Jira task checklist

Task Name	Description	Env	Responsibility
Obtain [SOURCE_NAME] Access & Technical Documentation	**Objective** Get all the access, credentials, and documentation needed so we can safely and reliably ingest data from the [SOURCE_NAME] system. --- **Description** This task covers everything required to connect to the [SOURCE_NAME] system from our environment. It includes setting up user/service access, collecting connection and authentication details, understanding the data structure, and documenting how often data will be updated. The outcome is that we can successfully test a connection from Dev and have all information stored securely and centrally. --- **Scope** • **Access to source system** • Request and obtain a user or service account for the [SOURCE_NAME] system. • Ensure the account has the correct permissions for data access (read-only unless otherwise agreed). • **Connection details (as applicable)** **Collect API details:** • Base URL • Endpoints • Required headers • Query parameters **Collect database details:** • Server/host • Port • Database name • Schema • Any required network info **Collect SFTP details:** • Host • Port • Folder/path • File naming conventions • **Authentication details** Obtain the required auth method and details, e.g.: • OAuth (client ID/secret, token URL, scopes, etc.) • API key(s) • Username and password • Certificates/keys Clarify any token expiry, rotation, or renewal process. • **Network / IP whitelisting** • Collect the list of IP addresses or ranges from which we will connect (Dev at minimum). • Share these with the source owner for whitelisting if required. • **Sample data** • Request sample data files or sample API responses that are representative of real data. • Use samples to validate structure, formats, and edge cases. • **Frequency and SLA** Confirm with the source owner: • Data refresh frequency (e.g. real-time, hourly, daily) • SLA for data availability and expected response times --- **Deliverables** • **Credentials**	Dev Test Prod	Data Engineer

	• All access credentials (accounts, keys, secrets, certificates) are stored securely in Key Vault (or the agreed secure secrets store). • **Documentation** • All technical and data documentation (connection details, auth steps, schema, IPs, refresh frequency, SLA) is stored in the code. • **Source owner confirmation** A confirmation email/message from the [SOURCE_NAME] owner stating that: • Access has been granted • Connection details and expectations (frequency/SLA) are correct --- **Definition of Done** • A test connection from the Dev environment to [SOURCE_NAME] is successful using the stored credentials. • All received documentation is uploaded to the agreed central repository. • All credentials are stored securely in Key Vault.		

[ENV] : Network Whitelisting for [SOURCE_NAME]	**Objective**
	Ensure network connectivity from Dev environment Connects to [SOURCE_NAME].
	Scope
	• Obtain outbound IP of Azure Function / Container App
	• Validate IP with Tanish/Marc before raising request
	• Raise firewall/whitelisting request for:
	  ◦ VDI
	  ◦ Azure Function
	  ◦ Container App
	  ◦ GitHub runners (if required)
	• Confirm port-level access (443/22/etc.)
	Deliverables
	• Approved firewall change request
	• Connectivity test successful
	Definition of Done
	• Connection test from Dev Azure Function successful
		Dev
		Test
		Prod
			Dev Ops

Collect [SOURCE_NAME] Schema and Table Metadata details	**Objective** Identify all source tables, fields, and any dependencies needed for [PROJECT], and document how they will be used for the purpose of [PROJECT PURPOSE]. Note - Cautious while working or check if the source will be obsolete after some, then in that case there might be some rework/effort Identify the complexity/Priority for data load --- **Scope** • **Identify required source tables** • List all source tables needed for [PROJECT]. • Capture each table's structure: schema name, table name. • **Key and identifier details** • Identify primary keys, composite keys, and any important business/CLI identifiers used for joins or lookups. • **Data volume estimation** • Estimate data volume for each required table (e.g. row counts, growth per day/month). • Note any high-volume tables that may impact performance or storage. • **Sensitive data assessment** • Identify columns that contain sensitive or PII (Personally Identifiable Information). • Flag these fields clearly for later masking, encryption, or access control. --- **Deliverables** • **Source Table Inventory Document** • List of all required source tables, with basic details (schema, table name, purpose, dependencies). • **Column-level Metadata** • For each table: column name, data type, key/identifier flags, sensitivity/PII flags, and short description where available. --- **Definition of Done** • All required source tables for [PROJECT] are identified. • Table and column details (including keys and sensitive fields) are documented in the Source Table Inventory and column-level metadata.	Dev	
Create Azure Function – [SOURCE_NAME]	**Objective** Build an Azure Function in the Development environment to extract data from [SOURCE_NAME] for the [USE_CASE]. **Scope** • Create a Python Azure Function to connect to [SOURCE_NAME] and extract the required data. • Use Python 3.11 as the runtime environment for the function. **Deliverables** • The Azure Function is deployed and available in the Dev environment. **Definition of Done** • The Azure Function runs successfully in Dev	Dev	Dev Ops

Implement Data Ingestion from [Source] to Kafka	**Objective** Implement ingestion pipeline to publish data from [SOURCE_NAME] to Kafka For Full Load **Scope** • Implement producer logic in Azure Function • Handle error scenarios **Deliverables** • Data published successfully to Kafka topic and Fabric Application zone • Sample messages validated • Throughput validated **Definition of Done** • Messages visible in Kafka • No data loss during retry • Error handling tested	Dev	Data Engineer
Implement Delta/Incremetal Logic for [Source]	**Objective** Implement logic to ingest only new or updated records from source. **Scope** • Load Incremental data • Validate late-arriving data handling • Backfill support **Deliverables** • Metadata storage created • Successful incremental test run **Definition of Done** • No duplicate ingestion • Only new records processed • Metadata updated after each successful run	Dev	Data Engineer
Setup Github Repo and Create CI /CD Pipeline	**Objective** Set up an automated deployment pipeline for [SOURCE_NAME] so that code can be built, tested and deployed to Dev automatically, and to Prod with manual approval. **Scope** 1. **Create GitHub repository** • Create a new GitHub repository for the [SOURCE_NAME] codebase. **Configure environment variables per environment** • Ensure secrets are stored securely (e.g. GitHub Secrets) and are correctly used by the pipeline. **Deliverables** • A fully working CI/CD pipeline in GitHub Actions for [SOURCE_NAME]. • Automatic deployment to Dev on successful pipeline runs (as defined in the branching strategy). • Manual approval-based deployment to Prod with a clear approval step and responsible approvers defined. **Definition of Done** • Automatically built, tested, and successfully deployed to the environment via the pipeline. • Update the readme.	Dev	Data Engineer

DEV: Setup Monitoring & Data Validation for [Source]	**Objective** Implement monitoring and validation checks for ingestion pipeline. **Scope** • Enable Application Insights • Create ingestion success/failure logs • Implement row count validation • Implement schema validation check • Track ingestion duration • Validate Kafka message count vs source count **Deliverables** • Monitoring dashboard created • Validation queries implemented • Test failure scenario validated **Definition of Done** • Metrics visible in monitoring tool • Validation alerts triggered on failure	Dev Test Prod	Data Engineer

Set Up Alerting and Logging	**Objective**
	
	Implement automated alerting for ingestion pipeline failures and performance degradation.
	
	Scope
	
	• Configure and validate alerts for the ingestion pipeline, including:
	  ◦ Function execution failures
	  ◦ Kafka publish failures
	  ◦ Zero records ingested for a scheduled run
	  ◦ Abnormally long execution time / SLA breach
	• Integration of alerts with email and/or Microsoft Teams channels
	• Definition and configuration of appropriate severity levels (e.g. Critical, High, Medium, Low)
	
	Deliverables
	
	• Alerts tested
	• Alert documentation created
	
	Definition of Done
	
	• Failure simulation triggers alert
	• Alert reaches responsible team
		Dev
		
		Test
		Prod
			Data Engineer

Deploy [Source] data to Production	**Objective**
	
	Deploy the finalized Python code for the [SOURCE_NAME] system to the Production environment, following deployment and security best practices.
	
	--- 	
	
	Description
	
	This task covers preparing, validating, and deploying the final Python code for [SOURCE_NAME] into Production. It ensures that the code is production-ready, uses proper configuration and secrets management, and that deployment is done in a controlled and auditable way.
	
	--- 	
	
	Scope
	
	Setup Pull request and review it.
	
	• **Deployment readiness**
	
	• Confirm that the Python code has passed all required checks in lower environments (Dev / QA / UAT).
	• Ensure all known defects for this release are either resolved or accepted.
	
	• **Configuration & secrets**
	
	• Verify that all Prod configuration (environment variables, connection strings, endpoints) is set correctly.
	• Ensure all secrets (keys, passwords, tokens) are stored in a secure store (e.g. Key Vault, GitHub/Azure DevOps secrets) and not in code.
	
	• **Deployment process**
	
	• Use the approved CI/CD pipeline or standard deployment process to deploy the Python code to Production.
	• Follow the agreed change management process (e.g. change ticket, approvals, CAB if required).
	• Perform a controlled deployment (e.g. scheduled window, blue/green/canary if applicable).
	
	• **Post-deployment validation**
	
	• Run smoke tests or basic functional checks to confirm that the Python code runs correctly in Production.
	• Verify logging and monitoring are working (logs, alerts, dashboards).
	
	• **Documentation & handover**
	
	• Update deployment notes / release documentation with:
	  ◦ Deployed version / commit
	  ◦ Deployment date and time
	  ◦ Any known issues or follow-up items
	• Inform relevant stakeholders that the deployment is complete.
	
	--- 	
	
	Deliverables
	
	• Python code for [SOURCE_NAME] successfully deployed to the Production environment.
	• Updated configuration and secrets for Production stored in the approved secure store.
	• Deployment / release notes documented in the project Confluence/SharePoint or release tracker.
	
	--- 	
	
	Definition of Done
	
	• Deployment to Production completes without errors using the approved process.
	• Smoke tests in Production pass and the application behaves as expected.
	• No secrets are stored in source code or plain text; all are managed via the secure store.
	• Deployment details are documented and communicated to stakeholders.
		Prod	Support Engineer

Implement Security Policy		Dev Test Prod	Data Engineer

Ingest [SOURCE_NAME] Data to [INTERMEDIATE_LAYER_NAME] for Fabric Processing	**Objective** Extract data from [SOURCE_NAME] and store it in the [INTERMEDIATE_LAYER_NAME] (e.g. ADLS Gen2, Blob Storage, File Share). This intermediate layer will be the trusted source for downstream loading into Microsoft Fabric. Direct ingestion into Fabric is not feasible because of: [Network restriction / API limitation / Security constraint / Performance limitation / Architectural decision]. --- **Scope** • Build or configure a process to extract data from [SOURCE_NAME]. • Write the extracted data to the defined intermediate layer ([INTERMEDIATE_LAYER_NAME]). • Ensure the data in the intermediate layer is complete, consistent, and ready for ingestion into Microsoft Fabric. --- **Deliverables** • **Data written to intermediate layer** • Data from [SOURCE_NAME] is successfully stored in [INTERMEDIATE_LAYER_NAME]. • **Folder / path structure created** • Clear and consistent folder or path structure in the intermediate layer (e.g. /source_name/entity/date=YYYY-MM-DD/). • **Incremental load logic implemented** • Logic in place to load only new or changed data (e.g. based on timestamp, watermark, or change flag). • **Metadata updated** • Relevant metadata is captured and maintained (e.g. load time, source system, record counts, watermark value). • **Monitoring configured** Monitoring and logging set up for: • Job status (success/failure) • Data volume checks • Error handling --- **Definition of Done (DoD)** • **Successful test execution in [ENV]** • End-to-end run in the target environment ([ENV], e.g. Dev/UAT/Prod) completes successfully. • **No duplicate files on re-run** • Re-running the job does not create duplicate files or duplicate data in the intermediate layer. • **Watermark updated correctly** • Watermark or equivalent mechanism is updated after each run and used correctly for incremental loads. • **Logs visible in monitoring system** • Execution logs (success, failures, metrics) are visible in the agreed monitoring/logging tool. • **Alert tested successfully** • At least one failure/alert scenario has been tested and notifications are received by the right team. • **Documentation updated** • Technical documentation (process flow, paths, incremental logic, watermark strategy, monitoring) is updated in the central repository (e.g. Confluence/SharePoint). • **Sign-off received from [Team/Owner]** • Formal sign-off from [Team/Owner] confirming the solution meets requirements and is ready for use by downstream consumers.	Dev Test Prod	Data Engineer

Setup [INTERMEDIATE_LAYER_NAME]	**Objective**
	
	Define and set up the [INTERMEDIATE_LAYER_NAME] (e.g. ADLS Gen2, Blob Storage, File Share) for data coming from [SOURCE], so that data can be loaded into Microsoft Fabric in a controlled and reliable way.
	
	Direct ingestion into Fabric is not feasible due to:
	[Network restriction / API limitation / Security constraint / Performance limitation / Architectural decision].
	
	--- 	
	
	Scope of Work
	
	1. **Intermediate Layer Setup**
	
	• **Storage account**
	
	• Validate that the storage account [storage_account_name] exists and is suitable for this use case,
	• Or create a new storage account [storage_account_name] if one does not exist.
	
	• **Container**
	
	• Create or validate the container [container_name] in the storage account to store data from [SOURCE].
	
	• **Folder structure**
	
	• Define and document the folder/path structure for organizing data (for example):
	  ○ /[source]/[entity]/date=YYYY-MM-DD/
	• Ensure the structure supports incremental loads and downstream consumption by Fabric.
	
	• **File naming convention**
	
	• Define and document a standard naming pattern:
	  ○ "[source]*[entity]*[YYYYMMDDHHMMSS].[format]"
	• Clarify how each part (source, entity, timestamp, format) will be populated.
	
	• **Retention policy**
	
	• Define and document retention rules for data stored in the intermediate layer:
	  ○ Raw retention: [X days]
	  ○ Archive retention: [X days]
	
	--- 	
	
	Deliverables
	
	• Documented intermediate layer design (storage account, container, folder structure, naming convention, retention).
	• Storage account [storage_account_name] created/validated for use as the intermediate layer.
	• Container [container_name] created/validated in the storage account.
	• Agreed and documented folder structure and file naming convention.
	• Documented retention policy for raw and archived data.
	
	--- 	
	
	Definition of Done
	
	• The storage account [storage_account_name] and container [container_name] are available and ready to use.
	• Folder structure and file naming convention are clearly documented and approved by the relevant team/owner.
	• Retention policy (raw and archive) is defined, documented, and agreed.
	• The intermediate layer design is stored in the central documentation location and referenced for future ingestion tasks.
		Dev
		
		Test
		Prod
			Data Engineer