Additionally, a use case for Apache NiFi will be published separately to demonstrate how countries can implement this tool effectively.

2.3 Data storage and management

Data storage refers to the system and processes used to store data collected from various sources or generated through analytical processes, either in raw or aggregated formats. It involves the organization, management, and retrieval of data assets to support various applications and services. Selecting an appropriate storage mechanism is an important aspect of modern IT architecture design and operation. Mature data architecture includes data-warehouses alongside supporting data lake or data lake house architecture.

Data warehouses: Data warehouses store well-organized, transformed data with predefined schemas, ensuring quality and consistency through ETL processes. They use star or snowflake schemas, organizing data into dimensions and fact tables for efficient querying and analysis.

Data lakes: Data lakes store diverse, raw data without strict structuring, providing flexibility for various data types. Key characteristics include:

- As-is data ingestion promoting schema-on-read.
- Support for both structured, semi-structured, and unstructured data
- Scalability and adaptability for evolving analytical needs.
- Capability for exploratory analysis and advanced analytics.

Given the focus on innovative use of new and existing data, this section emphasizes data lake architecture. However, the choice of architecture depends on the specific needs of NSO. The modular design allows for flexibility and scalability. Proper data storage and access management practices are essential to ensure data security, privacy, and efficient processing. Below are few considerations, tools and strategies for data lake storage.

2.3.1 Few considerations in data lake storage:

1. **Storage platform**: Modern data lake typically uses traditional distributed file systems (e.g., Hadoop HDFS) or object storage systems (MinIO, Amazon S3, Azure Blob Storage, Google Cloud Storage, etc.). These platforms provide scalable, durable, and cost-effective storage for large volumes of data.
2. **Storage format**: Data can be stored in efficient formats like Parquet, AVRO, ORC, etc. which offer row- or column-oriented storage with compression and encryption. These formats save storage space and reduce access/load time compared to traditional data formats like CSV, JSON, SPSS, etc.
3. **Data access management**: Data lake can implement Role-Base Access Control (RBAC) or Policy-Based Access Control (PBAC) to manage access at different level. Fine-grained control can also be applied at the object level.
4. **Data organization strategy:** Data in the data lake can be organized in buckets using medallion architecture like [e.g., Raw, Anonymized/Bronze, Staged/Silver, Aggregate/Gold, etc.] and partitioned by meaningful attributes (e.g. year, location, or category, etc.) to improve query performance and usability.

2.3.2 Data storage platforms:

Modern data storage technologies for data lakes include a variety of tools and platforms that leverage advancements in cloud computing, distributed computing, and big data processing. These technologies aim to provide scalable, efficient, and cost-effective solutions for storing and managing data within a data lake. Here are some of the prominent data lake platforms that support on-premises deployment:

Distributed File System in data lake: Systems like Hadoop Distributed File System (HDFS) store and manage vast amounts of data across interconnected nodes. They are well-suited for batch-oriented processing and analytics, offering high throughput and fault tolerance. Many early adopters of big data technology use Hadoop ecosystems including DANE-Colombia. They could be costly to manage in terms of resource utilization.

Object storage: Platforms like MinIO, Amazon S3, and Azure Blob Storage store data as objects within containers, each with a unique identifier and metadata. Object storage is ideal for unstructured data and offers scalability, durability, and cost-effectiveness. Many new implementations include Object storage solution.

A quick comparison of few of these platforms is shown in the table below:

Feature	**MinIO**	**Hadoop HDFS**
Architecture	Object storage system	Distributed file system

Feature	**MinIO**	**Hadoop HDFS**
API Compatibility	Amazon S3 compatible	HDFS API
Data Structure	Object-based	Block-based
Deployment	Lightweight, container-friendly	Heavyweight, cluster-oriented
Scalability	Horizontal, containerized	Horizontal, requires name node planning
Performance	High throughput, optimized for small/large files	Optimized for large files, slower for small files
Fault Tolerance	Erasure coding, distributed design	Replication-based
Cloud-Readiness	Cloud-native design	Originally designed for on-premises
Complexity	Simple setup and maintenance	Complex setup and operational overhead
Ecosystem Integration	Works with S3-compatible tools	Native integration with Hadoop ecosystem
Use Cases	Modern cloud applications, microservices	Traditional big data processing (Hadoop ecosystem)
Security	IAM, encryption, RBAC	Kerberos, ACLs
Consistency Model	Strong consistency	Eventually consistent
License	Open source (AGPLv3)	Open source (Apache)

In the annex section "Data ingestion tools" you can see an example of python script that ingest data from an ftp server to and MinIO.

2.3.3 Data storage formats:

Modern data lake architectures support various storage formats to accommodate diverse data types and analytical needs. Some common formats include:

1. **Parquet**: A columnar storage format optimized for analytics offering efficient compression and encoding. Many countries have seen file size reduction of almost 90% when using Parquet with Snappy compression. See Annex 'Parquet'
2. **ORC (Optimized Row Columnar)**: A columnar storage format designed for high-performance analytics with features like predicate pushdown and lightweight compression.
3. **Avro**: A compact and efficient binary format supporting schema evolution and rich data types. Ideal for data serialization and exchange.
4. **JSON (JavaScript Object Notation)**: A human-readable format for semi-structured and unstructured data.
5. **CSV (Comma Separated Values)**: A simple, text-based format widely used for tabular data. While less efficient than columnar formats for analytics, CSV remains versatile and easy to work with.

A quick comparison of few of these formats is shown in the table below:

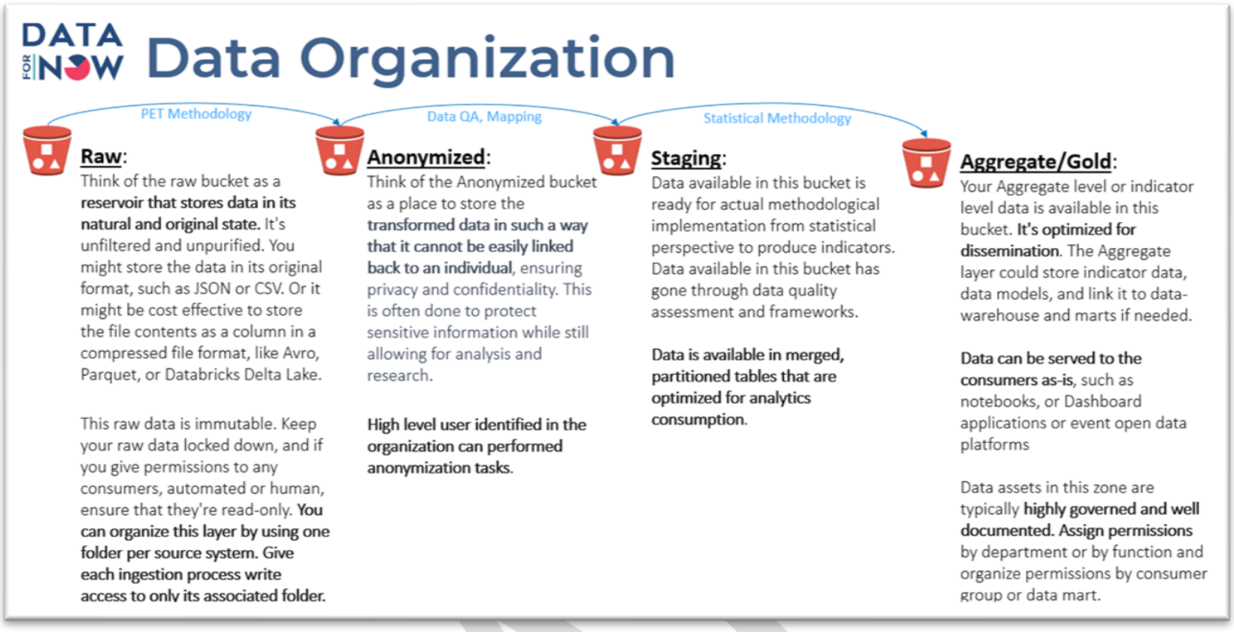
Feature	**Parquet**	**ORC**	**Avro**	**CSV**
Type	Columnar	Columnar	Row-based	Row-based
Development	Apache (originally Cloudera/Twitter)	Apache (originally Hortonworks)	Apache	N/A (standard format)
Compression	Excellent (built-in)	Excellent (built-in)	Good	Poor (requires external compression)
Schema Support	Self-describing	Self-describing	Self-describing	No schema
Schema Evolution	Limited	Limited	Excellent	N/A
Query Performance	Excellent for analytical queries	Excellent for analytical queries	Good for record processing	Poor for large datasets
Write Performance	Moderate	Moderate	Fast	Very fast
Ecosystem Support	Hadoop, Spark, Presto, Athena, Snowflake	Hadoop, Hive, Spark, Presto	Hadoop, Kafka, Spark	Universal
File Size	Small (highly compressed)	Smallest (highly optimized)	Medium	Large (uncompressed)
Random Access	Good	Good	Limited	Poor
Best For	Analytics, data warehousing	Hive/ORC optimized analytics	Data serialization, streaming	Simple data exchange, small datasets
Storage Efficiency	Very high	Very high	High	Low
Processing Overhead	Medium	Medium	Low	Very low

2.3.4 Data organization strategy

Effective data organization in a data lake is essential to ensure data discoverability, accessibility, and usability for various data consumers. Here are some good practices for data organization on a data lake:

1. **Hierarchical structure**: Organize data using a hierarchical folder structure based on categories, domains, or data sources, etc. partition data based on attributes like date, location, or type to improve query performance.
2. **Data catalog (Metadata)**: Implement a data catalog to serves as a central repository for metadata, data definitions, and data lineage.
3. **Data versioning and lifecycle management**: Track changes to datasets and manage retention policies using tools like MinIO or Apache Atlas.

4. **Logical data model**: Define a logical data model to guide data organization and categorization. Consider frameworks like Linked Open Data (LOD) or SDMX for interoperability.



Data Organization

PET Methodology → Data QA, Mapping → Statistical Methodology

Raw:
Think of the raw bucket as a **reservoir that stores data in its natural and original state.** It's unfiltered and unpurified. You might store the data in its original format, such as JSON or CSV. Or it might be cost effective to store the file contents as a column in a compressed file format, like Avro, Parquet, or Databricks Delta Lake.

This raw data is immutable. Keep your raw data locked down, and if you give permissions to any consumers, automated or human, ensure that they're read-only. **You can organize this layer by using one folder per source system. Give each ingestion process write access to only its associated folder.**

Anonymized:
Think of the Anonymized bucket as a place to store the **transformed data in such a way that it cannot be easily linked back to an individual**, ensuring privacy and confidentiality. This is often done to protect sensitive information while still allowing for analysis and research.

High level user identified in the organization can performed anonymization tasks.

Staging:
Data available in this bucket is ready for actual methodological implementation from statistical perspective to produce indicators. Data available in this bucket has gone through data quality assessment and frameworks.

Data is available in merged, partitioned tables that are optimized for analytics consumption.

Aggregate/Gold:
Your Aggregate level or indicator level data is available in this bucket. **It's optimized for dissemination.** The Aggregate layer could store indicator data, data models, and link it to data-warehouse and marts if needed.

Data can be served to the consumers as-is, such as notebooks, or Dashboard applications or event open data platforms

Data assets in this zone are typically **highly governed and well documented. Assign permissions** by department or by function and organize permissions by consumer group or data mart.

Figure 2-3 Sample data buckets to organize data

While implementing in various NSO under Data4Now initiative, below suggested folders/buckets are created enhancing the medallion architecture:

1. **Raw**: Contains unprocessed data from surveys, administrative records, and external sources.
2. **Anonymized**: Contains data anonymization using privacy enhancement technologies (PET).
3. **Staging**: Holds cleaned, validated, and standardized data.
4. **Aggregate/Gold**: Contains finalized statistical reports, indicators, and insights.
5. **Archive**: Stores historical data for trend analysis and research.

2.3.5 Data access management

Effective data access management safeguards sensitive data, prevents unauthorized access, and ensures compliance with data protection regulations. Key practices include:

1. **Policy-Based Access Control (PBAC)**: Assign permissions based on user roles and responsibilities.

2. **Least Privilege Principle**: Grant users the minimum access required for their tasks.

3. **Data Classification**: Classify data by sensitivity and apply appropriate access controls.

These are just few points in data access management. NSO may need to define more custom access management strategy based on their need.

2.4 Data processing and analytics

Data processing involves cleaning, modeling, or linking datasets to prepare them for analysis. This process uses standard or innovative methodologies to produce meaningful statistics and indicators. The results are then stored or published in reports or dashboards. This is where business logic is implemented along with data quality rules. Data processing tools are hence one of the most integral parts of IT architecture from a statistical perspective. It can play a crucial role in facilitating collaboration among different teams within data architecture. Few considerations to make in this layer along with tools are discussed in below sections.

2.4.1 Few considerations:

1. **Flexible data access**: Platforms that can easily integrate with existing architecture and provide directly access data are essential. This allows statisticians, data engineers, and data scientists to work with the same datasets, eliminates data silos and ensures consistency in data usage.
2. **Reproducibility**: the ability to reproduce analyses and experiments easily by rerunning processes without external interference is critical. This is crucial for validation of results and the accuracy of analysis over time.
3. **Interactive environment**: An interactive environment where users can write and execute code, visualize data, and create explanatory narratives all in one place is equally important. Such environment fosters collaboration, enabling teams to share insights, code, and documentation effectively.
4. **Collaboration tool**: Platforms that allow teams to create notebooks containing code snippets, scripts, and data analysis procedures promote transparency and collaboration. Statisticians, data engineers, and data scientists can share their work, enabling team members to understand and contribute to each other's analyses.
5. **Visualization and Documentation**: Integrating code execution with rich visualizations and text enables teams to present their findings comprehensively. Statisticians can explain methodologies, data engineers can document transformations, and data scientists can highlight model results within a single document.
6. **Extension Capabilities**: Support to various programming languages and libraries like Python, R, etc. makes it adaptable to different team members' expertise and needs.

Providing a common platform to perform data modeling and analysis brings various teams together while providing security, consistency and efficiency.

2.4.2 Data processing tools

Several tools are widely used for data processing and analytics, each offering unique capabilities to support statistical workflows:

1. **JupyterHub/JupyterLab**: Enables collaborative, notebook-based exploration and analysis.
2. **Apache Spark**: Handles large-scale data processing with APIs for Python, Java, and Scala, making it suitable for distributed computing.

3. **Dask**: Facilitates scalable computation for Python-based workflows, particularly for parallel processing.
4. **SQL Engines (e.g., Trino):** Provide high-performance querying for both structured and semi-structured datasets
5. **QGIS:** A geographic information system (GIS) tool that allows users to view, edit, analyze, and publish spatial data.

Advantages, disadvantages, and use cases for some of these tools are further detailed in Annex Section 5.7 "Data processing tools".

2.5 Data visualization and dissemination

Data visualization involves representing data in graphical or visual formats, such as charts, graphs, dashboards, or interactive maps. These tools help users explore patterns, trends, and insights effectively. Visualization helps communicate complex data intuitively and can drive decision-making in organizations.

Data Dissemination involves making processed and analyzed data accessible to the intended audience through various means like APIs, data portals, downloadable datasets, or reports. Effective dissemination ensures data is accessible, reusable, and easy to understand for target users, such as analysts, policymakers, or the public.

2.5.1 Few considerations from a data lake Perspective:

- **Integration**: Visualization and dissemination tools should be integrated seamlessly with the data lake to fetch real-time or batch-processed data. Technologies like APIs or connectors facilitate this.
- **Scalability**: Systems should be designed to handle large volumes of data flowing in from the data lake, accommodating concurrent queries or visualizations.
- **Accessibility**: Dashboards and visualizations should be intuitive and user-friendly.
- **Data Governance**: Appropriate access control, anonymization, and security measures are important factors that should be in place when sharing or visualizing data.
- **Formats and Standards**: Disseminated data should comply with open standards (e.g., JSON, CSV, XML, SDMX) to enable interoperability and reuse.
- **User-Centric Design**: Tailor visualization tools and dissemination methods to meet diverse user needs, offering both high-level summaries and granular data.

2.5.2 Data visualization tools

Apache Superset: A modern, open-source data exploration and visualization platform. It provides an intuitive interface for building dashboards and exploring data from various sources, including relational databases and data lakes. It supports a wide range of data connectors and is designed to make data exploration and visualization accessible to users of all skill levels.

Metabase: A user-friendly analytics and dashboard creation platform that simplifies data exploration and visualization. It is designed to be accessible to non-technical users while offering advanced features for data analysts.
A quick comparison of few of these platforms is shown in the table below:

Feature	**Apache Superset**	**Metabase**
Primary Focus	Enterprise-grade data exploration and visualization	User-friendly analytics and dashboarding
License	Open Source (Apache)	Open Source (AGPL) with commercial options
Target Users	Data analysts, scientists, engineers	Business users, analysts (less technical focus)
Learning Curve	Moderate to steep	Low (designed for ease of use)
SQL Knowledge Required	Yes (for advanced features)	Optional (has SQL and no-code options)
Visualization Options	Extensive (100+ chart types)	Good (fewer options but covers essentials)
Data Source Connectors	40+ databases and SQL engines	20+ databases and SQL engines
Dashboarding	Advanced with complex layouts	Simplified but effective
Data Exploration	Excellent (core strength)	Good
Self-service Analytics	Moderate	Excellent (core strength)
Embedding	Supported	Supported (premium feature in paid plans)
Alerting	Basic	More comprehensive
Community Size	Large, active	Large, active
Best Use Case	Complex data analysis, extensive visualization needs	Quick insights, business user adoption
Enterprise Features	Through community plugins	Through paid plans
Governance & Security	Strong	Basic in open source, stronger in paid version

2.6 Security and Authorization

Entitlements and security are among the most important components of modern IT architecture. They serve as cross cutting elements that support the overall **governance** of the infrastructure. Properly implemented, they enable secure access to resources while safeguarding sensitive information, ensuring data privacy, and maintaining compliance with relevant regulations.

Balancing the need to make data available to more users with the requirements of security and compliance can be challenging. However, several approaches can help achieve this balance. Few considerations to make in this layer along with tools are discussed in below sections with focus on access control.

2.6.1 Few considerations:

1. **Authentication**: Implement robust identity management systems (e.g., LDAP, OAuth2) to verify the identity of users and services.
2. **Authorization**: Use role-based access control (RBAC) or policy-based access control (PBAC) to define and enforce permissions effectively.
3. **Encryption**: Ensure data is encrypt both at rest (e.g., AES-256) and in transit (e.g., TLS/SSL).
4. **Auditing and monitoring**: Continuously monitor access logs and employ anomaly detection mechanisms to identify potential security breaches.
5. **Compliance**: Adhere to relevant data protection regulations, such as General Data Protection Regulation (GDPR), or applicable national frameworks.

Functionalities, Advantages, disadvantages, and use cases for security are further detailed in Annex "Security and authorization".

2.7 DevOps and Containerization

DevOps practices integrate software development and IT operations, enabling faster and more reliable updates to systems and applications. Containerization encapsulates applications along with their dependencies, ensuring consistent deployment across diverse environments. Below are some tools and approaches to consider for implementing DevOps and containerization:

2.7.1 Kubernetes cluster

A Kubernetes cluster is a production-grade container orchestration platform that automates the deployment, scaling, and management of containerized applications.

However, using a Kubernetes cluster is not mandatory for deploying data platforms or other infrastructure components. The decision to adopt Kubernetes should be based on your organization's specific requirements, goals, and operational capacity. For a detailed breakdown of the advantages and disadvantages of Kubernetes, tools for deployment and management, and recommendations on when to use it, refer to Annex "Kubernetes cluster".

2.7.2 Alternative Approaches

1 **Virtual Machines or Bare Metal:** For straightforward deployments, traditional virtual machines(VMs) or bare metal servers may be sufficient and easy to manage.

2 **Docker Swarm**: Offers a simpler learning curve and native Docker integration but has limited scalability compared to Kubernetes. It is best suited for smaller deployments.

3 **Managed Container Services:** Consider using managed Kubernetes services (e.g., Amazon EKS, Azure AKS, Google GKE) to reduce operational overhead while leveraging from Kubernetes features.

4 **Platform-as-a-Service (PaaS):** For simpler applications, PaaS offerings like Heroku, AWS Elastic Beanstalk, or Google App Engine can provide easier deployment and management without needing to manage Kubernetes.

Kubernetes provides significant benefits for managing complex, containerized applications but introduces added complexity and costs. The decision to use Kubernetes should be guided by your specific use case, application complexity, and operational capacity. For simpler deployments or organizations new to container orchestration, alternative approaches or managed services may offer a more practical starting point. Conversely, for large-scale, complex applications requiring advanced orchestration and automation, Kubernetes can provide powerful scalable solutions.

When evaluating options, consider the features, resource requirements, and complexity of each approach to determine the best fits for your use case and deployment environment.

3 Basic requirements for minimum data lake technology stack used in Data4Now

3.1 Overview

Based on the platform where the technology stack is deployed, technology stack, volume/variety of the data, new and innovative methodology implementation including machine learning algorithms will influence the hardware and technical skill requirements for the infrastructure. We will briefly discuss some of the requirements to deploy 'Figure 1-1 Minimum data lake technology stack used in Data4Now' in this section.

3.2 Skills requirements

Proposed minimum data lake technology stack may require a diverse set of skills to deploy and manage it. Few of them include:

- **System Administration:** Operating system administration (Linux) to set up and manage server environments along with Package management, user management, security configurations, and system monitoring.
- **Infrastructure Management:** Proficiency in managing and configuring server hardware, networking, and storage devices. Knowledge of virtualization and containerization technologies (e.g., Docker, Kubernetes). Ability to provision and manage virtual machines, containers, and storage resources using Infrastructure as Code approach.
- **Networking and Security:** Knowledge of network architecture, including routing, load balancing, and firewall configuration. Security best practices, including user access control, encryption, and authentication mechanisms.
- **Scripting and Automation:** Proficiency in scripting languages like Python or Bash for automating routine tasks.
- **Version Control System**: Version Control System GIT is a critical tool used to manage changes to source code and other files in the repository.
- **Problem Solving and Troubleshooting:** Strong analytical and problem-solving skills to diagnose issues and implement solutions. Effective troubleshooting and debugging of IT problems.
- **Data Management:** Knowledge of setting up and managing databases or object storage solutions like MinIO. Understanding data redundancy, durability, and backup strategies

The following table summarizes the key tools in the stack and the corresponding skills required to deploy and manage them effectively:

Tool / Component	**Primary Function**	**Required Skills**	**Skill Level**	**Notes**
Apache NiFi	Data ingestion & flow orchestration	Data pipeline design XML/JSON handling Network protocols (FTP, API) NiFi UI & processors	Intermediate	Visual interface reduces coding needs; scripting optional
MinIO	Object storage	Linux system admin S3 API familiarity IAM policy configuration Storage architecture	Intermediate	Similar to AWS S3; integrates with AD/LDAP
JupyterHub	Collaborative data analysis	Python/R scripting Package management (pip, conda) JupyterLab extensions User management	Intermediate to Advanced	Useful for statisticians and data scientists
Trino (PrestoSQL)	Data virtualization & querying	SQL (advanced) Schema design Connector configuration Query optimization	Advanced	Useful for federated queries across MinIO, DBs
Kubernetes	Container orchestration	Cluster setup & management Helm charts kubectl CLI Networking & volumes	Advanced	Optional for small deployments; essential for scaling
Active Directory (AD)	Identity & access management	LDAP/AD integration RBAC/ABAC concepts Security policies	Intermediate	Can be integrated with NiFi, MinIO, JupyterHub
Helm	Kubernetes package manager	Helm CLI YAML templating Chart customization	Intermediate	Used for deploying NiFi, JupyterHub, etc.
Git	Version control	Git CLI Branching & merging Repo management	Basic to Intermediate	Essential for managing deployment scripts
Linux (Ubuntu)	OS for all components	Shell scripting Package management (apt) System monitoring	Intermediate	Base OS for all deployments
Python / R	Data processing	Data wrangling (pandas, dplyr) Visualization Statistical modeling	Intermediate to Advanced	Used within JupyterHub notebooks
Longhorn	Persistent storage for Kubernetes	Kubernetes storage concepts iSCSI/NFS setup Helm deployment	Intermediate	Optional but recommended for resilience

3.3 Platform

There are several factors to consider when deciding where the platform will be hosted. Few of them include budget, security requirements, scalability needs, and technical expertise. Below we try to highlight some of the options.

3.3.1 On-Premises Infrastructure

On-premises solutions are physically located at an organization's site or in a hosting location. The hardware, applications, and all the data are stored on servers or a private cloud where it is protected with a firewall at that location. Although it provides full control over hardware, software, and data, it has high initial investment cost along with recurring maintenance and disaster recovery cost associated with it.

For reasons of security and data sovereignty, many NSOs have opted to host on-premises model, while some have used cloud solution hosted within their territory as required by their law.

3.3.2 Cloud Infrastructure

When talking about cloud infrastructure, we will be focusing on IaaS (Infrastructure as a Service) to implement IT Architecture. Having said that there also exists PaaS (Platform as a Service) and SaaS (Software as a Service) within cloud services and should be considered as needed.

IaaS is renting third-party hardware or virtualized computer resources including storage and networking over the internet. We decide and manage the platform, operating system, technology stack, development tools, and configuration of the system. Some of the popular IaaS providers include Azure, AWS, GCP, Digital-Ocean, Local telecom operator, etc.

This option eliminates large initial investment costs but could incur data transfer costs and dependency on internet.

The choice between on-premises and cloud infrastructure is dependent upon several factors, including financial resources, security considerations, scalability requirements, and the technical capacity available within the organization. Many enterprises opt for a hybrid approach, integrating on-premises and cloud-based solutions to leverage the advantages of both. However, managing a hybrid infrastructure presents certain challenges, including the seamless integration of on-premises and cloud components, ensuring robust security across both environments, and enhancing the capacity of IT personnel to operate effectively within a diverse technological landscape.

3.4 Operating system

We have used Ubuntu 24.04 OS as a base to configure the minimum data lake technology stack used in Data4Now setting.

3.5 Hardware requirements

The hardware requirements for the infrastructure are dependent on the size and complexity of the data, as well as the methodological requirements for generating various statistical outputs. These may range from basic data processing to geospatial analysis and machine learning approaches, while also considering the necessity for concurrent and parallel processing. It is essential to account for the scalability and growth potential of the data platform when defining the hardware infrastructure.

It should be noted that all recommendations regarding the number of processing cores refer to **logical cores** rather than physical cores. Logical cores represent the number of CPU threads recognized and utilized by the operating system.

Furthermore, hardware recommendations have been generalized to provide flexible guidance across different deployment scenarios. A simplified reference table is provided below, which presents indicative hardware resource allocations based on user concurrency per component. It is important to emphasize that these recommendations serve as general guidelines and should be tailored to the specific use case, considering performance evaluations and operational requirements.

Hardware needs vary by user concurrency and data volume. Below are indicative specifications:

Concurrent Users	CPU Cores (Per Component as applicable)	RAM (Per Component as applicable)	Disk (Per Component as applicable)	Network Bandwidth
1	Minimum: 2 cores Recommended: 4 cores	Minimum: 4 GB Recommended: 8 GB	Minimum: 100 GB Recommended: 200 GB	Gigabit Ethernet
5	Minimum: 4 cores Recommended: 8 cores	Minimum: 8 GB Recommended: 16 GB	Minimum: 150 GB Recommended: 400 GB	Gigabit Ethernet
10	Minimum: 8 cores Recommended: 8+ cores	Minimum: 16 GB Recommended: 32 GB	Minimum: 200 GB Recommended: 800 GB	Gigabit Ethernet
25	Minimum: 8+ cores Recommended: 16 cores	Minimum: 32 GB Recommended: 64 GB	Minimum: 300 GB Recommended: 1.5 TB	10 Gigabit Ethernet

Concurrent Users	CPU Cores (Per Component as applicable)	RAM (Per Component as applicable)	Disk (Per Component as applicable)	Network Bandwidth
50	Minimum: 16 cores Recommended: 32 cores	Minimum: 64 GB Recommended: 128 GB	Minimum: 500 GB Recommended: 2.5 TB	10 Gigabit Ethernet
100	Minimum: 32 cores Recommended: 64 cores	Minimum: 128 GB Recommended: 256 GB	Minimum: 1 TB Recommended: 4 TB	10 Gigabit Ethernet

3.5.1 Apache NiFi

A minimum cluster configuration requires at least two nodes. (Additional reference: Apache NiFi System Requirement)

Example minimum node

Node	CPU	Memory	Storage
Coordinator/Primary/Zookeeper	\>2 cores	\> 4 GB	\> 10 GB

Additional considerations:

Memory: Tool's performance benefits from having enough memory to manage data flows efficiently. More memory allows for better caching and reduces the need to read from disk frequently.

Disk Type: Using Solid State Drives (SSDs) can significantly improve performance due to faster read/write speeds compared to traditional Hard Disk Drives (HDDs).

Number of Nodes: If you are planning to deploy in a clustered setup for high availability and load balancing, the hardware requirements for each node in the cluster should meet or exceed the recommended specifications.

Network: A fast and reliable network connection is essential, especially when dealing with data flows that involve multiple sources and destinations.

Java Version: Many of these tools require Java to run. Make sure to use a compatible Java version based on the tools version you are using.

Monitoring and Optimization: As your data flows and usage patterns evolve, consider monitoring your tool instance's resource utilization and performance. You might need to fine-tune settings and resources based on the specific requirements of your data flows.

3.5.2 MinIO

MinIO is a software-defined high performance distributed object storage server. You can run MinIO on consumer or enterprise-grade hardware and a variety of operating systems and architectures.

A cluster should ideally have a minimum of 4 nodes, to meet redundancy requirements, and a maximum of 32 nodes. A 32-node cluster can store an average of 200 petabytes of data, or 6,250 terabytes per node.

For performance reasons, each node uses on average 1 to 4 GB RAM, 500 milli-core CPU and 400 MHz CPU. The Disk size depends on storage requirements. For big data applications, the main unit for measuring the capacity of a data node is the terabyte. So, we can consider a minimum storage capacity of at least 512 GB.

Node	CPU	Memory	Storage
4	\>2 cores	4 to 16 GB	\> 100 GB
8	\>4 cores	8 to 32 GB	\> 8TB
16	\>8 cores	16 to 64 GB	\> 16 TB
32	\>16 cores	64 to 128 GB	\> 32 TB

Addition references: https://min.io/product/reference-hardware

3.5.3 JupyterHub

To determine the size requirements of JupyterHub, consider these two factors:

1. number of active Notebook sessions that will run concurrently.
2. complexity of the operations being performed in Notebook

Recommended Memory = (Maximum Concurrent Users * Maximum Memory per User) + 128 MB
Recommended vCPUs = (Maximum Concurrent Users * Maximum CPU Usage per User) + 20%
Recommended Disk Size = (Total Users x Maximum Disk Usage per User) + 2 GB

Number of concurrent users	CPU	Memory	Storage
10	6 cores	6 GB	12 GB

Number of concurrent users	CPU	Memory	Storage
100	51 cores	60 GB	102 GB
500	501 cores	300 GB	5 TB

Additional references: https://data.berkeley.edu/choosing-right-jupyterhub-infrastructure

3.6 Sample use-case to estimate Hardware requirement

As an example, we have below assumption that will help estimate hardware requirement to establish a minimum viable data lake platform.

Assumptions:

- **Data growth**: 1 GB of new data monthly in compressed Parquet format.
- **Data flow**: Handles hundreds of MB of data monthly.
- **User Load**: 10 concurrent JupyterHub users working with hundreds of MB of data.

Based on the assumptions outlined above, the hardware requirements can be estimated as below:

- **Storage:**
    - OS + Kubernetes: 50GB per node
    - MinIO: Start with 500GB, expandable based on growth
    - Additional space for logs and temporary files: 100GB
    - **Total: At least 650GB storage, recommend 1TB for growth**
- **CPU**
    - Base Kubernetes system: 2 cores
    - NiFi: 2 cores (light processing load)
    - MinIO: 2 cores
    - JupyterHub: 4 cores (shared among 10 users)
    - **Total: 10 CPU cores (minimum)**
- **Memory:**
    - Base Kubernetes system: 4GB
    - NiFi: 4GB
    - MinIO: 4GB
    - JupyterHub: 20GB (approximately 2GB per user)
    - **Total: 32GB RAM (minimum)**

4 Cases Studies

To Be added in next release, but here are few NSO stories for reference.

1. Colombia
2. Senegal
3. Sierra Leone
4. Vietnam
5. Tunisia
6. Maldives
7. Namibia

5 Technical implementation guide for minimum data lake technology stack used in Data4Now

5.1 Overview

Deploying any platform involves necessary planning of the infrastructure, technologies, team, budget, and processes. It is a multifaceted process that involves careful planning, robust infrastructure, and adherence to good practices. In the previous section, we briefly touched upon some of the basic requirements for minimum data lake technology stack. This section will try to navigate the workings of the platform deployment, offering insights into key concepts, architectural considerations, deployment strategies, and maintenance practices. However, before jumping into the actual technical implementation, it is important to plan the implementation strategies and resource needs. A phased implementation approach could be a good way forward that allows NSOs to gradually adopt components based on their priorities and capacity.

For the documentation of the deployment process of "Figure 1-1 Minimum data lake technology stack used in Data4Now", we will use Kubernetes cluster (k8s) as a scalable platform. However, it can also be deployed in docker or standalone server as per the need of NSO. The selected open data stack includes MinIO as storage tool, JupyterHub with Python and R notebooks enabled as analytics tool, Apache NiFi as data ingestion tool, along with existing Active Directory integrated into each of the tools as authentication and authorization tool. This configuration represents a minimum setup, and additional tools or components can be integrated as per the need of the NSO.

There are two sections needed to deploy and manage the Kubernetes cluster. First is the Administrators workstation that is used to manage the cluster. Second is the actual Datacenter or servers where the Kubernetes cluster will be installed. Below diagram highlights these two components.

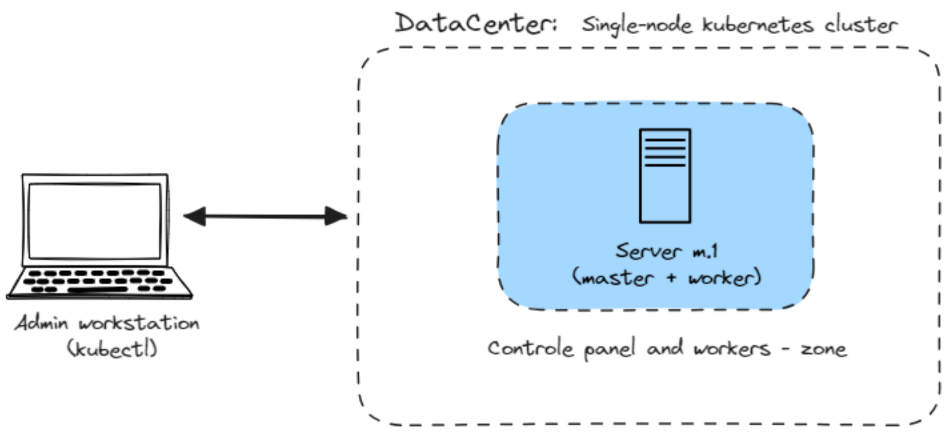

DataCenter: Single-node kubernetes cluster
Server m.1
(master + worker)
Controle panel and workers - zone
Admin workstation
(kubectl)

FIGURE 5-1 DEPLOYMENT OF DATA PLATFORM - SINGLE NODE CLUSTER

The Kubernetes cluster can be further configured to be multi-node cluster to achieve high availability, fault tolerance, and scalability of the applications deployed. Thus far, NSO of Tunisia (INS) is the only agency where we have used multi-node cluster deployment. Meanwhile in other NSOs like Viet Nam (GSO), Namibia (NSA), SierraLeone (Stats SL), have used single-node cluster deployment.

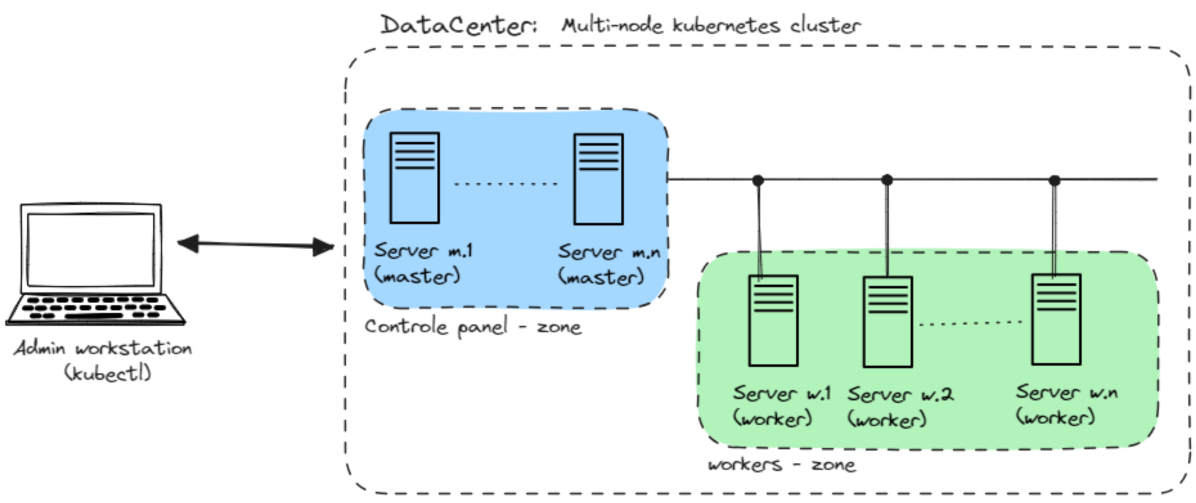


FIGURE 5-2 DEPLOYMENT OF DATA PLATFORM - MULTI-NODE CLUSTER

The selection of the appropriate cluster architecture, whether single-node or multi-node, is primarily contingent upon the available infrastructure capacity and the level of high availability required for seamless access to the data lake. A thorough assessment of these factors is essential to ensure sustainability, resilience, and operational efficiency of the deployed system.

To facilitate effective administration and management of the infrastructure, it is imperative that both the administrator's workstation and the designated server(s) be configured with the necessary tools and software components. These include, but are not limited to :

- The installation of kubectl, Helm, Git, Lens, and the MinIO Client (mc) on the administrator's workstation to enable administrative tasks and system monitoring.
- The deployment of a Kubernetes cluster configured with persistent storage to ensure data durability and system reliability.

The diagram below provides a comprehensive overview of the deployment of infrastructure, illustrating the key components and their interconnections within the system. The base operating system for the server/datacenter is Ubuntu.

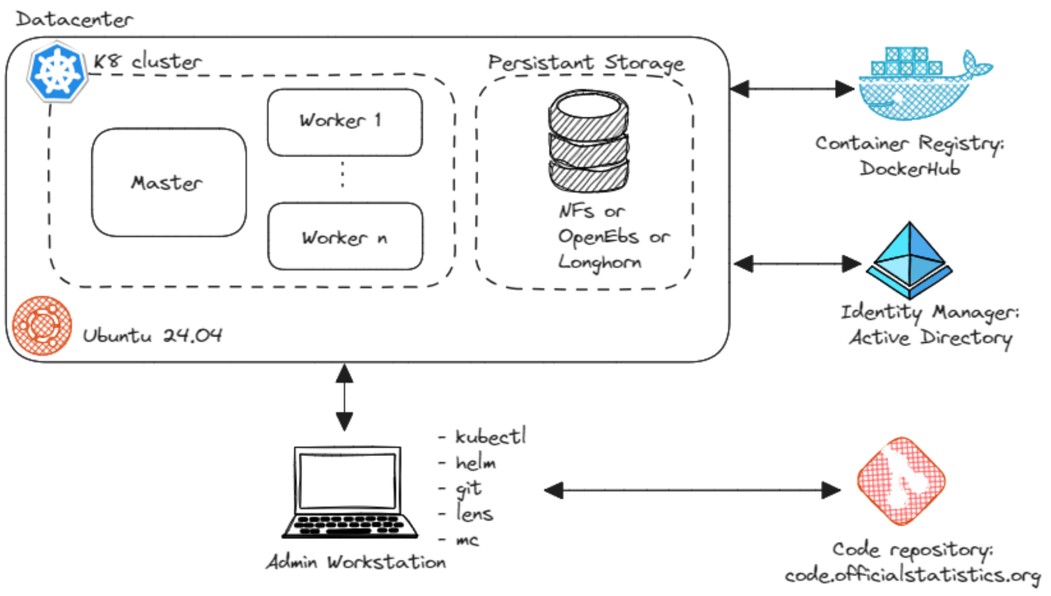

Datacenter
K8 cluster
Master
Worker 1
Worker n
Persistant Storage
NFs or OpenEbs or Longhorn
Ubuntu 24.04
Container Registry: DockerHub
Identity Manager: Active Directory
- kubectl
- helm
- git
- lens
- mc
Admin Workstation
Code repository: code.officialstatistics.org

FIGURE 5-3 KUBERNETES INFRASTRUCTURE OVERVIEW

All deployed tools are open source, with images sourced from DockerHub. A connection with Active Directory ensures a unified authentication mechanism for all users. The platform's code is available in the repository at code.officialstatistics.org, with a branch repository created for each country.

In addition to the Kubernetes cluster, an equally crucial tool is deployed on the servers: the Persistent Storage System. This system facilitates the management of application data within the Kubernetes cluster by providing mechanisms for backup and restoration of the data.

5.2 Checklist

To facilitate the deployment of the platform, a structured set of steps has been established. An initial checklist, consisting of three (3) steps, was developed during deployments in NSOs of Vietnam (GSO) and Namibia (NSA).

A more refined version of the checklist was developed following a mission in NSO of Tunisia (INS). This version was updated based on insights gained from the deployment in NSO of Sierra Leone (StatsSL). As a result, the checklist evolved from three (3) steps to nine (9) steps, incorporating multiple sub-steps to enhance clarity and efficiency.

The complete list of steps and sub-steps included in the checklist is available as an Excel file, as presented in Annexes 'Minimum data lake technology stack for Data4Now deployment checklist'.

5.3 Deployment

The deployment of the Kubernetes cluster shall be conducted sequentially following the steps outlined in the checklist.

5.3.1 Prerequisites

This section outlines the essential prerequisites for deploying the minimum data lake infrastructure, ensuring that all necessary configurations are in place. The servers are expected to be set up with Ubuntu 24.04 or higher Operating System.

5.3.1.1 Resource Allocation and Component Distribution

Prior to the deployment of the platform, it is important to conduct a comprehensive assessment of resource allocation and the distribution of components across various servers. This assessment aims not only to determine the necessary server capacity but also to define the parameters for configuring resources allocated to each component of the platform. For further information, please refer to 'Hardware requirements'

5.3.1.2 Collect Essential Information

The following preparatory information should be readily available:

1. **Master Node Identification:** The Internet Protocol (IP) address of the master node, hereinafter referred to as {master_ip_address}.
2. **Worker Node Addressing:** The IP address of each worker node, required for deployment documentation, and hereinafter referenced as {worker_ip_address}.
3. **Node Naming Convention:** Each node should be uniquely identified as "master," "node1," "node2," etc., referenced in the deployment command line as {node_name}.
4. **Code Repository Access:** The hyperlink to the official code repository, referenced as {code_repository_link}.

5.3.1.3 Workstation Setup

It is assumed that the workstation operates on the Windows operating system.

5.3.1.3.1 Install the Essential Tools: Git, Lens, Docker-Desktop

You can deploy the tools (Git, Lens, Docker-desktop) directly from their respective websites. To facilitate deployment, we are using chocolate package manager to install the required tools using the code below:

1. Open PowerShell as an administrator and execute the following commands:

```
## Install Windows Chocolatey Package Manager
C:\>Set-ExecutionPolicy Bypass -Scope Process -Force; `
[System.Net.ServicePointManager]::SecurityProtocol = `
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/in
stall.ps1'))
```

2. Reopen PowerShell as the current user and install the required tools:

```
## Install Git
C:\>choco install git -y
## Install Lens
C:\>choco install lens -y
## Install Docker-Desktop (Optional)
C:\>choco install docker-desktop -y
```

5.3.1.3.2 Establishment of Working Directories

The directory structure of the workspace shall conform to the following scheme:

```
$HOME\data4now\                   # Main project workspace folder
         |__cli/                    # CLI subfolder to store downloaded binaries
         |   |__helm.exe              # Helm tool
         |   |__mc.exe                # MinIO client tool
         |   |__kubectl.exe           # Kubernetes client tool
         |
         |__code/                   # Subfolder for code content
```

Execute the following commands to create the required directories:

```
## Create main workspace directory
C:\>mkdir "$HOME\data4now"
## Create subdirectory for binary files
C:\>mkdir "$HOME\data4now\cli"
## Create subdirectory for code content
C:\>mkdir "$HOME\data4now\code"
```

5.3.1.3.3 Configuration of Environment Variables

Execute the following PowerShell commands to configure the necessary environment variables:

```
## Define workspace path
C:\>[System.Environment]::SetEnvironmentVariable('D4N_WORKSPACE', "$HOME\data4no
w", [System.EnvironmentVariableTarget]::User)
## Define CLI workspace path
C:\>[System.Environment]::SetEnvironmentVariable('D4N_WORKSPACE_CLI', "$HOME\dat
a4now\cli", [System.EnvironmentVariableTarget]::User)
## Define code workspace path
C:\>[System.Environment]::SetEnvironmentVariable('D4N_WORKSPACE_CODE', "$HOME\da
ta4now\code", [System.EnvironmentVariableTarget]::User)
## Assign master IP address
C:\>[System.Environment]::SetEnvironmentVariable('D4N_MASTER_IP_ADDRESS', "{mast
er_ip_address}", [System.EnvironmentVariableTarget]::User)
## Assign repository link
C:\>[System.Environment]::SetEnvironmentVariable('D4N_REPOSITORY', "{code_reposi
tory_link}", [System.EnvironmentVariableTarget]::User)
## Update system PATH with CLI directory
C:\>$currentPath = [System.Environment]::GetEnvironmentVariable('Path', [System.
EnvironmentVariableTarget]::User)
C:\>$newPath = "$currentPath;$Env:D4N_WORKSPACE_CLI"
C:\>[System.Environment]::SetEnvironmentVariable('Path', $newPath, [System.Envir
onmentVariableTarget]::User)
```

5.3.1.3.4 Downloading Essential Binaries and Cloning the Repository

```
## Download Helm
C:\>Invoke-WebRequest -Uri "https://get.helm.sh/helm-v3.17.1-windows-amd64.zip"
-OutFile "$Env:D4N_WORKSPACE_CLI\helm.zip"
C:\>Add-Type -AssemblyName 'System.IO.Compression.FileSystem'
C:\>[System.IO.Compression.ZipFile]::ExtractToDirectory("$Env:D4N_WORKSPACE_CLI\
helm.zip", $Env:D4N_WORKSPACE_CLI)
C:\>Move-Item -Path "$Env:D4N_WORKSPACE_CLI\windows-amd64\helm.exe" -Destination
 $Env:D4N_WORKSPACE_CLI
C:\>Remove-Item -Path "$Env:D4N_WORKSPACE_CLI\helm.zip"
C:\>Remove-Item -Path "$Env:D4N_WORKSPACE_CLI\windows-amd64" -Recurse

## Download Kubernetes client (kubectl)
C:\>$version = (Invoke-RestMethod -Uri https://dl.k8s.io/release/stable.txt).Tri
m()
C:\>$url = "https://dl.k8s.io/$version/bin/windows/amd64/kubectl.exe"
C:\>Invoke-WebRequest -Uri $url -OutFile "$Env:D4N_WORKSPACE_CLI\kubectl.exe"

## Download MinIO client (mc)
C:\>Invoke-WebRequest -Uri "https://dl.min.io/client/mc/release/windows-amd64/m
c.exe" -OutFile "$Env:D4N_WORKSPACE_CLI\mc.exe"

## Clone the repository
C:\>git clone "$Env:D4N_REPOSITORY.git" $Env:D4N_WORKSPACE_CODE
```

5.3.1.4 Configuration on Master Node

```
## Update package lists
$sudo apt-get update -y
## Install SSH service
$sudo apt-get install -y openssh-server
## Enable root SSH login
$sudo echo "PermitRootLogin yes" | sudo tee -a /etc/ssh/sshd_config > /dev/null
## Restart SSH service
$sudo systemctl restart ssh
## Define repository environment variable
$echo 'D4N_REPOSITORY="{code_repository_link}"' | sudo tee -a /etc/environment
## Reboot the server
$sudo reboot
```

5.3.1.5 Configuration on Worker Node

```
## Update package lists
$sudo apt-get update -y
## Install SSH service
$sudo apt-get install -y openssh-server
## Enable root SSH login
$sudo echo "PermitRootLogin yes" | sudo tee -a /etc/ssh/sshd_config > /dev/null
## Restart SSH service
$sudo systemctl restart ssh
```

5.3.2 Server Preparation (All Nodes)

This section outlines the necessary steps to prepare all nodes in the infrastructure for deployment. The commands below ensure system updates, disable swap memory, and configure kernel parameters essential for Kubernetes operations.

```
## Update package lists to ensure the latest versions are available
$sudo apt-get update -y
## Disable swap to ensure Kubernetes functions properly
$sudo swapoff -a
$sudo sed -i '/swap/d' /etc/fstab
## Load required kernel modules for Kubernetes networking
$sudo cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
## Activate the required modules immediately
$sudo modprobe overlay
$sudo modprobe br_netfilter
## Configure system networking parameters for Kubernetes
$sudo cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1  # Allow bridge traffic to be processed
by iptables
net.bridge.bridge-nf-call-ip6tables = 1  # Ensure IPv6 bridge traffic is also pr
ocessed
net.ipv4.ip_forward                 = 1  # Enable IP forwarding for routing
EOF
## Apply the new system configurations
$sudo sysctl --system
```

5.3.3 Container Runtime Installation

This section details the installation and configuration of the container runtime, which is a fundamental requirement for Kubernetes nodes. The following steps ensure a reliable and efficient installation of Docker and cri-dockerd.

```
## Install prerequisite packages for Docker
$sudo apt install -y curl gnupg2 software-properties-common
## Add the official Docker GPG key to the system for package verification
$sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearm
or -o /usr/share/keyrings/docker-archive-keyring.gpg
## Add the Docker repository to the system's package sources
$sudo echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings
/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_rele
ase -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
## Update package lists to include Docker's repository
$sudo apt update -y
## Install Docker and its necessary components
$sudo apt install -y docker-ce docker-ce-cli containerd.io
## Enable Docker to start at system boot
$sudo systemctl enable docker
## Start the Docker service
$sudo systemctl start docker
## Install cri-dockerd (Container Runtime Interface for Docker)
$VERSION=0.3.4
## Download the cri-dockerd archive
$sudo wget https://github.com/Mirantis/cri-dockerd/releases/download/v${VERSION}
/cri-dockerd-${VERSION}.amd64.tgz
## Extract the contents of the downloaded archive
$sudo tar xvf cri-dockerd-${VERSION}.amd64.tgz
## Move the cri-dockerd binary to the appropriate system directory
$sudo mv cri-dockerd/cri-dockerd /usr/local/bin/
## Verify the installation by checking the version
$sudo cri-dockerd --version
## Download the necessary systemd service files for cri-dockerd
$sudo wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packagi
ng/systemd/cri-docker.service
$sudo wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packagi
ng/systemd/cri-docker.socket
## Move the service files to the systemd directory
$sudo mv cri-docker.socket cri-docker.service /etc/systemd/system/
## Modify the service file to reference the correct binary location
$sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/system
d/system/cri-docker.service
## Reload systemd to recognize the new services
$sudo systemctl daemon-reload
## Enable cri-dockerd to start at system boot
$sudo systemctl enable cri-docker
## Start the cri-dockerd service
$sudo systemctl start cri-docker
## Check the status of the cri-dockerd service to ensure it is running
$sudo systemctl status cri-docker
```

5.3.4 Secure Shell (SSH) Access Configuration

This section outlines the necessary steps to establish secure and password-less SSH access between the master node and worker nodes. This setup facilitates seamless remote management and communication between the nodes in a Kubernetes cluster.

5.3.4.1 Generating an SSH Key on the Master Node

```
## Generate an SSH key pair with RSA encryption (4096-bit) without a passphrase
$sudo ssh-keygen -t rsa -b 4096 -N "" -f ~/.ssh/id_rsa
## Append the generated public key to the list of authorized keys
$sudo cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
## Restart the SSH service to apply changes
$sudo systemctl restart ssh
```

5.3.4.2 Distributing the SSH Key to Worker Nodes

```
## Copy the SSH public key from the master node to each worker node
$ssh-copy-id -i ~/.ssh/id_rsa.pub root@<node_ip_address>
## Restart the SSH service on the worker node to apply the changes
$ssh root@<node_ip_address> 'sudo systemctl restart ssh'
```

5.3.5 Installation of Kubernetes Cluster

This section provides a structured approach to setting up a Kubernetes cluster, including installing necessary dependencies, configuring the cluster, deploying it, and setting up a metrics server for monitoring resource utilization.

5.3.5.1 Installation of Required Packages

```
## Install essential dependencies for Kubernetes cluster management
$sudo apt-get install -y socat conntrack
## Download KubeKey, a lightweight tool for Kubernetes installation and cluster
management
$sudo curl -sfL https://get-kk.kubesphere.io | sh -
## Move the KubeKey binary to a system-wide directory for easy execution and
remove temporary installation files
$sudo mv kk /usr/local/bin && sudo rm kubekey*
```

5.3.5.2 Creating and Editing the Cluster Configuration

```
## Generate a configuration file for the Kubernetes cluster
$sudo kk create config -f kubernetes-config.yaml
## Open the configuration file for editing and specify cluster details
# Sample configuration:
#  hosts:
#  - {name: {node_name}, address: {worker_ip_address}, internalAddress:
{worker_ip_address}, user: root, privateKeyPath: "~/.ssh/id_rsa"}
#  ...
#  kubernetes:
#   # List of supported versions can be found at:
#   # https://github.com/kubesphere/kubekey/blob/master/docs/kubernetes-
versions.md
#    version: v1.25.3
#    containerManager: docker
#  ...
#  network:
#    plugin: flannel
$sudo nano kubernetes-config.yaml
```

5.3.5.3 Deploying the Kubernetes Cluster

```
## Deploy the cluster using the specified configuration file
$sudo kk create cluster -f kubernetes-config.yaml
```

5.3.5.4 Deploying the Metrics Component

```
## Deploy the Metrics Server to monitor cluster resource usage
$sudo kubectl apply -f $D4N_REPOSITORY/blob/main/docs/configs/metrcics-server-
components.yaml
## Verify that the Metrics Server deployment is active
$sudo kubectl get deployment metrics-server -n kube-system
## Ensure that the Metrics Server pod is running
$sudo kubectl get pods --namespace kube-system
## Display node-level resource usage statistics
$sudo kubectl top nodes
```

5.3.6 Installation of Persistent Storage

This section outlines the necessary steps to install and configure persistent storage using Longhorn. These steps ensure that each node is equipped with the required dependencies, and that Longhorn is correctly deployed and verified within the Kubernetes cluster.

5.3.6.1 Installation of Required Packages for Longhorn

```
## Update package lists to ensure availability of the latest versions
$sudo apt update
## Install Open-iSCSI and NFS support, which are essential for Longhorn storage
operations
$sudo apt install -y open-iscsi nfs-common
## Enable and start the iSCSI daemon to ensure automatic startup on boot
$sudo systemctl enable iscsid
$sudo systemctl start iscsid
## Load the iSCSI kernel module to support iSCSI storage connections
$sudo modprobe iscsi_tcp
## Persistently enable the iSCSI module to ensure it loads at boot time
$echo "iscsi_tcp" | sudo tee /etc/modules-load.d/iscsi_tcp.conf
```

5.3.6.2 Installation of Longhorn from the Master Node

```
## Add the official Longhorn Helm repository to the Helm package manager
$sudo helm repo add longhorn https://charts.longhorn.io
## Update the Helm repository to fetch the latest available versions
$sudo helm repo update
## Create a dedicated namespace for Longhorn within the Kubernetes cluster
$sudo kubectl create namespace longhorn-system
## Deploy Longhorn using Helm within the dedicated namespace
$sudo helm install longhorn longhorn/longhorn --namespace longhorn-system
## Verify the installation by listing all pods in the Longhorn namespace
$sudo kubectl get pods -n longhorn-system
## Apply a Kubernetes configuration to expose the Longhorn UI via a NodePort
service
$sudo kubectl apply -f $D4N_REPOSITORY/blob/main/docs/configs/longhorn-
proxy.nodeport.yaml
## The Longhorn UI can now be accessed via HTTP at:
http://<server_ip_address>:30080
```

5.3.6.3 Verification of Longhorn Deployment

```
## Ensure that the Longhorn pod manager is running on each node
$sudo kubectl get pods -n longhorn-system -o wide | grep {node_name}
## Verify the integration of nodes with Longhorn storage
$sudo kubectl get nodes.longhorn.io -n longhorn-system
```

5.3.7 Configuring Kubectl on the Workstation

This section provides guidance on configuring kubectl on the workstation, ensuring connectivity with the Kubernetes cluster. The process varies based on whether kubectl is newly installed or if an existing configuration is present.

5.3.7.1 Configuration for a Newly Installed Kubectl

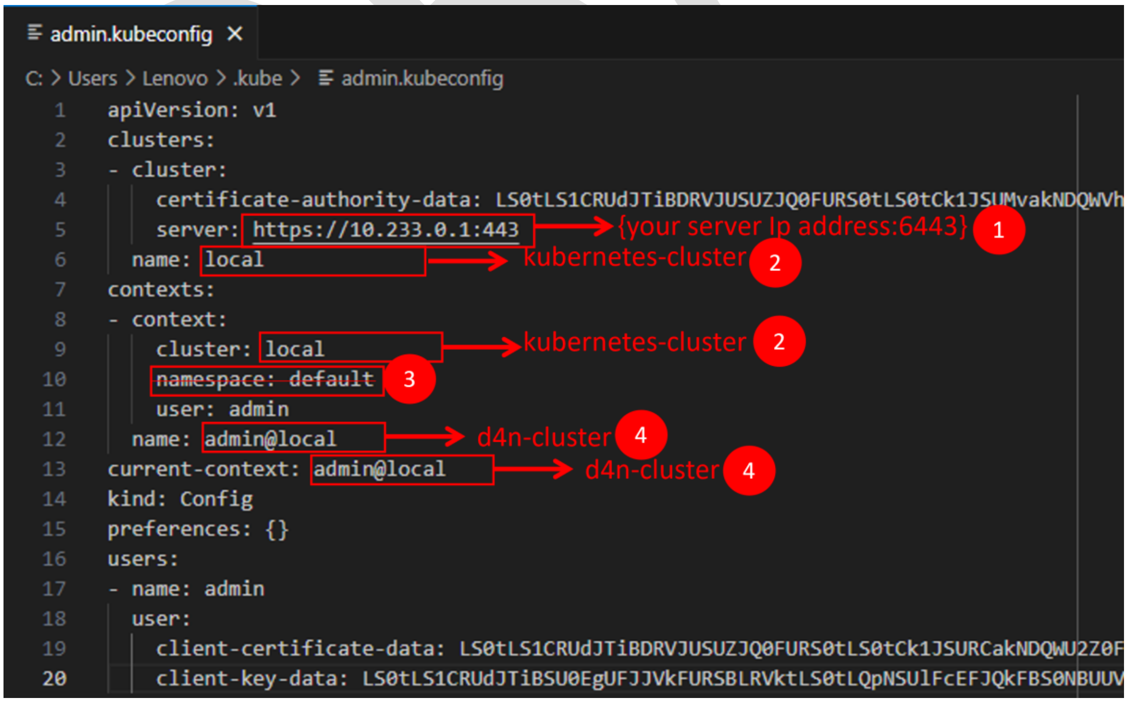
5.3.7.1.1 Download the kubernetes config file from the cluster

```
## Create the .kube directory to store configuration files
C:\>mkdir $HOME\.kube
## Copy the kubeconfig file from the master node to the local workstation
C:\>scp root@$Env:D4N_MASTER_IP_ADDRESS:/etc/kubernetes/admin.conf
$HOME\.kube\config
## Verify the cluster connection by checking available contexts
C:\>kubectl config get-contexts
## Verify that the cluster nodes are accessible
C:\>kubectl get nodes
```

5.3.7.1.2 Edit/Modifying the Kubeconfig File

The kubeconfig file should be updated as shown below:

- ❑ 1: address to the server
- ❑ 2: Cluster name
- ❑ 3: default namespace
- ❑ 4: context name



```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUMvakNDQWVh
    server: https://10.233.0.1:443
  name: local
contexts:
- context:
    cluster: local
    namespace: default
    user: admin
  name: admin@local
current-context: admin@local
kind: Config
preferences: {}
users:
- name: admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCakNDQWU2Z0F
    client-key-data: LS0tLS1CRUdJTiBSU0EgUFJJVkFURSBLRVktLS0tLQpNSUlFcEFJQkFBS0NBUUV
```

FIGURE 5-4 EDIT THE KUBECONFIG FILE

5.3.7.1.3 Verify the access to the cluster

```
## Verify the Cluster Connection Again
C:\>kubectl config get-contexts
C:\>kubectl get nodes
```

5.3.7.2 *Configuration When an Existing Cluster is Already Set Up*

If the workstation already has an existing Kubernetes cluster configuration, follow these steps to merge the new kubeconfig file.

5.3.7.2.1 Download the kubernetes config file from the cluster

```
## Copy the new kubeconfig file from the master node to the local workstation
C:\>scp root@$Env:D4N_MASTER_IP_ADDRESS:/etc/kubernetes/admin.conf
$HOME\.kube\new-config
```

5.3.7.2.2 Modifying the new-config File

The new-config file should be edited as illustrated on "Figure 5-4 Edit the kubeconfig file"

5.3.7.2.3 Merge both the new and current config files

```
## Temporarily set the KUBECONFIG environment variable to include both
configurations
C:\>$Env:KUBECONFIG="$HOME\.kube\config;C:\$HOME\.kube\new-config"
## Backup the existing configuration file before making changes
C:\>cp $HOME\.kube\config $HOME\.kube\config.bak
## Merge the new kubeconfig file with the existing one
C:\>kubectl config view --flatten > $HOME\.kube\config.merged
## Replace the default kubeconfig file with the merged configuration
C:\>mv -Force $HOME\.kube\config.merged  $HOME\.kube\config
## Remove the temporary new-config file
C:\>Remove-Item -Path $HOME\.kube\new-config
## Reset the KUBECONFIG environment variable to point to the final configuration
C:\>$Env:KUBECONFIG = "$HOME\.kube\config"
```

5.3.7.2.4 Verify the access to the cluster

```
## Verify the Cluster Connection After Merging
C:\>kubectl config get-contexts
## Set the default context to the appropriate Kubernetes cluster
C:\>kubectl config use-context <context-name>
## Verify that the nodes are accessible
C:\>kubectl get nodes
```

5.3.8 Clean-Up Procedures

This section provides guidance on securing the server environment by disabling root SSH login on all nodes, including both master and worker nodes. Restricting root access enhances security and mitigates unauthorized access risks.

5.3.8.1 Disabling Root SSH Login

```
## Remove the line that explicitly enables root login from the SSH configuration
file
$sudo sed -i '/^PermitRootLogin yes$/d' /etc/ssh/sshd_config
## Restart the SSH service to apply the changes
$sudo systemctl restart ssh
```

5.3.9 Deployment of the Minimum Data Lake Stack

This section outlines the step-by-step procedure for deploying the essential components of a data lake, including MinIO for storage, JupyterHub for analytics, and Apache NiFi for data ingestion. The deployment is performed from the workstation onto a Kubernetes cluster.

5.3.9.1 Deploy MinIO (Storage)

```
## Navigate to the MinIO deployment directory
C:\>cd $Env:D4N_WORKSPACE_CODE\d4n-minio
## Create a dedicated namespace for storage components
C:\>kubectl create namespace d4n-storage
## Copy the local configuration file and edit it if necessary
C:\>cp values.local.yaml config.yaml
## Install or upgrade MinIO using Helm, ensuring proper configuration
C:\>helm upgrade --cleanup-on-fail --install minio . --namespace d4n-storage --
values config.yaml
```

🛎️ **If deploying on a local Docker Desktop Kubernetes cluster**

```
## Apply the local load balancer configuration to expose MinIO at
http://localhost:9001
## The internal service endpoint for JupyterHub and Apache NiFi remains:
## http://minio.d4n-storage.svc.cluster.local:9000
C:\>kubectl apply -f local-loadbalancer.yaml
```

🛎️ **If deploying on a remote server**

```
## Apply the NodePort proxy configuration to expose MinIO at
http://{master_ip_address}:309
## The internal service endpoint for JupyterHub and Apache NiFi remains:
## http://minio.d4n-storage.svc.cluster.local:9000
C:\>kubectl apply -f proxy.nodeport.yaml --namespace d4n-storage
```

5.3.9.2 Deploy JupyterHub (Analytics)

```
## Navigate to the JupyterHub deployment directory
C:\>cd $Env:D4N_WORKSPACE_CODE\d4n-jupyterhub
## Add the official JupyterHub Helm repository and update Helm repositories
C:\>helm repo add jupyterhub https://hub.jupyter.org/helm-chart/
C:\>helm repo update
## Create a dedicated namespace for analytics components
C:\>kubectl create namespace d4n-analytics
## Copy the local configuration file and edit it if necessary
C:\>cp values.local.yaml config.yaml
## Install or upgrade JupyterHub using Helm with the specified configuration
C:\>helm upgrade --cleanup-on-fail --install jupyter jupyterhub/jupyterhub --
namespace d4n-analytics --version=3.0.3 --values config.yaml
```

🛎️ **If deploying on a local Docker Desktop Kubernetes cluster**

```
## Apply the local load balancer configuration to expose JupyterHub at
http://localhost:8080
C:\>kubectl apply -f local-loadbalancer.yaml
```

🛎️ **If deploying on a remote server**

```
## Apply the NodePort proxy configuration to expose JupyterHub at
http://{master_ip_address}:30808
C:\>kubectl apply -f proxy.nodeport.yaml --namespace d4n-analytics
```

5.3.9.3 *Deploy Apache NiFi (Ingestion)*

```
## Navigate to the Apache NiFi deployment directory
C:\>cd $Env:D4N_WORKSPACE_CODE\d4n-apache-nifi
## Add the official Apache NiFi Helm repository and update Helm repositories
C:\>helm repo add cetic https://cetic.github.io/helm-charts
C:\>helm repo update
## Copy the local configuration file and edit it if necessary
C:\>cp values.local.yaml config.yaml
## Create a dedicated namespace for ingestion components
C:\>kubectl create namespace d4n-ingestion
## Install or upgrade Apache NiFi using Helm with the specified configuration
C:\>helm upgrade --cleanup-on-fail --install nifi cetic/nifi --namespace d4n-ing
estion --version=1.2.1 --values config.yaml
```

🛎️ **If deploying on a local Docker Desktop Kubernetes cluster**

```
## Apply the local load balancer configuration to expose Apache NiFi at
https://localhost:8443
C:\>kubectl apply -f local-loadbalancer.yaml
```

🛎️ **If deploying on a remote server**

```
## Apply the NodePort proxy configuration to expose Apache NiFi at
https://{master_ip_address}:30443
C:\>kubectl apply -f proxy.nodeport.yaml --namespace d4n-ingestion
```

5.3.10 Integrate Active Directory

1 **Apache NiFi:** Apache NiFi has built-in support for user authentication and authorization. You can use NiFi's policies and user groups to define access controls based on roles and permissions. You can also integrate NiFi with external identity providers using OAuth or LDAP for more advanced RBAC scenarios. (Link)

2 **MinIO:** MinIO supports Identity and Access Management (IAM) policies that allow you to define RBAC. You can create users and groups and assign policies that define their access to buckets and objects. MinIO also supports integration with external identity providers like LDAP. (Link)

3 **Trino (PrestoSQL):** Trino offers a robust security framework that includes RBAC. You can configure access control using catalog-level, schema-level, and table-level permissions. Trino supports integration with Hive Metastore and LDAP for user management and authentication. (Link)

4 **JupyterHub:** JupyterHub allows you to integrate with various authentication providers (OAuth, LDAP) and define RBAC for notebooks and resources. You can use tools like Kubernetes RBAC or JupyterHub's built-in authentication mechanisms to control access to Jupyter notebooks. (Link)

Integrate Identity Provider: If possible, integrate a single identity provider or SSO solution to manage user identities and roles centrally. Please see 'Active Directory Distinguished Name (DN)' if you have issue on extracting Distinguish Name from AD.

Remember that RBAC implementation can be complex, especially when dealing with a diverse set of tools and components. Collaboration between IT, security, and application teams is essential to ensure a cohesive and effective RBAC setup.

5.3.11 Using the tools

We are in the process of documenting selected use cases focused on setting access policies in MinIO and building data pipelines in Apache NiFi. These examples aim to demonstrate practical ways the tools can be used in real-world scenarios. In addition to this, many other guides and tutorials are available online.

- Minio use-case: Structure the storage for Labor Force and Living Standard datasets
- Apache NiFi use cases: Dataflow to Ingestion sample data
- Run Python script in jupyterhub to ingest data from an ftp server to MinIO. see "Coding tools"

A. Annexes

A.1 Data4Now: IT Guiding Questionnaire

The main objective of this questionnaire is to Identify requirements of the NSO/NSS in country and the challenges it is facing (mostly focusing on data flow) in technology landscape and is developed and used by Data for Now team. Many times, the human element is left behind in these processes, so a focus on skill analysis and how to fill those gaps should also be identified in the process while following the IT (modernization) strategy of the organization. These challenges should be used as opportunities to fill skill gaps, or process modernization while incorporating new and innovative data sources, methods and tools while modernizing the IT architecture. Add new columns for additional answers. Download IT Guiding Questionnaire

Group	**ID**	**Question**	**Answer 1**	**Answer 2 …**	**Example**
General	g-1	Date			
	g-2	Country			
	g-3	Individual responding to questions (Add as many people as needed separated by semicolon)			*Name/Organization/Title; Name/Organization/Title*
	g-4	Could you briefly describe the Current Context - Existing situation/process?			
	g-5	Who are the Teams/people involved in the process?			
	g-6	Could you share the IT strategy of the organization?			
	g-7	Could you share any IT assessments or previous reports on the IT Infrastructure or IT architecture diagram?			
	g-8	Could you share the IT team structure (organization chart)			
Data Source	ds-1	Who is the owner of the data and what is the dataset?			*NSO, NSS, Ministry Name, Gov agency Name, MNO, Private Company, NGO Name, etc.*
	ds-2	Is there a mandate/agreement/MoU to get Data from source? If yes, could you briefly explain.			
	ds-3	Are there any challenges in data source?			*data access, data quality, data format, data inconsistency, classification/harmonization, etc.*
Tools/technology stack currently used:					

Data Ingestion Process/ tools	di-1	Who is the main person/team/section involved in ingestion of this data?			
	di-2	How do they get Data from Source?			*email, flash-drive, web-scrapping, db-connection (ODBC/JDBC), Data-API(REST/SOAP), sFTP, GraphQL, XBRL, SDMX, Swagger(OpenAPI), etc.*
	di-3	In which format is data source originally received?			*JSON, XML, CSV, Excel, .dat, .stat, .txt, Apache Parquet, Avro, HDF5, other Binary format, etc.*
	di-4	How often do you receive this dataset?			*monthly/annually/ occasionally, etc.*
	di-5	Does it involve geospatial specific data? If yes, how is it handled?			*(shp, geojson, kml, geoTiff, gpkg, gdb – Esri, nc, tab, etc.)*
	di-6	Does the data structure follow any standards like SDMX/DDI? Does it have Metadata?			
	di-7	Do you use any tool/software/code/scripts to ingest data?			*(Python, R, Apache NiFi, Pentaho, Talend, Informatica, etc.)*
	di-8	Did the relevant person/team receive/attended any training in above tool/script or required tool/platform for data ingestion?			
	di-9	Can the ingestion process be automated?			
	di-10	Do you follow the ETL or ELT process?			
	di-11	How is data transformed into the structure/format that you need? Does it happen during the ingestion process or later?			
	di-12	Are there any challenges in data ingestion process including human and technical capacity?			
Data Storage	db-1	Who is the main person/team/section involved in managing the storage section?			
	db-2	What kind of storage media is being used to store the ingested data?			*(Relational-Database, Data Center, File System-NAS, File Server, Data Warehouse, Data Lakes, SPSS/SAS application, Cloud Storage, Geospatial*

					Database (ESRI, QGis), etc.)
	db-4	Who manages security policies and how do you ensure its compliance?			
	db-5	Metadata: Do you use any metadata standards? (DDI, SDMX, ISO 11179, etc.) Any metadata management and dissemination tools?			
	db-6	Are there any challenges in data storage?			
Data Processing	dp-1	Who is the main person/team/section involved in data processing?			
	dp-2	Are there policy/procedures in place to ensure secure access to sensitive data? If yes, could you briefly explain.			*Role-based-access-control, user authentication, used logging and monitoring, Database access control, Threat detection, etc.*
	dp-3	Is there a data pseudonymization/anonymization process in place			
	dp-4	What tools/platforms/language do you use for querying and processing			*Jupiter Notebook, Python, R, SPSS, Stata, qGIS, ArcGIS, etc.*
	dp-5	How are records from different data sources linked (Record linking)?			*(e.g., using unique identifier, fuzzy matching, probabilistic linkage, machine learning algorithm, etc.)*
	dp-6	What are the main products or statistics produced from your statistical registers system			
	dp-7	**Are there any challenges in data Processing?**			
Data Disseminati on	dd-1	Who is the main person/team/section involved in data dissemination?			
	dd-2	What tools/platforms/language do you use for data dissemination			
	dd-3	Did the relevant person/team receive/attend any training in the above tool/platform for data dissemination?			
	dd-4	Are there any challenges in data Dissemination?			

A.2 Data ingestion tools

A.2.1 Coding tools

Python, R, Java, C#, Scala are some of the programming languages used in data ingestion coding framework. Below is an example of python script that ingests data from an ftp server to MinIO.

```
import os
from ftplib import FTP
import boto3
import datetime

def download_from_ftp(ftp_server, username, password, remote_folder, local_filename):
    with FTP(ftp_server) as ftp:
        ftp.login(user=username, passwd=password)
        ftp.cwd(remote_folder)
        with open(local_filename, "wb") as file:
            ftp.retrbinary("filename.csv", file.write)

def upload_to_s3(s3_bucket, local_filename, s3_subfolder):
    s3 = boto3.resource('s3')
    bucket = s3.Bucket(s3_bucket)
    s3_key = os.path.join("raw", s3_subfolder, "filename.csv")
    bucket.upload_file(local_filename, s3_key)

def move_on_ftp(ftp_server, username, password, remote_folder, timestamp):
    with FTP(ftp_server) as ftp:
        ftp.login(user=username, passwd=password)
        ftp.cwd(remote_folder)
        new_folder = os.path.join("2023", timestamp)
        ftp.mkd(new_folder)
        ftp.rename("filename.csv", os.path.join(new_folder, "filename.csv"))

def main():
    ftp_server = "10.0.0.1"
    ftp_username = "your_ftp_username"
    ftp_password = "your_ftp_password"
    s3_bucket = "your_s3_bucket"
    timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")

    # Step 1: Download from FTP and save locally
    download_from_ftp(ftp_server, ftp_username, ftp_password, "/2023", "filename.csv")

    # Step 2: Upload to S3 with timestamped subfolder
    upload_to_s3(s3_bucket, "filename.csv", timestamp)

    # Step 3: Move file within FTP server to a timestamped subfolder
    move_on_ftp(ftp_server, ftp_username, ftp_password, "/2023", timestamp)

    # Clean up local file after processing
    os.remove("filename.csv")

if __name__ == "__main__":
    main()
```

A.2.2 Low code Tools

Some Free and open-source low code tools used by various NSS and available in the market includes Apache NiFi, Airbyte, Pentaho PDI (Kettle), etc.

A.3 Data storage format

A.3.1 Parquet

Apache Parquet is a columnar storage file format optimized for analytical workloads. It is designed to efficiently store large volumes of data and is compatible with multiple data processing frameworks, such as Apache Spark, Apache Hive, Apache Drill, and more.
Parquet stores data in a way that allows selective reading of specific columns, significantly reducing I/O and speeding up data processing. It supports rich data types and schema evolution, making it highly versatile for big data analytics.

Good Practices

- Use Compression Wisely: Employ compression algorithms like Snappy or GZIP, depending on the trade-off between compression speed and size. (Link)
- Partition Data: Partition the dataset by frequent query dimensions (e.g., date or region) to minimize the amount of data scanned during queries.
- Optimize Column Order: Arrange columns in the file according to their usage frequency to optimize query performance.
- Chunk Data into Optimal Sizes: Avoid creating too many small Parquet files as this increases overhead. Target 128 MB to 1 GB file sizes to balance performance and manageability.
- Use Predicate Pushdown: Utilize tools that support predicate pushdown to filter rows during the scan phase, reducing unnecessary reads.
- Store Metadata Efficiently: Leverage the Parquet file's internal metadata structure for schema definitions, minimizing external dependencies.
- Validate Schema Consistency: Ensure consistency in schema when appending new data to avoid compatibility issues.
- Batch Writes: When writing Parquet files, use batch processing to optimize I/O performance and avoid file fragmentation.
- Avoid Nested Structures If Possible: Flatten deeply nested schemas if query performance and simplicity are critical.
- Evaluate Tool Compatibility: Ensure that the tool or framework you are using supports the latest Parquet specification to leverage its full capabilities.

Choosing the right storage format depends on factors like the nature of the data, query performance requirements, data volume, and the tools and technologies used in your data lake ecosystem. It is often a trade-off between storage efficiency, processing speed, schema flexibility, and compatibility with analytics tools.

A.4 Data Virtualization

Data virtualization in data lake architecture involves providing a unified and abstracted view of data from various sources, including structured, semi-structured, and unstructured data, without physically moving or replicating the data. It allows users and applications to access and query data as if it were in a specific location, even though the data might be distributed across different storage systems, databases, or formats. Data virtualization simplifies data access, enhances agility, and reduces the need for complex ETL/ELT processes.

Trino (formerly known as PrestoSQL) and Presto are popular open-source tools for data virtualization and federated querying in data lake architectures. They provide efficient and flexible ways to access and analyze data from multiple sources in real-time. Other open-source options for data virtualization include Apache Drill and Denodo, which offer similar capabilities to enable unified data access across heterogeneous sources within a data lake ecosystem.

1. **Trino (PrestoSQL)**: Trino is a distributed SQL query engine designed for high-performance data processing across a variety of data sources. It can query data from different storage systems like HDFS, cloud object storage, relational databases, and more. Trino enables data virtualization by allowing users to execute SQL queries that span multiple data sources seamlessly. It supports federated queries, which means it can access and join data from various sources as if they were in a unique location. (Link)
2. **Presto**: Presto is another popular open-source distributed SQL query engine that offers similar data virtualization capabilities. It can query data from various data sources, making it possible to perform complex analytics across different storage systems without the need for data movement. (Link)

Example

To run a Trino query on survey data stored in the MinIO subfolder raw/survey/2023, you need to create an external table in Trino that points to the data location in MinIO. Here is an example of how to achieve this:

1. Create an External Table:

Run the following Trino SQL query to create an external table that points to the survey data stored in the MinIO subfolder raw/survey/2023:

```
CREATE TABLE survey_data (
  column1 data_type,
  column2 data_type,
  ... -- Add other columns as per your data schema
)
WITH (
  format = 'ORC', -- Replace with the actual format of your data (e.g., Parquet, CSV, etc.)
  external_location = 's3a://raw/survey/2023/'
);
```

Make sure to replace data_type with the appropriate data types for your columns

2. Query the Data:

Once the external table is created, you can query the survey data using standard SQL queries in Trino:

```
SELECT * FROM survey_data WHERE condition_column = 'some_value';
```

Replace condition_column and 'some_value' with the appropriate filtering conditions based on your survey data requirements.

A.5 Data processing tools

A.5.1 Collaboration Platform – JupyterHub (Server)

JupyterHub brings the power of notebooks to groups of users. It gives users access to computational environments and resources without burdening the users with installation and maintenance tasks. Users including statisticians, researchers, and data scientists - can get their work done in their own workspaces on shared resources which can be managed efficiently by system administrators. It is customizable and scalable, and is suitable for small and large teams, and large-scale infrastructure. (Link)

A.5.2 Collaboration Platform - JupyterLab

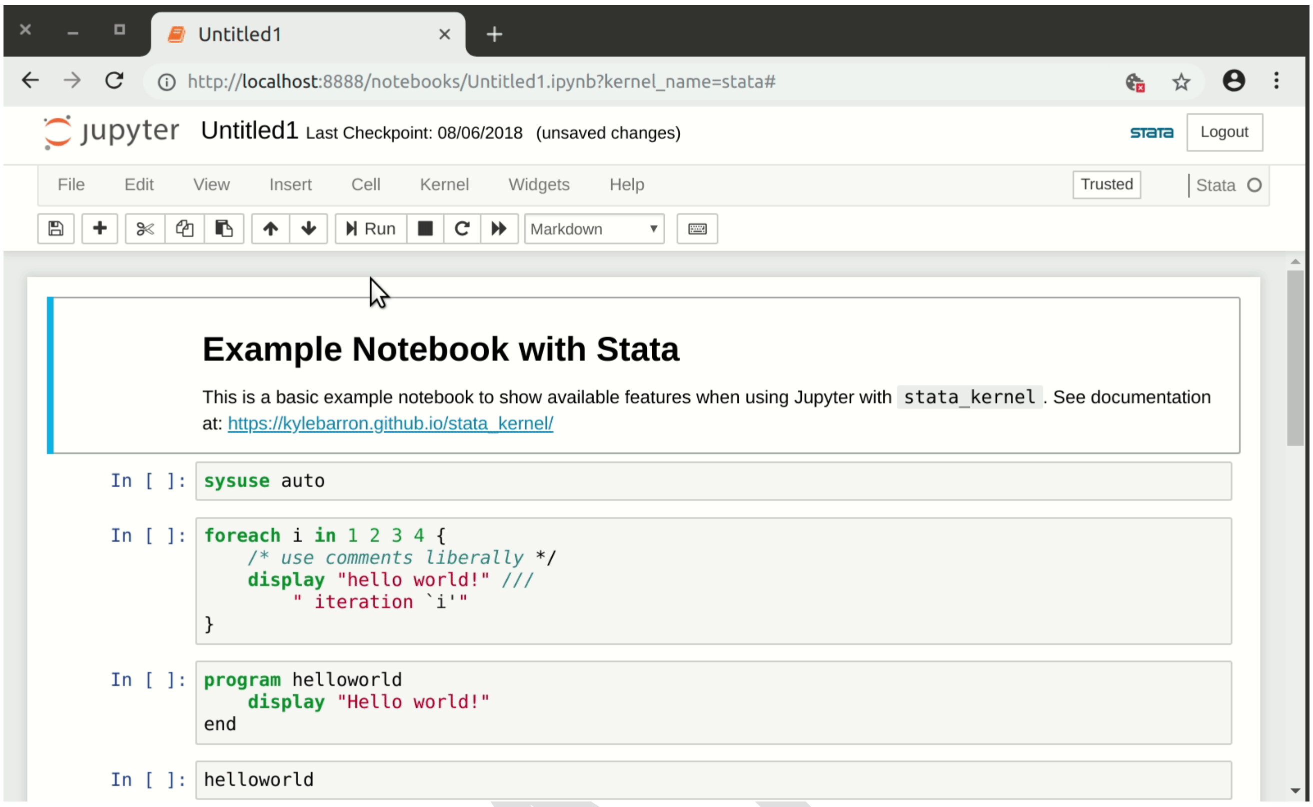
JupyterLab is an open-source, web-based interactive development environment (IDE) for Jupyter notebooks, code, and data. It provides a flexible interface for programming in languages like Python, R, and Julia, enabling data visualization, interactive computing, and reproducible research. It extends the functionality of Jupyter Notebooks with support for multiple panes, extensions, and collaborative features. In addition to Python and R, we can also use Stata and QGIS within JupyterLab as shown below:

Stata (Source)

A licensed version of Stata must already be installed. stata_kernel has been reported to work with at least Stata 13+ and may work with Stata 12.

We can run Stata in a Jupyter Notebook environment using the "Stata Kernel" or by using the "Stata in Jupyter" integration. This allows you to combine the interactive capabilities of Jupyter Notebooks with the statistical and data analysis power of Stata.

Running STATA on jupyterlab may have some limitation compared to running Stata directly in its native environment. You can find more details and limitation in the kernel documentation


Untitled1
http://localhost:8888/notebooks/Untitled1.ipynb?kernel_name=stata#
jupyter Untitled1 Last Checkpoint: 08/06/2018 (unsaved changes)
stata
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Stata
Run
Markdown

Example Notebook with Stata

This is a basic example notebook to show available features when using Jupyter with `stata_kernel`. See documentation at: https://kylebarron.github.io/stata_kernel/

```
In [ ]: sysuse auto
```

```
In [ ]: foreach i in 1 2 3 4 {
            /* use comments liberally */
            display "hello world!" ///
                " iteration `i'"
        }
```

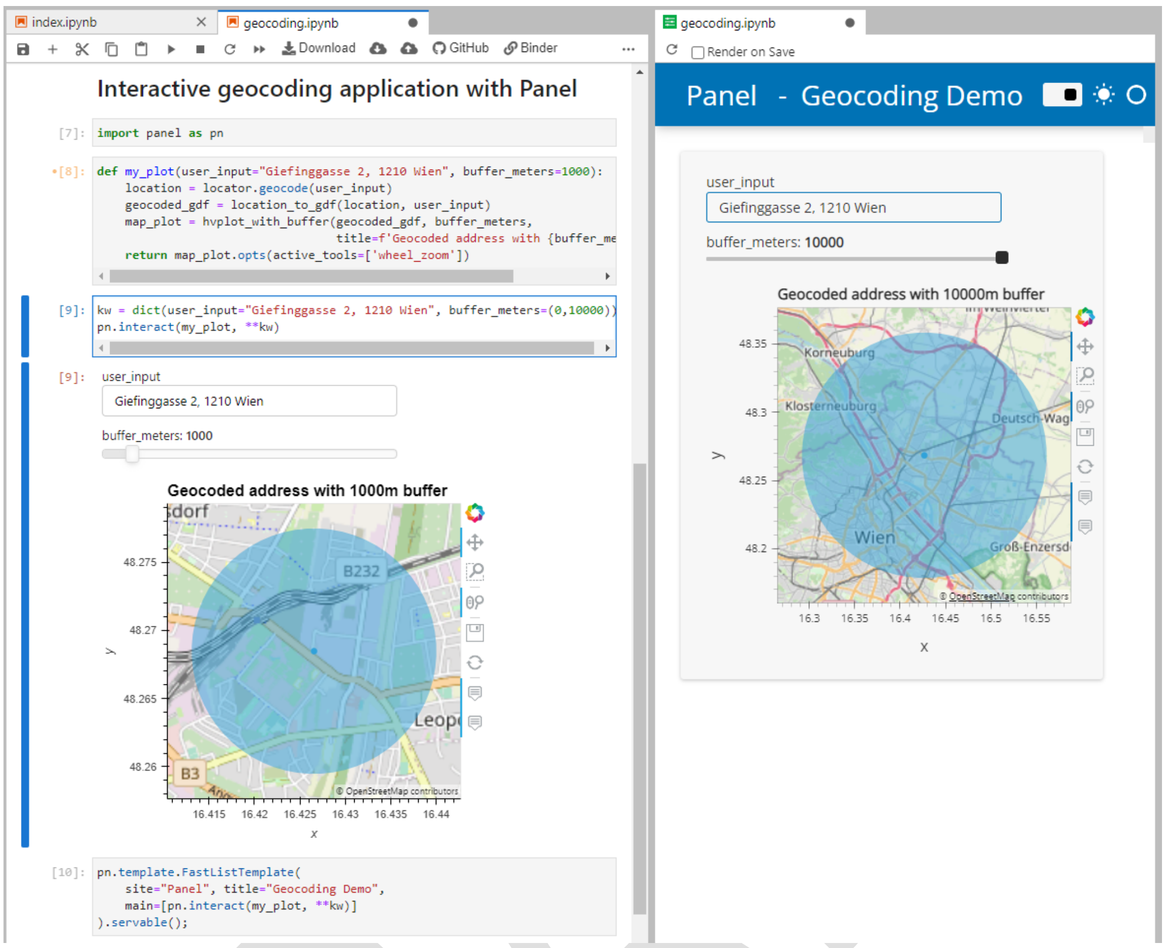
```
In [ ]: program helloworld
            display "Hello world!"
        end
```

```
In [ ]: helloworld
```

Geo-spatial analysis

QGIS, which stands for "Quantum Geographic Information System," is an **open-source** geographic information system (GIS) software that allows users to create, analyze, visualize, and manage geographic and spatial data. It provides a powerful platform for working with several types of spatial data, such as maps, satellite imagery, GPS data, and more.

It is possible to run QGIS within a Jupyter Notebook environment using PyQGIS to interact with QGIS from within a Python script, and you can execute these scripts in a Jupyter Notebook. This approach allows you to perform geospatial analysis, data manipulation, and more using the QGIS functionality exposed through PyQGIS. Source



Interactive geocoding application with Panel

```
[7]: import panel as pn
```

```
•[8]: def my_plot(user_input="Giefinggasse 2, 1210 Wien", buffer_meters=1000):
          location = locator.geocode(user_input)
          geocoded_gdf = location_to_gdf(location, user_input)
          map_plot = hvplot_with_buffer(geocoded_gdf, buffer_meters,
                                        title=f'Geocoded address with {buffer_me
          return map_plot.opts(active_tools=['wheel_zoom'])
```

```
[9]: kw = dict(user_input="Giefinggasse 2, 1210 Wien", buffer_meters=(0,10000))
     pn.interact(my_plot, **kw)
```

[9]: user_input: Giefinggasse 2, 1210 Wien

buffer_meters: 1000

Geocoded address with 1000m buffer

```
[10]: pn.template.FastListTemplate(
          site="Panel", title="Geocoding Demo",
          main=[pn.interact(my_plot, **kw)]
      ).servable();
```

user_input: Giefinggasse 2, 1210 Wien

buffer_meters: 10000

Geocoded address with 10000m buffer

A.5.3 Framework - Apache spark

Apache Spark is an open-source unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

(Source: https://spark.apache.org/docs/latest/)

A.5.4 Framework – Dask

Dask is an open-source Python library that allows users to perform parallel computing on large data sets and machine learning tasks. It provides distributed computing for Python (https://www.dask.org/)

A.5.5 DuckDB

DuckDB is an open-source high-performance analytical database system. It is designed to be fast, reliable, portable, and easy to use. DuckDB provides a rich SQL dialect, with support far beyond basic SQL. DuckDB supports arbitrary and nested correlated subqueries, window functions, collations, complex types (arrays, structs, maps), and several extensions designed to make SQL easier to use. DuckDB is available as a standalone CLI application and has clients for Python, R, Java, Wasm, etc., with deep integrations with packages such as pandas and dplyr. (Source - https://github.com/duckdb/duckdb)

A.5.6 Language - Python

A versatile language with libraries like Pandas, NumPy, SciKit-learn, statsmodels, dask, etc. which are widely used for data manipulation, analysis, and statistical computations. You can run Dask on: Laptops, Kubernetes, HPC job schedulers, Cloud SaaS services, and Legacy Hadoop/Spark clusters.

A.5.7 Language - R

A language dedicated to statistical computing and graphics with popular packages like dplyr, tidyverse and sf for geospatial analysis. It's favored by statisticians and data analysts for its extensive collection of statistical packages.

A.5.8 OpenRefine

An open-source tool for data cleaning and transformation. It provides features for data profiling, data enrichment, and reconciling inconsistencies.

A.5.9 Geospatial - QGIS

QGIS, formerly known as Quantum GIS, is a free, open-source geographic information system (GIS) software that allows users to view, edit, analyze, and publish spatial data. QGIS is a free alternative to proprietary GIS software like ESRI's ArcGIS products. It has similar functions and features, and supports a variety of spatial data file extensions, including .shp, .tif, .csv, and .img. QGIS is compatible with Linux, Unix, Mac, and Windows. (Source – www.qgis.org)

A.6 Data orchestration tools

A.6.1 Dagster:

Dagster is a data orchestrator specifically designed to address the challenges of data quality, testing, and monitoring in complex data workflows. It focuses on building reliable, testable, and maintainable data pipelines while emphasizing data lineage, observability, and data quality assurance.

Key features of Dagster:

- **Data Quality Assurance**: Dagster provides built-in features for data testing, validation, and quality assurance. It ensures that data flowing through the pipeline meets defined criteria and expectations.
- **Type System**: Dagster uses a strong type of system to ensure that data transformations are correctly and consistently defined, helping prevent runtime errors.
- **Modularization**: Dagster encourages modularization of data pipeline components, making it easier to test and maintain individual components.
- **Data Lineage**: Dagster tracks data lineage, meaning you can trace where data came from and where it goes in your pipeline. This is essential for understanding and troubleshooting data issues.

- **Orchestration**: While Dagster itself does not handle scheduling and execution, it can integrate with other orchestration tools like Apache Airflow or Prefect for running pipeline executions.

A.6.2 Apache Airflow:

Apache Airflow is a general-purpose workflow scheduler that allows you to define, schedule, and execute complex workflows as directed acyclic graphs (DAGs). It is widely used for orchestrating several types of tasks, including data processing, ETL, and more.

Key features of Apache Airflow:

- **Workflow Orchestration:** Airflow is focused on orchestrating workflows and automating task scheduling and executing in various scenarios.
- **Dynamic DAGs**: Airflow supports dynamically generating DAGs and tasks based on configurations, which can be useful for handling varying workloads or dynamic data sources.
- **Scheduling**: Airflow provides powerful scheduling capabilities for executing tasks at specific times or intervals.
- **Extensibility**: Airflow allows you to extend its functionality through custom operators, hooks, and plugins, enabling integration with a wide range of systems and services.
- **Community and Ecosystem**: Apache Airflow has a large and active community, resulting in a rich ecosystem of contributed operators, plugins, and integrations.

A.7 Security and authorization

A.7.1 Functionalities to consider in security

1. **Data Anonymization/Pseudonymization:** Anonymizing or pseudonymizing sensitive data before sharing allows you to provide access to data without revealing the identities of individuals. This approach maintains privacy while enabling analysis.
2. **Data Masking/Tokenization:** Replace sensitive data with masked or tokenized versions while preserving the data format. This approach is useful when the actual data is not required for analysis, but the structure is.
3. **Secure Data Sharing Platforms:** Implement secure data sharing platforms that enforce strict access controls and encryption for data in transit and at rest. These platforms often allow fine-grained control over who can access which parts of the data.
4. **Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC):** Implement RBAC and ABAC mechanisms to ensure that users can only access data and perform actions that are relevant to their roles and responsibilities.

5. **Data Virtualization:** Data virtualization tools allow users to access data from various sources without having direct access to the underlying data. This provides a layer of abstraction while maintaining control over data access.
6. **Data Sharing Agreements:** Establish clear data sharing agreements with external parties, defining the purposes for which the data will be used, the security measures that need to be in place, and the responsibilities of both parties.
7. **Secure APIs:** Provide controlled access to data through secure APIs. This enables users to retrieve the data they need without direct access to the underlying database.
8. **Time-Limited Access:** Grant temporary access to data for specific purposes and timeframes. Once the designated period is over, the access is revoked.
9. **Data Minimization:** Share only the minimum amount of data necessary for the intended analysis or purpose, reducing the risk of exposing sensitive information.
10. **Data Governance and Auditing:** Implement robust data governance practices, including tracking data access and usage. Regular audits help ensure that data is being used appropriately, and that compliance is maintained.
11. **Consent Management:** If applicable, obtain explicit consent from data subjects for sharing their data. Implement mechanisms to manage and track consent preferences.
12. **Encryption and Key Management:** Encrypt data before sharing it and manage encryption keys securely. This way, even if unauthorized access occurs, the data remains unintelligible without the proper decryption keys.
13. **Secure Collaboration Tools:** Utilize secure collaboration and analytics platforms that provide a controlled environment for sharing and analyzing data without exposing the raw data to users.
14. **Education and Training:** Educate users about the importance of data privacy and security, ensuring they understand their responsibilities and the potential risks associated with mishandling data.
15. **Regular Risk Assessments:** Conduct regular risk assessments to identify potential vulnerabilities and address them before they lead to security breaches.
16. **Data Classification:** Classify data based on its sensitivity and restrict access accordingly. Different levels of data may have different access requirements.

Remember that each organization's situation is unique, so it's important to tailor these approaches to your specific data, user base, and compliance requirements. Collaboration between IT, legal, compliance, and business stakeholders is crucial to successfully implement these approaches while maintaining security and privacy.

A.7.2 Access Control

Focusing more from Access Control, Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) are access control mechanisms that help manage user access to data and resources in a way that aligns with security and compliance requirements.

Role-Based Access Control (RBAC): In an RBAC system, access is determined based on the roles that users hold within the organization. Each role has associated permissions that define what actions a user with that role can perform. Users are then assigned roles, and their access rights are determined by the permissions associated with those roles. RBAC simplifies access management by grouping users into roles and applying permissions at the role level.

Attribute-Based Access Control (ABAC): ABAC is a more flexible access control model that considers various attributes and conditions when making access decisions. In ABAC, access is granted or denied based on the values of attributes associated with users, resources, and the environment, along with predefined policies.

Feature	**Role-Based Access Control (RBAC)**	**Attribute-Based Access Control (ABAC)**
Core Concept	Access based on predefined roles	Access based on attributes/policies
Decision Basis	Who you are (role)	Who, what, when, where, why (context)
Complexity	Simpler to implement and manage	More complex implementation
Flexibility	Limited (fixed role definitions)	Highly flexible (dynamic decisions)
Granularity	Coarse-grained	Fine-grained
Scalability	Role explosion in complex environments	Scales better for complex scenarios
Policy Expression	"User A has Role X"	"Users with Attribute B can access Resource C under Condition D"
Implementation Effort	Lower	Higher
Maintenance	Easier for simple systems, harder as roles multiply	Complex initially, but can be more manageable long-term
Typical Use Cases	SMBs, simpler access needs	Large enterprises, dynamic environments
Dynamic Context	Limited or none	Extensive (time, location, device, etc.)
Examples	Active Directory groups, traditional file permissions	XACML, AWS IAM policies, Okta ABAC
Role/Attribute Management	Centralized role assignment	Distributed attribute collection
Audit Complexity	Easier to audit	More complex audit trails
Risk Management	Less precise, may lead to over-permissioning	More precise control reduces risk

It's worth noting that some systems combine both RBAC and ABAC principles to create a hybrid approach that leverages the strengths of each model to provide a balance between simplicity and flexibility in access control.

In addition to that, the organization must also implement:

- **Authentication and Identity management**: Verify user identity before any interaction with the platform and limit access to tools based on user roles. A regular review and update of access permissions ensures they're appropriate and up-to-date according to organization employees.
- **Authorization and access control**: The recommendation is to define access policy based on user groups(roles) and assign uses to the various groups for users and groups. Then define an access control list related to groups and users and apply to the resources available on the data lake.
- **Network security**: You can establish firewalls and define an IP address range for your trusted clients. You can also segment your network to isolate the data lake environment from other systems.
- **Data protection**: The data lake provides encryption mechanisms to protect the data. Data in transit can be secure using the industry-standard Transport Layer Security (TLS 1.2) protocol to secure data over the network.
- **Auditing**: Implement comprehensive auditing mechanisms to track user activities and data access. Set up real-time monitoring to detect suspicious activities and potential security breaches

A.7.3 Tools

Centralized access management through **Active Directory** (AD) and managing roles and access at the **application level** both have their own advantages and disadvantages. The choice between these approaches often depends on the specific needs and context of your organization. Let's examine the advantages and disadvantages of each approach and how they relate to implementing **role-based access**:

Central Access Management through Active Directory:

Advantages:

1. **Centralized Control:** Active Directory provides a centralized location for managing user identities, authentication, and access control across multiple applications. This can streamline administration and reduce the risk of inconsistencies.
2. **Single Sign-On (SSO):** Active Directory allows for single sign-on, enabling users to authenticate once and access various applications without the need for multiple logins. This improves user experience and security.
3. **Scalability:** AD is designed to handle large numbers of users and resources, making it well-suited for enterprises with complex access management requirements.
4. **Integration:** Many enterprise applications and services can integrate directly with Active Directory for authentication and authorization, simplifying access management.

Disadvantages:

1. **Dependency:** A centralized approach means that if there's a problem with the Active Directory infrastructure, it could impact access to multiple applications.
2. **Limited Granularity:** While Active Directory supports role-based access to some extent, it might not provide the same level of fine-grained access control as application-level management.

Managing Role and Access at the Application Level:

Advantages:

1. **Granular Control:** Managing access at the application level allows for more fine-grained control over who has access to specific features and functions within an application.
2. **Flexibility:** Application-level access control is flexible and can be tailored to the specific needs of each application. This can be particularly useful when applications have varying access requirements.
3. **Isolation:** If a particular application experiences issues with its access management, it won't necessarily affect the access controls of other applications.

Disadvantages:

1. **Complexity:** Managing access at the application level can become complex, especially in organizations with numerous applications. It can lead to duplicated efforts and inconsistencies in access policies.
2. **Increased Administration:** Each application's access management needs to be set up and maintained separately, which can be time-consuming and resource-intensive.
3. **Security Risks:** If access control isn't well-implemented at the application level, there's a risk of security vulnerabilities or misconfigurations.

ISN working on windows environment already used an on-premises version of AD (LDAP). Though some open-source alternatives exist such as: OpenLDAP or 389 Directory Server

User roles and groups on data lake involves defining access permissions and privileges for different users based on their responsibilities and needs. Here are some examples of user roles and groups that could be configured

Role/group	**Description**	**Tools**	**Bucket**
administrators	- Managing user accounts, configuring security settings - Maintaining infrastructure: Update, Upgrade etc. - Ensuring metadata management, backups	All	All
Data engineer	- Designing and maintaining data pipelines, ETL processes, data integration - Create and manage data pipeline and ETL-related	- Apache NiFi - Dagster - Jupyterlab	- raw -

Role/group	Description	Tools	Bucket
	folders - Permissions to create and manage scheduled workflows	- MinIO - Trino	anonymized - staging
Data steward	- Ensuring data anonymization, adhering to data governance policies	- Trino - MinIO - JupyterLab	- raw - anonymized - staging - aggregated
Data analyst	- Analyzing and generating insights from data, creating reports and visualizations. - Developing and maintains data models - Create and run custom queries and scripts	- MinIO - Trino - Jupyterlab (R, Python, QGIS)	-staging - aggregated
Public access	- Accessing publicly available datasets or reports - Read-only access to publicly accessible datasets and reports	- Trino - Jupyterlab	- aggregated

A.7.3.1 Active Directory

Active Directory (AD) is Microsoft's directory and identity management service for Windows domain networks. It was introduced in Windows 2000, is included with most MS Windows Server operating systems, and is used by a variety of Microsoft solutions like Exchange Server and SharePoint Server, as well as third-party applications and services.
Source Useful link: what is LDAP authentication

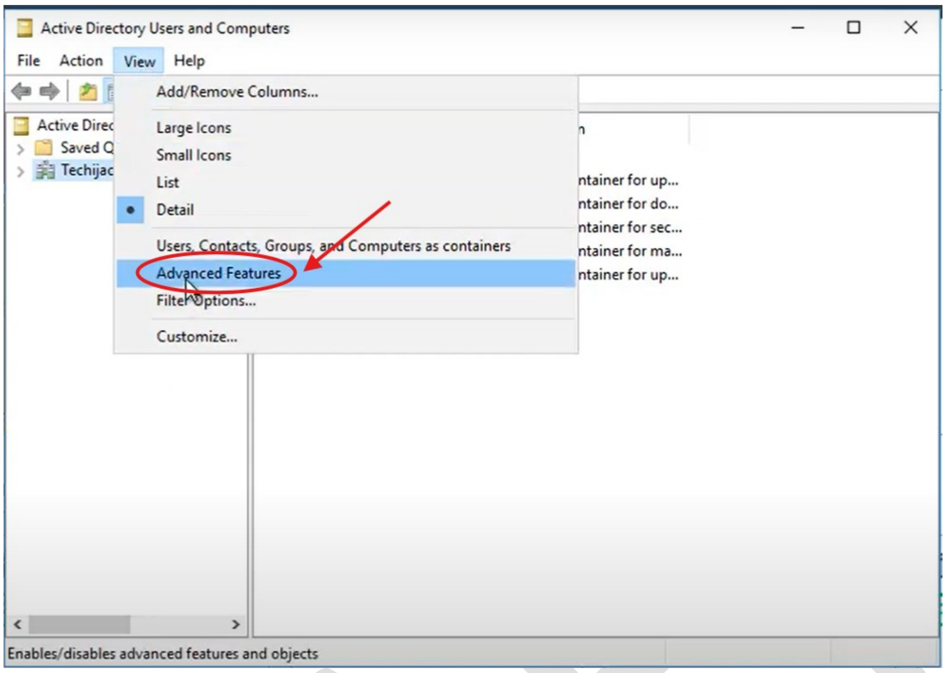
A.7.3.2 OpenLDAP

OpenLDAP Software is an open-source implementation of the Lightweight Directory Access Protocol(LDAP).LDAP stands for Lightweight Directory Access Protocol. As the name suggests, it is a lightweight protocol for accessing directory services, specifically X.500-based directory services. LDAP runs over TCP/IP or other connection-oriented transfer services. The nitty-gritty details of LDAP are defined in RFC2251 "The Lightweight Directory Access Protocol (v3)" and other documents comprising the technical specification RFC3377. This section gives an overview of LDAP from a user's perspective.
Source

A.7.4 Active Directory Distinguished Name (DN)

The connection between components and Active Directory for user identification relies on the distinguished name (DN) of users, groups, and organizational units. The DN uniquely identifies each entry in the directory. Below is the process for retrieving the DN of an organizational unit, user, or group.

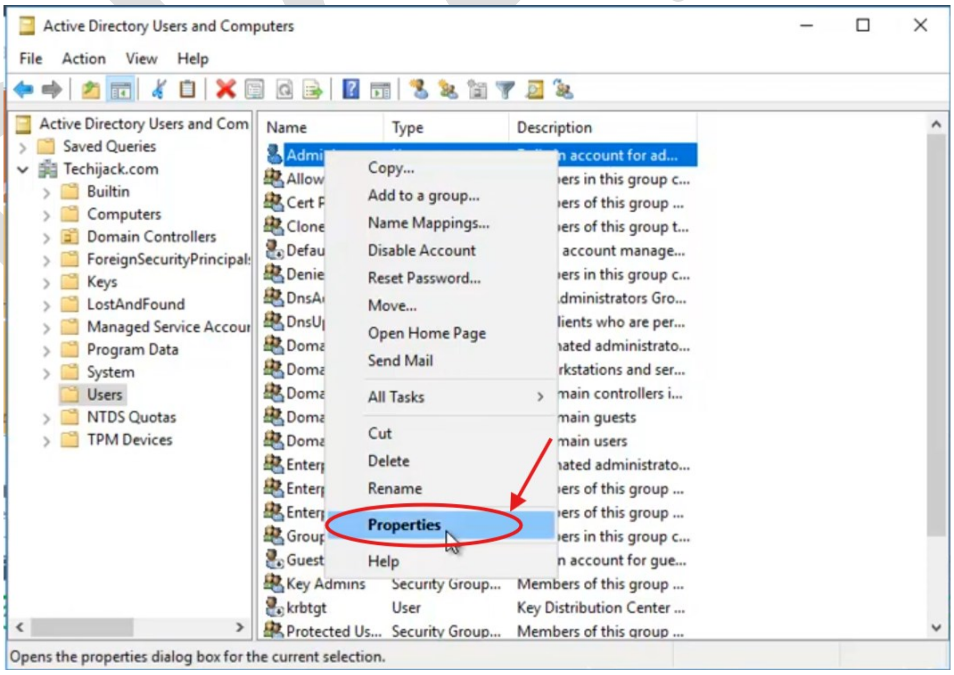
To retrieve DN you first need to enable View->Advanced Features



Enables/disables advanced features and objects

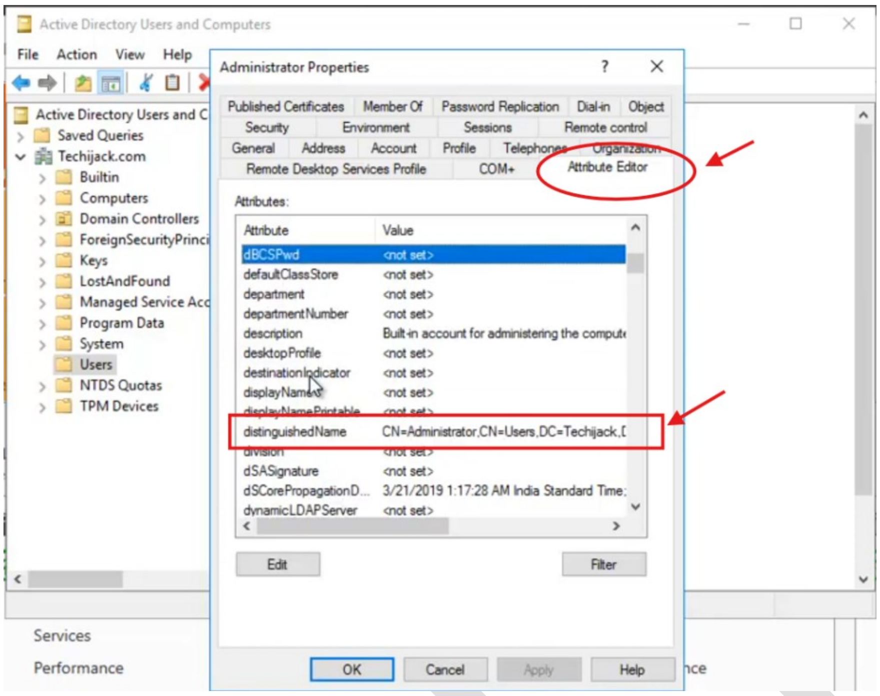
FIGURE A-1 ACTIVE DIRECTORY ADVANCE FEATURE

This brief tutorial 🎬 will guide you on how to retrieve the DN for a user, group, or organizational unit. Below step-by-step diagram shows how to retrieve the DN for user.



Opens the properties dialog box for the current selection.

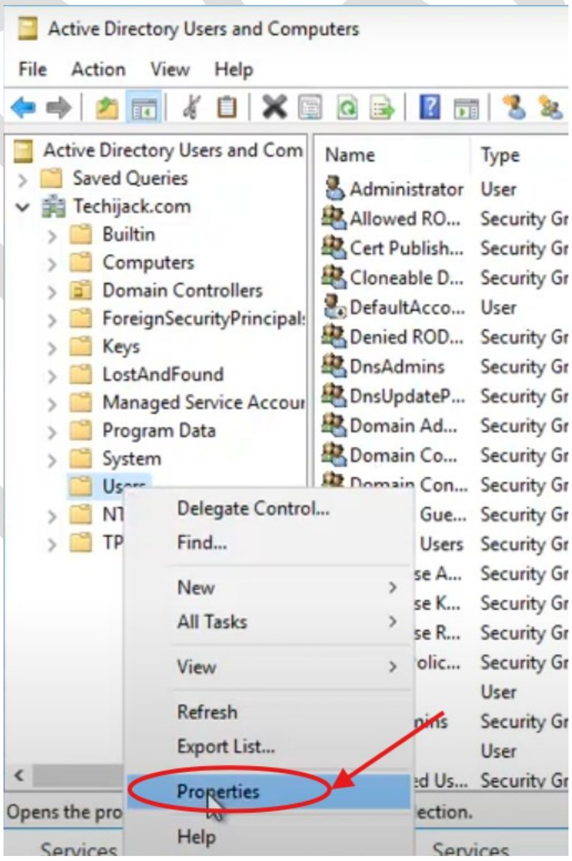
Right click the user and select properties as shown in above figure. In the properties, open the 'Attribute Editor' tab and find 'distinugushedName' property as shown in below figure.



Attribute	Value
dBCSPwd	<not set>
defaultClassStore	<not set>
department	<not set>
departmentNumber	<not set>
description	Built-in account for administering the comput
desktopProfile	<not set>
destinationIndicator	<not set>
displayName	<not set>
displayNamePrintable	<not set>
distinguishedName	CN=Administrator,CN=Users,DC=Techijack,[
division	<not set>
dSASignature	<not set>
dSCorePropagationD...	3/21/2019 1:17:28 AM India Standard Time;
dynamicLDAPServer	<not set>

FIGURE A-2 ACTIVE DIRECTORY DISTINGUISHED NAME FOR USER

Similarly, you can retrieve the User Group DN. Just right click on the group as shown in below figure and follow same step as above by selecting 'Attribute editor' tab and find 'distinugushedName' property.



A.8 Kubernetes cluster

A Kubernetes cluster is a production-grade container orchestration platform that automates the deployment, scaling, and management of containerized applications. In a DevOps context, it serves as the foundation for implementing continuous deployment, microservices architecture, and cloud-native applications. Using a Kubernetes cluster is not mandatory for deploying data platforms or other infrastructure components. Whether you should use Kubernetes depends on your specific requirements and goals. Here's a breakdown of the advantages and disadvantages of using Kubernetes, as well as recommendations for when to use it.

Advantages of Using Kubernetes

1. Scalability and High Availability
   - Automatic scaling of applications based on demand
   - Built-in load balancing and distribution
   - Self-healing capabilities for failed containers
   - Multi-zone and multi-region deployment support
2. DevOps Integration
   - Declarative configuration management
   - Integration with CI/CD pipelines
   - Rolling updates and rollbacks
   - Infrastructure as Code (IaC) support
3. Resource Optimization
   - Optimized container scheduling and placement
   - Resource quota management
   - Automated bin packing
   - Cost optimization through resource sharing

Disadvantages of Using Kubernetes

1. Operational Complexity
   - Steep learning curve for teams
   - Complex networking and security configurations
   - Requires specialized expertise for maintenance
   - Monitoring and troubleshooting challenges e.g., "Complexity in debugging issues across distributed systems."
2. Resource Requirements
   - Significant infrastructure overhead for small applications
   - Higher operational costs for small-scale deployments
   - Memory and CPU intensive for the control plane
3. Security Considerations
   - Complex security configuration requirements
   - Multiple attack surfaces to protect

- Certificate management overhead
- Regular security updates needed

Kubernetes offers significant benefits for managing complex, containerized applications but comes with added complexity and costs. Whether you should use Kubernetes depends on your specific use case, application complexity, and operational capacity. For simpler deployments or for those new to container orchestration, alternative approaches or managed services may offer a more practical starting point. For large-scale, complex applications requiring advanced orchestration and automation, Kubernetes can provide powerful solutions.

Below are few tools available to deploy/manage Kubernetes cluster

A.8.1 Kubeadm

Kubeadm is the official Kubernetes cluster bootstrapping tool maintained by the Kubernetes community. It focuses specifically on the core cluster initialization and management tasks. Its main strengths lie in its widespread adoption, extensive documentation, and strong community support. The tool excels at providing a standardized way to create conformant clusters. However, kubeadm requires more manual configuration for additional components and can involve more complex setup procedures compared to more automated solutions.

Advantages:

- Official Kubernetes tool with extensive community support
- Well-documented and standardized approach
- Greater flexibility in cluster configuration
- Strong security practices and regular updates
- Wide ecosystem compatibility

Disadvantages:

- A more complex initial setup process
- Requires additional tools for complete cluster management e.g., "Third-party tools for logging, monitoring, or networking."
- Manual configuration needed for many add-ons
- Steeper learning curve for beginners

A.8.2 KubeKey

KubeKey is an open-source installer developed by KubeSphere that provides a more streamlined approach to deploying Kubernetes clusters. It combines the installation of Kubernetes and related cloud-native tools into a single process. The primary advantages of KubeKey include its simplified deployment process, built-in support for various add-ons and components, and the ability to manage the full lifecycle of clusters. However, it does have some limitations, such as being less widely adopted in the community compared to kubeadm and having fewer troubleshooting resources available.

Advantages:

- Automated installation of both Kubernetes and common add-ons
- Simplified cluster lifecycle management
- Includes built-in support for air-gapped (offline) environments.
- Easier integration with KubeSphere and related tools
- More streamlined upgrade process

Disadvantages:

- Smaller community compared to kubeadm
- Limited troubleshooting resources
- Less flexibility for customizing individual components
- KubeKey's features are closely integrated with KubeSphere, potentially limiting its appeal for standalone Kubernetes users.

A.9 Kubernetes cluster management platform

A.9.1 Rancher

Rancher is an enterprise-grade Kubernetes management platform that enables organizations to run containers across multiple clusters. It provides a unified control plane for managing both on-premises and cloud-based Kubernetes deployments.

A.9.2 KubeSphere

KubeSphere is a distributed operating system for cloud-native application management, providing a more application-centric approach to cluster management. It emphasizes ease of use and includes features specifically designed for DevOps workflows.

A.9.3 Lens

Lens is an advanced integrated development environment (IDE), and management interface specifically designed for working with Kubernetes clusters. Unlike Rancher or KubeSphere which are installed in Kubernetes cluster, Lens is installed in admin workstation that saves resources of the cluster. It provides a sophisticated graphical user interface that simplifies cluster management and monitoring tasks.

Lens serves as a unified platform for developers and operators to manage multiple Kubernetes clusters through a desktop application. It offers real-time cluster insights, resource management capabilities, and integrated terminal access, making it significantly easier to interact with Kubernetes environments compared to command-line tools alone.

Feature	**Rancher**	**KubeSphere**	**Lens**
Type	Full Kubernetes management platform	Kubernetes distribution & management platform	Kubernetes IDE & management tool
Deployment	Self-hosted (server-based)	Self-hosted (runs on Kubernetes)	Desktop application
Interface	Web-based UI	Web-based UI	Desktop application
Scope	Multi-cluster management	Multi-cluster management	Multi-cluster management

Feature	Rancher	KubeSphere	Lens
Installation Complexity	Moderate	Moderate to high	Low (desktop installation)
Primary Use Case	Enterprise Kubernetes management	All-in-one platform with integrated tools	Developer & operator workstation tool
Multi-Tenancy	Strong	Excellent (core feature)	Limited
User Management	Comprehensive	Comprehensive	Basic
Application Catalog	Yes (App Catalog)	Yes (App Store)	No
Observability	Basic + integrations	Comprehensive built-in	Basic + extensions
CI/CD	Via integrations	Built-in (DevOps Project)	No
Service Mesh	Via integrations	Built-in (Istio)	No
Cluster Provisioning	Strong (multiple providers)	Limited	No (connects to existing)
Developer Experience	Good	Good	Excellent (core focus)
Operations Focus	Strong	Strong	Moderate
Extensions/Plugins	Yes	Yes	Yes (extensive)
Resource Requirements	Moderate	High	Low (local only)
Licensing	Open Source (Apache 2.0)	Open Source (Apache 2.0)	Open Source + Commercial

A.9.4 Alternatives

1. **Portainer:** Portainer is an open-source management UI for Docker and Kubernetes. It provides a simple and easy-to-use interface for managing containerized applications.

2. **OpenShift Origin (OKD):** OKD (OpenShift Origin) is the open-source upstream project of Red Hat OpenShift. It provides a Kubernetes-based container platform with additional features for application development and deployment.

A.9.5 Lightweight Alternatives

1. **K3s:** K3s is a lightweight Kubernetes distribution designed for resource-constrained environments and edge computing.

2. **Minikube:** Minikube is a tool for running a local Kubernetes cluster on your machine. It's ideal for development and testing purposes

3. **MicroK8s:** MicroK8s is a lightweight, single-node Kubernetes distribution developed by Canonical (the creators of Ubuntu).

Conclusion

The choice between these alternatives depends on your specific needs:

- **For Comprehensive Management:** If you need extensive features and enterprise-grade capabilities, tools like Rancher or OpenShift Origin (OKD) offer robust solutions.
- **For Simplicity and Lightweight Environments:** If you need a more lightweight or local development solution, K3s, Minikube, or MicroK8s might be more suitable.

A.10 Minimum data lake technology stack for Data4Now deployment checklist

Download checklist

N°	Steps	Workstation	Master Node	Worker(s) Node	%completed
0%					
1- Prerequisites					
This section outlines the essential prerequisites for deploying the minimum Data lake infrastructure, ensuring that all necessary configurations are in place. It details the collection of critical network and system information, such as node IP addresses and repository links, to facilitate a structured deployment process. The workstation setup includes the installation of essential tools and the configuration of environment variables to maintain a standardized deployment environment. Furthermore, it provides step-by-step instructions for downloading required binaries, setting up working directories, and configuring both master and worker nodes.					
	1.1- Resource Allocation and Component Distribution	☐ DONE			
	1.2- Collect Essential Information	☐ DONE			
	1.3- Workstation Setup				
	1.3.1- Install the Essential Tools: Git, Lens, Docker-Desktop	☐ DONE			
	1.3.2- Establishment of Working Directories	☐ DONE			
	1.3.3- Configuration of Environment Variables	☐ DONE			
	1.3.4- Downloading Essential Binaries and Cloning the Repository	☐ DONE			
	1.4- Configuration on Master Node		☐ DONE		
	1.5- Configuration on Worker Node			☐ DONE	
2- Server Preparation					0%
This section outlines the necessary steps to prepare all nodes in the infrastructure for deployment. The commands below ensure system updates, disable swap memory, and configure kernel parameters essential for Kubernetes operations.					
	Server Preparation		☐ DONE	☐ DONE	
3- Container runtime					0%
This section details the installation and configuration of the container runtime, which is a fundamental requirement for Kubernetes nodes. The following steps ensure a reliable and efficient installation of Docker and cri-dockerd.					
	Install docker and cri-docker		☐ DONE	☐ DONE	
4- Secure Shell (SSH) Access Configuration					0%
This section outlines the necessary steps to establish secure and password-less SSH access between the master node and worker nodes. This setup facilitates seamless remote management and communication between the nodes in a Kubernetes cluster.					
	4.1- Generating an SSH Key		☐ DONE		
	4.2- Distributing the SSH Key		☐ DONE		
5- Install the Kubernetes Cluster					0%

N°	Steps	Workstation	Master Node	Worker(s) Node	%completed
This section provides a structured approach to setting up a Kubernetes cluster, including installing necessary dependencies, configuring the cluster, deploying it, and setting up a metrics server for monitoring resource utilization.					
	5.1- Installation of Required Packages		☐ DONE		
	5.2- Creating and Editing the Cluster Configuration		☐ DONE		
	5.3- Deploying the Kubernetes Cluster		☐ DONE		
	5.4- Deploying the Metrics Component		☐ DONE		
6- Install the Persistent Storage					0%
This section outlines the necessary steps to install and configure persistent storage using Longhorn. These steps ensure that each node is equipped with the required dependencies, and that Longhorn is correctly deployed and verified within the Kubernetes cluster.					
	6.1- Installation of Required Packages for Longhorn		☐ DONE		
	6.2- Installation of Longhorn		☐ DONE		
	6.3- Verification of Longhorn Deployment		☐ DONE		
7- Configuring Kubectl on the Workstation					0%
This section provides guidance on configuring `kubectl` on the workstation, ensuring connectivity with the Kubernetes cluster. The process varies based on whether `kubectl` is newly installed or if an existing configuration is present.					
	7.1- Configuration for a Newly Installed Kubectl				
	7.1.1- Download the kubernetes config file from the cluster	☐ DONE			
	7.1.2- Edit the Kubeconfig File	☐ DONE			
	7.1.3- Verify the access to the cluster	☐ DONE			
	7.2- Configuration If an Existing Cluster is Already Set Up				
	7.2.1- Download the kubernetes config file from the cluster	☐ DONE			
	7.2.2- Modifying the `new-config` File	☐ DONE			
	7.2.3- Merge both the new and current config files	☐ DONE			
	7.2.4- Verify the access to the cluster	☐ DONE			
8- Clean-Up Procedures					0%
This section provides guidance on securing the server environment by disabling root SSH login on all nodes, including both master and worker nodes. Restricting root access enhances security and mitigates unauthorized access risks.					
	8.1- Disabling Root SSH Login		☐ DONE	☐ DONE	
9- Deploy the Minimum Data lake					0%
This section outlines the step-by-step procedure for deploying the essential components of a data lake, including MinIO for storage, JupyterHub for analytics, and Apache NiFi for data ingestion. The deployment is performed from the workstation onto a Kubernetes cluster.					
	9.1- Deploy MinIO (storage)	☐ DONE			
	9.2- Deploy JupyterHub (analytics)	☐ DONE			
	9.3- Deploy Apache NiFi (ingestion)	☐ DONE			

A.11 Sample skill development recommendations for IT teams

Foundational skills:

- Linux system administration (Shell scripting, package management, user permissions, system monitoring, etc.)
- Networking basics (IP addressing, DNS, firewalls, SSH, VPN, etc.)
- Version control (Git basics, branching, collaboration workflows, etc.)

Platform and infrastructure

- Docker & Kubernetes (Containerization, Helm charts, kubectl, cluster management, etc.)
- Storage Management (MinIO policies, S3 APIs, persistent volumes, Longhorn, etc.)
- Monitoring & Logging (basic alerting, Lens IDE, Rancher Academy, Prometheus, Grafana, etc.)

Data engineering and integration

- Apache NiFi (Flow design, processors, templates, scheduling, etc.)
- Data formats and API (JSON, Parquet, ORC, data transfer protocols, etc.)
- ETL/ELT concepts (Data loading strategies, data cleaning, data transformation, etc.)

Data analysis and visualization

- JupyterHub & Notebooks (Python/R scripting, pandas, matplotlib, seaborn, etc.)
- Data Science Basics (Descriptive stats, basic ML, data storytelling, etc.)
- SQL & Trino (Querying structured/unstructured data, federated queries)

Security and governance

- Identity & Access Management (Active Directory, LDAP, RBAC, etc.)
- Data Privacy & Protection (Encryption, anonymization, secure data sharing, etc.)
- Governance tools (metadata management, data cataloging, etc.)

More to be added.

B. List of Abbreviations

- AD: Active Directory - A widely used proprietary identity and access management system that reinforces security.
- AI: Artificial Intelligence - Intelligence demonstrated by machines, as opposed to natural intelligence displayed by humans.
- ANSD: Agence Nationale de la Statistique et de la Démographie - The National Statistical Office of Senegal.
- API: Application Programming Interface - A set of rules that allows different software applications to communicate with each other.
- AWS: Amazon Web Services - A cloud computing platform provided by Amazon.
- CAD: Collaborative on Administrative Data - A United Nations initiative focused on administrative data use in official statistics.
- CKAN: Comprehensive Knowledge Archive Network - An open-source data portal platform.
- CSV: Comma Separated Values - A simple, text-based format for storing tabular data.
- DA-13: Development Accounts 13 - A UN initiative focusing on data innovation and technology.
- DANE: Departamento Administrativo Nacional de Estadística - The National Statistical Office of Colombia.
- DDI: Data Documentation Initiative - A standard for capturing metadata about research and survey data.
- ELT: Extract, Load, Transform - A data integration process where data is first extracted from sources, loaded into a target system, and then transformed.
- ETL: Extract, Transform, Load - A traditional data integration process where data is first extracted from sources, transformed to fit operational needs, and loaded into a target system.
- FOSS: Free and Open-Source Software - Software that can be used, studied, modified, and distributed by anyone.
- FTP: File Transfer Protocol - A standard network protocol used for transferring files between a client and server.
- GDPR: General Data Protection Regulation - A regulation in EU law on data protection and privacy.
- GIS: Geographic Information System - A system designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data.
- GSO: General Statistics Office - The National Statistical Office of Vietnam.
- HDFS: Hadoop Distributed File System - A distributed file system designed to run on commodity hardware.
- HTTP: Hypertext Transfer Protocol - The foundation of data communication for the World Wide Web.
- INE: Instituto Nacional de Estadística - The National Statistical Office of Uruguay.
- INEGI: Instituto Nacional de Estadística y Geografía - The National Statistical Office of Mexico.
- INS: Institut National de la Statistique - The National Statistical Office of Tunisia.
- JSON: JavaScript Object Notation - A lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.
- k8s: Kubernetes - An open-source platform for automating deployment, scaling, and operations of application containers.

- LDAP: Lightweight Directory Access Protocol - An open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services.
- LOD: Linked Open Data - A method of publishing structured data so that it can be interlinked and become more useful through semantic queries.
- MBS: Maldives Bureau of Statistics - The National Statistical Office of Maldives.
- ML: Machine Learning - A subset of artificial intelligence that provides systems the ability to learn from data and improve from experience.
- MOU: Memorandum of Understanding - A formal agreement between two or more parties.
- NSA: Namibia Statistics Agency - The former name of Namibia's National Statistical Office.
- NSO: National Statistical Office - Government agencies responsible for collecting, processing, and publishing official statistics.
- NSS: National Statistical System - The entire network of institutions and entities involved in the collection, processing, and dissemination of official statistics in a country.
- OAuth2: Open Authorization 2.0 - An industry-standard protocol for authorization.
- ORC: Optimized Row Columnar - A columnar storage format designed for high-performance analytics.
- PBAC: Policy-Based Access Control - A method of managing user access to resources based on policies.
- PDI: Pentaho Data Integration - A data integration tool that enables ETL from a variety of sources.
- PET: Privacy Enhancement Technologies - Technologies that help protect data privacy.
- PxWeb: A tool for disseminating statistical data on the web.
- QGIS: Quantum Geographic Information System - An open-source geographic information system.
- RBAC: Role-Based Access Control - A method of regulating access to computer or network resources based on roles.
- S3: Simple Storage Service - Amazon's object storage service.
- SDMX: Statistical Data and Metadata eXchange - An international initiative to standardize and modernize the exchange of statistical data and metadata.
- SFTP: Secure File Transfer Protocol - A network protocol that provides file access, file transfer, and file management over any reliable data stream.
- SSL/TLS: Secure Sockets Layer/Transport Layer Security - Protocols for establishing authenticated and encrypted links between networked computers.
- SSH: Secure Shell - A cryptographic network protocol for operating network services securely over an unsecured network.
- Stats SL: Statistics Sierra Leone - The National Statistical Office of Sierra Leone.
- UNESCAP: United Nations Economic and Social Commission for Asia and the Pacific - A regional commission that promotes economic and social development in the Asia-Pacific region.
- VPN: Virtual Private Network - A service that encrypts internet connections to protect online privacy.
- XML: eXtensible Markup Language - A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.