June-2024

Crawler Framework Design

Robbins, Logan Jameson

Rojas, Eduardo

## Table of Contents

[Table of Contents 1](#_Toc174085896)

[Revision History 5](#_Toc174085897)

[Crawler overview. 6](#_Toc174085898)

[General Security Guides 7](#_Toc174085899)

[General Best Practices Recommendations 7](#_Toc174085900)

[Technical specifications to Incorporate. 8](#_Toc174085901)

[What Data Can be Crawled 10](#_Toc174085902)

[Crawling Federated Data 10](#_Toc174085903)

[Crawler Framework Scope Definition. 11](#_Toc174085904)

[What is the Crawler. 11](#_Toc174085905)

[What the Crawler is Not. 11](#_Toc174085906)

[Crawler Components. 11](#_Toc174085907)

[Contract. 12](#_Toc174085908)

[Crawlers. 12](#_Toc174085909)

[Sources & Resources. 12](#_Toc174085910)

[Products. 12](#_Toc174085911)

[Crawler Intermediate Storage (CwIS). 12](#_Toc174085912)

[Crawler Framework General View. 13](#_Toc174085913)

[Crawler Framework Logic. 13](#_Toc174085914)

[Outbound call to AirFlow (Contract) 13](#_Toc174085915)

[Interaction Scenarios. 14](#_Toc174085916)

[Unattended 14](#_Toc174085917)

[Attended. 14](#_Toc174085918)

[Types of Sources. 14](#_Toc174085919)

[Sources with Resources. 14](#_Toc174085920)

[Resource Sources (web). 14](#_Toc174085921)

[Landing Process. 14](#_Toc174085922)

[Scheduled Events. 14](#_Toc174085923)

[On-Demand Events (API). 14](#_Toc174085924)

[Sensor-Triggered Events. 14](#_Toc174085925)

[About the Contract. 15](#_Toc174085926)

[Contract Structure. 15](#_Toc174085927)

[Intermediate Storage (CwIS). 16](#_Toc174085928)

[Technical and Functional Specifications 16](#_Toc174085929)

[Crawler Framework Process. 18](#_Toc174085930)

[Attended Model. 19](#_Toc174085931)

[On-Demand Request. 19](#_Toc174085932)

[On-Demand Controller Request Payload. 19](#_Toc174085933)

[Process pseudocode (3): 20](#_Toc174085934)

[Unattended Model. 21](#_Toc174085935)

[Scheduled Mode. 21](#_Toc174085936)

[Scheduled Crawler Controller Request Payload. 21](#_Toc174085937)

[Process pseudocode (3): 22](#_Toc174085938)

[On-Demand Request. 23](#_Toc174085939)

[On-Demand Crawler Controller Request Payload. 23](#_Toc174085940)

[Process pseudocode (3): 24](#_Toc174085941)

[Crawler Framework Implementation. 25](#_Toc174085942)

[Federated Data 25](#_Toc174085943)

[Crawler Implementation. 25](#_Toc174085944)

[Controllers 25](#_Toc174085945)

[Crawler Controller. 25](#_Toc174085946)

[On-Demand Crawler Controller. 25](#_Toc174085947)

[On-Demand Controller. 25](#_Toc174085948)

[AGS Sync Controller. 25](#_Toc174085949)

[Crawlers. 25](#_Toc174085950)

[Azure Secrets Crawler. 25](#_Toc174085951)

[AGS Crawler. 25](#_Toc174085952)

[Cleanup Crawler. 25](#_Toc174085953)

[Monitor Crawler. 25](#_Toc174085954)

[Notification Crawler. 26](#_Toc174085955)

[Invokes Data Pipeline. 26](#_Toc174085956)

[Invokes Airflow. 26](#_Toc174085957)

[Invokes Others. 26](#_Toc174085958)

[Provider Architecture. 27](#_Toc174085959)

[What are Airflow Providers? 27](#_Toc174085960)

[Understanding Operators and Hooks 27](#_Toc174085961)

[Relevance of Operators and Hooks in Providers 27](#_Toc174085962)

[Steps to Create a Custom Provider 27](#_Toc174085963)

[1. Define Your Operators. 27](#_Toc174085964)

[2. Develop Your Hooks. 28](#_Toc174085965)

[3. Package Your Provider. 28](#_Toc174085966)

[DAG Implementation. 29](#_Toc174085967)

[Crawler Framework Actionability. 30](#_Toc174085968)

[Initialization Mechanisms. 30](#_Toc174085969)

[Payload Definition for Apache-Airflow DAGs. 30](#_Toc174085970)

[Attended Crawler. 30](#_Toc174085971)

[Unattended Crawler. 30](#_Toc174085972)

[On-Demand Unattended Crawler. 30](#_Toc174085973)

[Custom Crawler. 30](#_Toc174085974)

[Third-Party Interface. 30](#_Toc174085975)

[Airflow DAGs. 30](#_Toc174085976)

[Other Pipeline Providers. 30](#_Toc174085977)

[Contracts Table Design 31](#_Toc174085978)

[Table Definitions and DDL statements. 31](#_Toc174085979)

[Create the table. 31](#_Toc174085980)

[Create the Contract Sources table. 31](#_Toc174085981)

[Contract Monitor Table Design 31](#_Toc174085982)

[Table Definitions and DDL statements. 31](#_Toc174085983)

[Create the table. 32](#_Toc174085984)

[Crawl Monitor Table Design 32](#_Toc174085985)

[Table Definitions and DDL statements. 32](#_Toc174085986)

[Create the table. 32](#_Toc174085987)

[~~Database Relationship Flow Overview~~ 32](#_Toc174085988)

[Controller Logic Overview 33](#_Toc174085989)

[Contract Controller 33](#_Toc174085990)

[Crawler Controller (On-Demand or Unattended) 33](#_Toc174085991)

[Crawler (each individually) 33](#_Toc174085992)

[Invoke Controller 34](#_Toc174085993)

[Ingestion Monitor Controller 34](#_Toc174085994)

[Ingestion Client (TBD) 34](#_Toc174085995)

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Date | Description |
| 1.0.0.0 | Robbins, Logan Jameson  Rojas, Eduardo | 07.15.2024 | Preliminary review for developer process |
| 1.0.0.1 | Rojas, Eduardo | 07.16.2024 | Controller payload added |
| 1.0.0.2 | Rojas, Eduardo | 07.18.2024 | [General Security Guides](#_General_Security_Guides) |
| 1.0.0.3 | Rojas, Eduardo | 07.19.2024 | Intermediate Storage (CwIS) general guidelines. |
| 1.0.0.4 | Rojas, Eduardo | 07.23.2024 | Design project references updated |
| 1.0.0.5 | Rojas, Eduardo | 07.30.2024 | [Provider Architecture](#_Provider_Architecture.) section added. |
| 1.0.0.6 | Rojas, Eduardo | 08.06.2024 | Document Merged |
| 1.0.0.7 | Rojas, Eduardo | 08.08.2024 | Ingestion Pipeline integration and Contract ‘[custom\_params](#_Contract_Structure.)‘. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Crawler overview.

The Crawler Framework is the preferred tool for managing the interaction between Sources or Resources and the client's preferred data pipeline. Comprising both proprietary providers (e.g., [Apache-Airflow providers](https://github.com/apache/airflow/tree/main/airflow/providers)) and custom providers (e.g., [Intel Custom Providers](https://github.com/intel-innersource/firmware.boot.uefi.iafw.devops.infrastructure.airflowETL/tree/dev-migration/airflow/providers/intel): see them as examples), it facilitates the accessibility of business resources within the "Ingestion Data Pipeline" to "Crawler Intermediate Storages (CwIS)," based on a contract that defines the interaction parameters.

This movement of resources does not include data extraction or transformation into a canonical model, such as JSON. However, it does involve extracting a series of desired variables, classifiers, and other essential data for producing the metadata associated with the resource.

The contract consists of a set of parameters that provide the Crawler with the necessary behavioral inputs, such as execution frequency, sources, resources, access rights, entitlements, required endpoints, and others that will be detailed in the design.

## General Security Guides

In the realm of information security, ensuring data privacy, user permission traceability, and intellectual property protection are paramount.

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)][crawler-security tab] |

Our Crawler Framework's main objective is to carry out the movement, coordination, and monitoring of these data movements, starting from the original repository (Source) to the intermediate repository (CwIS) and maintaining the traceability of the associated permissions through meta-data using the custom schema in JSON.

### General Best Practices Recommendations

* **JSON Schema Validation.** Implement rigorous schema validation to ensure the integrity and consistency of the JSON data structure. The use Pydantic ([JSON Schema](https://docs.pydantic.dev/latest/concepts/json_schema/)) for schemas and payload implementation could facilitate our work.
* **Permission Mapping.** Establish a comprehensive mapping strategy to accurately translate [AGS entitlements](https://ags.intel.com/) from the original repository to the new one.
* **Data Encryption**: Encrypt sensitive data during transit and at rest to protect against unauthorized access.
* **Minimal Privilege Principle.** Ensure that access to data in transit or at rest by users is not guaranteed and access through applications and services is limited to reading only and only to required applications.
* **Domain Data Access.** Ensure that data in transit and at rest is only accessible by proprietary applications and services. An Ingestion Pipeline 'A' should not be allowed to access data in transit from Ingestion Pipeline 'B'.
* **Access Control.** Restrict access to data in transit and at rest, particularly to the data that makes up the meta-data in JSON schema.
* **Audit Logging.** Implement robust audit logging to capture all changes and access events, ensuring full traceability. Do not expose sensitive data or key values; Consider feeding logs with values; that facilitate the reconstruction of facts, support and conflict resolution.
* **Data Integrity Checks.** Use checksums or hashes to verify data integrity during and after the migration process. Consider that data at rest may remain in this state for longer than the pre-established time.
* **Version Control.** Use version control for JSON schema definitions to track changes and updates over time.
* **Compliance and Regulatory Standards.** Ensure the migration process complies with relevant data protection regulations and standards. Refer to '[Information Security Policies](https://circuit.intel.com/content/corp/infosec/policies.html)' for more information.
* **User Training.** Final access to information depends on the accuracy, correctness and integrity of sensitive data, such as: AGS entitlements; powered by the Crawler Framework. Ensure you implement required security protocols to prevent misuse and improve security awareness.

### Technical specifications to Incorporate.

* **Primary Entitlements**. The retrieval and utilization of entitlements associated with the Source or Resource must always be achieved as part of the Crawler process from the Resource itself.

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)][crawler-security tab] |

* **Synthetics Entitlements**. In the absence of entitlements associated with the Resource or the inability to retrieve them from the Source or Resource, the entitlements established in the contract are mandatory.

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)][crawler-security tab] |

* **Entitlement Priority**. Primary entitlements will be set by default. In their absence, Resource-level entitlements will take precedence over contract-level entitlements. In the absence of resource level entitlements, contract level entitlements will be used for reading rights.
* **Access to Sources and Resources**. Both Controllers and Crawlers will access Sources and Resources using alternative authentication mechanisms according to the Source. Azure Secrets will be used as a repository for the required parameters, such as connectors, tokens, usernames, and passwords as appropriate and according to the PAM policy if applicable..

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)][crawler-security tab] |

* **Proper Data Handling**. As indicated in the "General Best Practices Recommendations" the solution will implement additional mechanisms for protecting sensitive data, such as entitlements and parameters of the sources and services.
* **Crawler Intermediate Sources (CwIS)**. Access to CwIS is privileged only for the services that strictly require it. Controllers, Crawlers, and the respective Ingestion Pipeline are the only entities with access. The separation of information domains will ensure that the information landed in CwIS is securely accessible only by the respective Ingestion Pipeline and the components of the Crawler Framework.

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)][crawler-security tab] |

* **Resource Correlation**. At all times, it must be certain that the group of entitlements associated with a Resource match those retrieved from the Source or Resource[[1]](#footnote-2). These entitlements will remain unaltered and will follow the "General Best Practices Recommendations" described earlier.
* **Entitlement Refresh**. The Crawler Framework must provide mechanisms for alternate synchronization of entitlements from the Sources or Resources at regular intervals (approximating real-time) to ensure that the final data repository (e.g., chunks in the vector database) reflects changes from the source or contracts when there are no "Primary Entitlements."

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)][crawler-security tab] |

## What Data Can be Crawled

There are two types of data: Federated Data and User Data.

**Federated Data** is any data that:

1. Is accessible by more than one person
2. Relies on data governance/infosec framework for access

Examples include Wikis, SharePoint, NFS files with access to more than one person, etc.

When a user submits a Crawler request for a resource labeled as Federated, it will be ingested according to the active Federated Ingestion Pipeline and stored in the Federated Data Layer.

**User Data** is any data that:

1. Is meant for usage only by the user

Examples include Emails, Excel Spreadsheets, public PDFs, HTTP, etc.

### Crawling Federated Data

Federated Data requires approval of the Data Owner before it can be ingested for use in Forge. To accomplish this, a System Account should be provisioned, and the User should submit an access request for that System Account. If the Data Owner approves, this data will then be accessible to ALL USERS who have access to read it via the Entitlement System (AGS, AD, etc.). When the user submits the request to Crawl this data, they will pass the System Account ID in the contract and store the password in accordance with PAM policy for access by the Airflow server processing the request.

The Crawler Controller will be responsible for determining if the Resource is Federated. If so, an attribute flag is\_federated will be added to the contract for downstream processing.

Scenarios:

* If the data requested to be crawled is open to "All Blue Badge", it can be crawled
* If the data requested to be crawled is open to "All Green Badge", it can be crawled
* If the data is owned by the user submitting the request, it can be crawled
* If the data requested to be crawled is NOT owned by the user submitting the crawl, it will be denied unless the valid System Account ID is found in the request

Note: The Crawler Controller is responsible for validating that the User is the owner of the System Account.

## Crawler Framework Scope Definition.

### What is the Crawler.

The scope of the Crawler includes a DAG to move Resources (or sub-resources from Source) to a common location (CwIS[[2]](#footnote-3)). Along with the Resource itself, an accompanying metadata file will include information related to downstream processing [see: [crawlereddoc](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/Ingestion/crawler_schema.json)]. In addition to the movement of data, security protocols for storage within and access to the CwIS by downstream apps will be implemented.

### What the Crawler is Not.

The scope of the Crawler does not include accessing any of the data within the Source, extraction of any data from the Source (beyond metadata).

### Crawler Components.

|  |
| --- |
|  |
| Visio reference: [[GAI Arch In Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Enterprise%20Platform/GAI%20Arch%20In%20Process.vsdx?d=wcf3ab909411b4be7b4213e309e1478bc&csf=1&web=1&e=b2laPi)] |

#### Contract.

Defines all required parameters and instructions needed for the Crawler to access and transfer Resources from Sources to the Crawler Intermediate Storage (CwIS) area.

##### Crawlers.

Airflow DAGs with the following responsibilities:

* Execute the transfer of Resources from Sources to the CwIS.
* Create the metadata file that accompanies the Resource.
* Provide instructions and scheduling information for invoking the ingestion DAG
* Notify the Controller that the Crawler process has completed.

##### Sources & Resources.

* SharePoint.
* Azure.
* DB.
* Wiki.
* Web Page.

##### Products.

The result of the Crawler are two Products in the CwIS.

* Resource

The data artifact in file format

* resource\_contract.resource.metadata: [[applications.aut omation.gai.design](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/crawler_resource_contract.json)].

A file that accompanies.

##### Crawler Intermediate Storage (CwIS).

Azure Blob or Minio (as available). Data encrypted at rest. AGS persisted via metadata at the Resource level.

## Crawler Framework General View.

Intro: What is the Landing Zone (Capabilities and Limitations).

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)] |

### Crawler Framework Logic.

1. The Job processing service will listen to messages on the event bus.
2. It will retrieve the exact dag\_id to be invoked (stored in the config of the service)
3. Once the message is received the service It will invoke the Airflow endpoint [Discuss: Need actual airflow endpoint from team]
4. Once the Airflow endpoint is invoked successfully it will update the job request table with the status “Queued” and update the job\_request table with the airflow\_dag\_id. and it will remove the inbound message from the event bus.
5. If the airflow call fails, the processing service will update the retry count in the job\_request table.
6. The processing service should retry for the specific number of times. If the retry count is exceeded, then it will remove the message from the bus and update the status of the job\_request to “failed”.
7. In the case of failure (any status other than 200 or 201), Capture the value of the “detail” from the error response (Refer Response Payload(Error)) to the “failure\_reason” column in the job\_request table

#### Outbound call to AirFlow (Contract)

|  |  |  |  |
| --- | --- | --- | --- |
| Paramter | Type | Required | Description |
| **crawl\_id** | Unique identifier. GUID structure |  | Identifier. |
| **folder** | String |  | Location of the file in Azure storage |
| **source** | String |  | Value should be azure |
| **container** | String |  | Azure container name |
| **dag\_run\_id** | String | Yes | Identifier for the DAG that needs to be run |
| **contract\_id** | String |  |  |

### Interaction Scenarios.

#### Unattended

Asynchronous

#### Attended.

Synchronous

### Types of Sources.

#### Sources with Resources.

Sources are the origins of the Resources (SharePoint, Azure, Databases, etc.)

Resources are the data artifacts (documents or other) retrieved from the Source.

For Wiki, the Wiki Site will be a Source and the Wiki Pages will be individual Resources. (Consider that Wiki Pages may have special entitlements, references, etc.).

#### Resource Sources (web).

In some cases, like a web page, a Source may also be the Resource.

### Landing Process.

#### Scheduled Events.

#### On-Demand Events (API).

#### Sensor-Triggered Events.

### About the Contract.

The Contract is a YAML file. One Contract can contain instructions for many Crawls, each of which can invoke Custom Crawlers DAGs, or the default provided by Airflow. An example is below:

#### Contract Structure.

|  |  |  |
| --- | --- | --- |
| Payload [[**Contract Schema**](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/contract_schema.yaml)] | Attribute | Description |
| contract:  contract\_id: "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA"  app\_id: "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA"  business\_id: "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA"  business\_unit: "Null"  data\_classification\_level : "Null"  personal\_data : "Null"  entitlements: []  contacts: []  schedule: "Null"  crawls:  - crawl:  crawl\_id: : "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA"  data\_classification\_level : "Null"  personal\_data : "Null"  entitlements: []  contacts: []  schedule: "0 0 \* \* 1-5"  source\_type: "sharepoint"  source: "Null"  resource: "Null"  credentials:  sharepoint: {}  azure\_blob: {}  azure\_bucket: {}  file\_system: {}  cwiss:  azure\_blob: {}  minio: {}  custom\_crawlers: []  ingestion:  airflow:  custom\_params: {}  credential: {}  monitor\_type: ""  articula8:  unstructuredio: | contract\_id | Unique GUID structure.  Contract Identifier. |
| app\_id | Unique GUID structure.  Attended mode requester id. |
| business\_id | Unique GUID structure.  Unattended mode requester id. |
| business\_unit | Unique GUID structure.  Unattended mode requester name. |
| data\_classification\_level | See: [Intel Data Classifications](https://circuit.intel.com/content/corp/infosec/policies/data-classifications.html). |
| personal\_data | See: [Intel Data Classifications](https://circuit.intel.com/content/corp/infosec/policies/data-classifications.html).  Values: no personal data, sensitive personal data, non-sensitive personal data. |
| entitlements | (**optional**) AGSs list related to all Resource. (see: [Security Guides](#_General_Security_Guides))  Can overridden by the Resource. |
| contacts | Contact email list for notifications. |
| crawls | List of ‘crawl’ |
| crawl\_id |  |
| data\_classification\_level |  |
| personal\_data |  |
| entitlements |  |
| contacts |  |
| schedule |  |
| source\_type |  |
| source |  |
| resource |  |
| credentials |  |
| cwiss |  |
| custom\_crawlers | Custom crawlers required by the Crawler pipeline. |
| ingestion | Ingestion definition structure required for Invoke Controller with connection parameters and custom ones. |
| Airflow |  |
| monitor\_type | Integration type for monitoring. Matches one of the following options: not\_monitor, ingestion\_cliente or call\_back |
| custom\_params | Set of specific parameters from ingestion pipeline that must be sent “as is”, without modification to ingestion. |
| Articula8 |  |
| Unstructuredio |  |
| Note. You can review the structure of the '**crawl**' in [[crawler\_request.yml](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/crawler_request.yml)]. | | |

Monitoring will be conducted at the resource level. This means if more than 1 file matches your resource path, one of the specified Crawler DAGs will be invoked for each file.

### Intermediate Storage (CwIS).

Intermediate storage solutions, such as Amazon S3, Google Cloud Storage, Azure Blob Storage, Redis, Apache Kafka, and RabbitMQ, play a crucial role in unstructured data ingestion pipelines. These tools act as temporary repositories that store data before it is processed or moved to its destination. They help in decoupling various stages of data processing, allowing for more flexible, scalable, and resilient data workflows. Utilizing intermediate storage solutions offers significant advantages, including improved data protection, reduced service latency, and enhanced communication through API protocols, supporting both synchronous and asynchronous operations.

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)] |

#### Technical and Functional Specifications

* **Data Protection**. Implement comprehensive data protection mechanisms including end-to-end encryption for both data at rest and in transit, employing industry-standard protocols such as AES-256 and TLS. Utilize role-based access control (RBAC) or AGS Entitlements to ensure only authorized access and enforce data integrity checks through checksums and hashing algorithms to maintain the accuracy and reliability of the stored data.
* **Latency**. Optimize for low latency by selecting storage solutions that support in-memory or SSD-backed options. Implement caching mechanisms to expedite access to frequently requested data and use geo-distributed storage to reduce latency by storing data closer to the users. These strategies will enhance the responsiveness and performance of the data pipeline.
* **API Communication**. Ensure robust and flexible API communication by utilizing RESTful APIs supported by comprehensive SDKs for various programming languages. Implement API rate limiting and throttling to maintain fair usage and prevent abuse, ensuring the APIs remain reliable and scalable under varying loads. Apigee will be mandatory for productions environments.
* **Support for Synchronous and Asynchronous Communication**. Consider utilize **message queuing systems** like RabbitMQ or Apache Kafka for efficient asynchronous communication and webhooks for real-time synchronous notifications. Design the architecture to support event-driven interactions, allowing services to respond dynamically to events and enhancing the overall system flexibility and responsiveness.
* **Interoperability and Low Complexity**. Foster interoperability by using standardized API interfaces and ensuring compatibility with common data formats like JSON, XML and YAML. Provide robust migration tools to facilitate seamless data transfer between different intermediate storage solutions, minimizing manual intervention and reducing operational complexity.
* **Low Coupling, High Cohesion**. Design the system with a modular architecture that promotes low coupling and high cohesion. Isolate services to minimize dependencies, and define clear, well-documented interfaces for each module.
* **High Degree of Parameterization**. Enable high parameterization by using transactional database storages or environment variables to manage settings, reducing the need for code changes or extended SLAs. Implement dynamic configuration management tools to allow for real-time parameter adjustments without disrupting services and leverage automated deployment tools to ensure consistent parameter management across different environments.

Intermediate storage solution will ensure data protection, minimize latency, support flexible communication methods, facilitate easy interchangeability, and promote a robust, maintainable architecture aligned with best practices.

## Crawler Framework Process.

The crawling process is made up of 4 main pillars.

* **Common functionality**: includes all Crawler Framework elements common for general functionality, such as: parameterization, contracts, customized crawler by source or resource; monitoring, control, notifications, and others.
* **Attended Mode:** includes on-demand functionality, to assist business applications that require real-time data pipeline service, with or without prior configuration of parameters (established in the contract).
* **Unattended Mode**: includes crawling functionality based on schedule triggers, or on-demand. Usually for bulk loads process and provide on-demand processing for off-schedule loads.
* **Ingestion modalities**: crawler framework has been designed considering the Apache-Airflow platform using DAGs as the main data pipeline ingestion. We also value connections to third-party tools, such as: Articula8\*, Unstructured IO Orchestrator, and other data pipeline orchestrators (see: [Invokes Crawler](#_Invokes_Data_Pipeline.)).

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)] |

### Attended Model.

#### On-Demand Request.

It corresponds to the Crawler Framework process responsible for answering user requests made by applications for data loads in real time (REST\*[[3]](#footnote-4)).

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)] |

The scenario begins when the user places its Resources into a Source (1) and requests the 'Crawler Framework Service' (2) to resolve the request through the service interface: Attended On-Deman Controller (3).

##### On-Demand Controller Request Payload.

|  |  |  |
| --- | --- | --- |
| Payload [[**AttendedOnDemandRequest.json**](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/design_schemas/AttendedOnDemandRequest.json)] | Attribute | Description |
| {  "conf": {  "correlation\_id": "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA",  "wwid": "00000000",  "timestamp": "yyyy‑MM‑dd HH:mm:ss. SSSSSS",  "contract": {  "app\_id": "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA",  "data\_classification\_level": "Intel Confidential (IC)",  "personal\_data": "Personal Data",  "entitlements": ["AGS-1”, “AGS-2"],  "contacts": ["email1@intel.com","email2@intel.com"],  "crawls": [  {"crawl": {"crawl\_id": "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA"}},  {"crawl": {"crawl\_id": "BBBBBBBB-BBBB-BBBB-BBBB-BBBBBBBBBBBBBB"}}  ]  }  },  "dag\_run\_id": "\_\_unique\_value\_\_"  } | correlation\_id | Unique GUID structure.  Identifier the client request. |
| wwid | (**optional**) User identification. |
| timestamp | Request timestamp. |
| app\_id | Unique GUID structure. |
| data\_classification\_level | See: [Intel Data Classifications](https://circuit.intel.com/content/corp/infosec/policies/data-classifications.html). |
| personal\_data | See: [Intel Data Classifications](https://circuit.intel.com/content/corp/infosec/policies/data-classifications.html).  Values: no personal data, sensitive personal data, non-sensitive personal data. |
| entitlements | (**optional**) AGSs list related to all Resource.  Can overridden by the Resource. |
| contacts | Contact email list for notifications. |
| crawls | Source set for crawling. |
| dag\_run\_id | Alphanumeric value.  Defines the ‘run\_id’ for this DAG execution. MUST be unique for Airflow environment. |
| Note. You can review the structure of the '**crawl**' in [[crawler\_request.yml](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/crawler_request.yml)]. | | |

#### Process pseudocode (3):

|  |
| --- |
|  |
| See full pseudocode at: [AttendedOnDemandController.py](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/design_schemas/AttendedOnDemandController.py).– |

### Unattended Model.

#### Scheduled Mode.

The Schedule Unattended mode corresponds to the Crawler Framework process required for previously agreed crawling Contracts. Usually known as bash load processes.

|  |
| --- |
|  |
| Visio reference: [[Crawler-framework-Process](https://intel.sharepoint.com/:u:/r/sites/gaiplatformteam/Shared%20Documents/General/Airflow/Crawler%20Framework/Crawler-framework-Process.vsdx?d=w05dcf460e64344419f71867a564bf68a&csf=1&web=1&e=sjEb0D)] |

The scenario begins when the pre-programmed agenda in the Contract (2c) associated with a Source or Resource raises an instance of the controller, the trigger then initiates an instance of the 'On-Demand Crawler Controller' (3c) exclusive for the Contract.

##### Scheduled Crawler Controller Request Payload.

|  |  |  |
| --- | --- | --- |
| Payload [**[UnattendedCrawlerRequest.json](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/design_schemas/UnattendedCrawlerRequest.json)**] | Attribute | Description |
| {  "conf": {  "contract\_id": "AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA",  },  "dag\_run\_id": "\_\_unique\_value\_\_"  } | contract \_id | Unique GUID structure.  Identifier the client’s Contract. |
| dag\_run\_id | Alphanumeric value.  Defines the ‘run\_id’ for this DAG execution. MUST be unique for Airflow environment. |
| Note. You can review the structure of the '**crawl**' in [[crawler\_request.yml](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/crawler_request.yml)]. | | |

#### Process pseudocode (3):

|  |
| --- |
|  |
| See design at: [UnattendedCrawlerController.py](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/design_schemas/UnattendedCrawlerController.py). |

#### On-Demand Request.

The Unattended On-Demand mode corresponds to the Crawler Framework process required for previously agreed crawling Contracts that in particular scenarios require off-schedule loading.

|  |
| --- |
|  |

The scenario begins when a request is made to execute in the controller upon demand of a Contract (2b) associated with a Source or Resource. The execution is carried out only once for each request and is exclusive for a pre-existing Contract (3b).

##### On-Demand Crawler Controller Request Payload.

|  |  |  |
| --- | --- | --- |
| Payload [**[UnattendedOnDemandRequest.json](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/design_schemas/UnattendedOnDemandRequest.json)**] | Attribute | Description |
| **{**  **"conf": {**  **"contract\_id": "**AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAAA**",**  **},**  **"dag\_run\_id": "**\_\_unique\_value\_\_**"**  **}** | contract \_id | Unique GUID structure.  Identifier the client’s Contract. |
| dag\_run\_id | Alphanumeric value.  Defines the ‘run\_id’ for this DAG execution. MUST be unique for Airflow environment. |
| Note. You can review the structure of the '**crawl**' in [[crawler\_request.yml](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/crawler_request.yml)]. | | |

#### Process pseudocode (3):

|  |
| --- |
|  |
| See design at: [UnattendedOnDemandCrawlerController.py](https://github.com/intel-innersource/applications.automation.gai.design/blob/main/Data/crawler/design_schemas/UnattendedOnDemandCrawlerController.py). |

## Crawler Framework Implementation.

### Federated Data

When the Crawler Controller determines data to be Federated, configuration submitted by the user for the specific Resource is ignored. At the end of the Crawl, the default Federated Ingestion Pipeline is invoked.

### Crawler Implementation.

#### Controllers

##### Crawler Controller.

##### On-Demand Crawler Controller.

##### On-Demand Controller.

##### AGS Sync Controller.

#### Crawlers.

##### Azure Secrets Crawler.

Text: Early Confirmation.

##### AGS Crawler.

Text: Early Confirmation.

##### Cleanup Crawler.

##### Monitor Crawler.

|  |
| --- |
|  |

- Monitoring Management.

- Data Model.

- State Tracking.

- Failure Notifications to Support.

- Proactive Client Notifications.

##### Notification Crawler.

#### Invokes Data Pipeline.

##### Invokes Airflow.

##### Invokes Others.

### Provider Architecture.

Apache Airflow is a powerful workflow orchestration tool that allows developers to automate and manage complex data pipelines. To enhance its extensibility, Airflow provides a mechanism for integrating with various external systems through Providers. Understanding how to create a custom provider can significantly enhance your ability to tailor Airflow to your specific needs. This introduction will guide you through the basics of creating custom providers and illustrate the relevance of operators and hooks as essential components within these providers.

#### What are Airflow Providers?

Are modular packages that extend Airflow's functionality by enabling it to interact with external systems, services, and tools. Each provider package bundles a set of operators, hooks, sensors, and other components required to interface with a particular system. By creating a custom provider, you can encapsulate all the integrations needed for your specific workflow requirements, making your DAGs cleaner and more maintainable.

#### Understanding Operators and Hooks

Before diving into the creation of custom providers, it is crucial to understand two foundational concepts: Operators and Hooks.

* **Operators**. Define the tasks in your DAGs. They encapsulate the logic for performing a particular action, such as running a SQL query, transferring data between systems, or executing a Python function. Operators are the building blocks of your workflows and determine the specific actions that Airflow will perform.
* **Hooks**. Are low-level interfaces that manage connections and interactions with external systems. They abstract the details of these interactions, making it easier for operators to perform their tasks without dealing with connection management and API specifics. Hooks provide reusable methods for connecting to databases, cloud services, and other external resources.

#### Relevance of Operators and Hooks in Providers

When creating a custom provider, operators and hooks are the primary components you will develop and package together. Here’s why they are essential:

* **Encapsulation of Logic**. Operators encapsulate the logic for specific tasks, while hooks handle the connectivity and interaction with external systems. This separation of concerns makes your code cleaner and more modular.
* **Reusability**. By creating custom hooks, you can reuse the connection logic across multiple operators, promoting code reuse and reducing duplication.
* **Modularity**. Providers group related operators and hooks into a single package, making it easier to distribute and manage integrations with external systems.

#### Steps to Create a Custom Provider

Creating a custom provider involves the following steps:

##### 1. Define Your Operators.

* Identify the tasks your DAGs need to perform.
* Create custom operators that encapsulate the task logic.

##### 2. Develop Your Hooks.

* Identify the external systems your operators need to interact with.
* Create custom hooks to manage the connections and interactions with these systems.

##### 3. Package Your Provider.

* Organize your operators and hooks into a provider package.
* Ensure your package follows the structure and conventions required by Airflow providers.

|  |
| --- |
|  |

### DAG Implementation.

e.g., Wiki DAG.

## Crawler Framework Actionability.

|  |
| --- |
|  |

### Initialization Mechanisms.

#### Payload Definition for Apache-Airflow DAGs.

#### Attended Crawler.

#### Unattended Crawler.

#### On-Demand Unattended Crawler.

### Custom Crawler.

Text: Wiki Example.

### Third-Party Interface.

#### Airflow DAGs.

#### Other Pipeline Providers.

## Contracts Table Design

The Contracts table holds information about a Contract, anchored by the contract\_id. Certain fields (valuable for metadata or reporting) will be used as columns, so long as they are not impacted by changes in the actual Contract design. The Contract itself, due to its fluid nature, will be stored as JSONB.

### Table Definitions and DDL statements.

Here is the SQL code to create these tables:

#### Create the table.

|  |  |  |
| --- | --- | --- |
| Contracts | Attribute | Description |
| -- Create the contracts table  CREATE TABLE contracts (  contract\_id SERIAL PRIMARY KEY,  mode VARCHAR(50) NOT NULL CHECK (mode IN ('attended', 'unattended')),  last\_run TIMESTAMP,  caller\_type VARCHAR(50) NOT NULL CHECK (caller\_type IN ('user', 'app')),  caller\_id VARCHAR(255) NOT NULL,  contract JSONB -- Using JSONB for better performance with JSON data  ); | contract\_id | Unique identifier for the contract |
| mode | Mode of the contract (attended, unattended), federated |
| last\_run | Timestamp of the last run |
| caller\_type | Type of the caller (user, app) |
| caller\_id | Identifier for the caller |
| contract | Contract (payload) details stored as JSON or YAML |

#### Create the Contract Sources table.

|  |  |  |
| --- | --- | --- |
| Contract Sources | Attribute | Description |
| **-- Create the contract\_source table**  CREATE TABLE **contract\_sources (**  **contract\_id** INT NOT NULL**,**  **source** VARCHAR**(50) NOT NULL CHECK (source IN ('http', 'wiki', 'sharepoint')),**  FOREIGN KEY **(contract\_id)** REFERENCES **contracts(contract\_id)**  **);** | contract\_id | Identifier linking to the contracts table |
| source | Lower case normalized http, wiki, sharepoint, fileshare, azure\_blob |

\*\* Note, this table is for normalization purposes, so that we don’t need to unpack the JSON to report on Sources Per Contract. It’s something that can also be added/backfilled at a later date.

## Contract Monitor Table Design

This table manages the state of the Contract at a master level during it’s Run.

It is different from the Contacts table which stores information about the Contract itself.

There are multiple states a Contract Run can be in:

* Pending (when the Crawler Controller starts triggering the Crawls)
* Crawling (when all Crawls are triggered successfully)
* Complete (after all Crawls complete \_from the Crawl Monitor Table)
* Partial Complete (some Crawls failed \_from the Crawl Monitor Table)
* Fail (all Crawls failed)

When a Contract Run starts, the Crawler Controller inserts a record into this table.

* The first step of the Crawler Controller is to insert a record to this table

### Table Definitions and DDL statements.

Here is the SQL code to create these tables:

#### Create the table.

|  |  |  |
| --- | --- | --- |
| Contract Monitor | Attribute | Description |
| -- Create the Contract Monitor table  CREATE TABLE contract\_monitor (  run\_id varchar,  contract\_id VARCHAR(50),  start\_time TIMESTAMP,  end\_time TIMESTAMP,  num\_crawls INT,  num\_complete INT  ); | run\_id | Unique ID sequential, DAG run ID |
| contract\_id | ID of the contract from contract table |
| start\_time | Timestamp of the start |
| end\_time | Timestamp of the end |
| num\_crawls | 17 |
|  | num\_complete | 16 |

## Crawl Monitor Table Design

This is the “Bridge” between Crawler Phase and the Ingestion Phase of “Forge Pipelines”.

The Crawl Monitor Table holds information about the individual Resource-level Crawls extracted from the Contract. Similar to the Contract Database, some metadata and reporting fields will be created as Columns, while the body of the Crawl (referred to as the config) will be stored as JSONB. Its most important function is to track the state of a Crawl DAG run.

* The first step of a Crawler is to insert a record to this table
* The last step of a Crawler is to update the record in this table

### Table Definitions and DDL statements.

Here is the SQL code to create these tables:

#### Create the table.

|  |  |  |
| --- | --- | --- |
| Crawl Monitor | Attribute | Description |
| -- Create the Crawl Monitor table  CREATE TABLE crawl\_monitor (  crawl\_id UNIQUE PRIMARY KEY,  run\_id VARCHAR  contract\_id VARCHAR(50),  start\_time TIMESTAMP,  end\_time TIMESTAMP,  state VARCHAR(50),  message TEXT,  entitlements JSONB,  Config JSONB  ); | crawl\_id | Unique identifier for the crawl |
| contract\_id | The ID this crawl originated from |
| start\_time | Timestamp of the start |
| end\_time | Timestamp of the end |
| state | Processing, Complete, Error |
| message | Message related to the run |
| entitlements | JSON dictionary |
| config | (payload) details stored as JSON |

## ~~Database Relationship Flow Overview~~

* ~~Crawler Controller explodes payload into List of Crawls (Resource-level, one row/data artifact)~~
* ~~Crawler creates this record in Pending state~~
* ~~Each record will have Entitlements associated~~
* ~~Config: resource name, parameters as inherited from contract, Ingestion configuration~~

|  |
| --- |
|  |

## Controller Logic Overview

Contract Controller -> Receives/Validates/Store contract payload  
Table: contracts

* Creates Record in contract table
* Invokes the Crawler Controller immediately, or schedules crawler for run
* IF run immediately, pass contract\_id to Crawler Controller

Crawler Controller (On-Demand or Unattended) -> Prepares a Contract for run  
Table: contract\_monitor

* Retrieves the Contract from the Contract Table
* Loops through <crawl> resources, merges inherited fields to create List
  + Check that user owns system\_id passed with the contract
* Check each <crawl> resource to see if it’s classified as Federated Data
  + Use provided credentials to retrieve Entitlements (wiki API, AGS, AzureAD)
    - If data is accessible by more than the system\_id and user\_id -> Federated Data
  + IF Federated AND Resource **has not** changed since last Crawl
    - Add flag to <crawl> json “is\_federated” : true, “skip”: true
  + IF Federated AND Resource has changed since last Crawl
    - Add flag to <crawl> json “is\_federated” : true, “skip”: false
    - Override <crawl><ingestion> to default\_federated config
* Create Record in contract\_monitor Table
  + run\_id, contract\_id, ... num\_crawls=len(List), num\_crawls\_complete = 0
* Invokes the Crawler DAGs for each Crawl
  + Pass run\_id, contract\_id

Crawler (each individually) -> Moves files from one location to another  
Table: crawl\_monitor, contract\_monitor

* Insert record in crawl\_monitor
  + crawl\_id (unique), run\_id, contract\_id, state=”Processing”, entitlements=JSONB
* Executes the transfer
* Update record in crawl\_monitor
  + state=”Complete” or state=”Error” & message=”...”
* Update record in contract\_monitor num\_crawls\_complete++

[### CRAWLER SCOPE IS OVER. CRAWL MONITOR TABLE IS THE BRIDGE TO INGEST ###]

Invoke Controller -> Starts Ingestion DAGs  
Table: ingestion\_monitor

* Sensor for new Records from crawl\_monitor in Complete state
  + Location of file in CwIS/blob from <config> in crawl\_monitor
  + <crawl><ingestion> part config retrieved from crawl\_monitor
  + <crawl><entitlements> part of config retrieved from crawl\_monitor
* Creates a record in ingestion\_monitor
  + ingestion\_id (unique), file\_path, .... state=”Started”
* Triggers Ingestion DAG or 3rd-Party tool for Ingestion pipeline

Ingestion Monitor Controller -> API that receive updates from Ingestion DAG

Table: ingestion\_monitor

* Receives update from Ingestion DAG that were called directly between steps
* Updates Ingestion Monitor Table
* IF state=”Complete”, REMOVE file from the CwIS??

Note: It’s possible that third-party ingestion tool has no ability to send us updates, in this case, there is nothing we can do. In this case state=”External”

####

Ingestion Client (TBD) -> Receives requests for status updates

* + Application installs this Client so that we can ping it for updates on Ingestion
  + User specifies the endpoint for us to call
  + Updates Ingestion Monitor Table

1. e.g. The READ access entitlements of the chunks in the vector database should match the READ and WRITE access entitlements for its Resource in the source. [↑](#footnote-ref-2)
2. CwIS. Crawler Intermedia Sources. [↑](#footnote-ref-3)
3. Restricted to 20 Resources per Source per request or less than 10 mega-bytes. [↑](#footnote-ref-4)