**Important Links:**

<https://github.com/ellucianEthos>

<https://github.com/ellucianEthos/python-pubsub-demo>

Experience Architecture: https://resources.elluciancloud.com/bundle/ellucian\_experience/page/c\_experience\_architecture.html

**Lesson 1: Ethos Overview**

**Ellucian Ethos** is a suite of products that enable institutions to manage their integrations to unifies data from across their ecosystem in support of dispersing information across their institution.

* The **Ethos Integration (EI)** allows applications to exchange data using the Ethos Data Model standards format with standard programming models and standard integration patterns.
* **The Ethos Data Model (EEDM)** is built on a common language, a single source of truth, an open platform, and extensibility.
  + **Ellucian Extend** supports the extensibility capabilities of Ellucian Ethos and allows institutions and partners to make extensions to Ellucian-defined Data Models.
* **Ethos Data Access** combines the power of the Ellucian Ethos Data Model and Ellucian Ethos Integration to store data from multiple tenants in the cloud-based data storage. Data Access is meant to enable users to query unified data performantly.
* **Ethos Identity Federation Services (EIFS)** is an essential component of the SaaS platform as it manages authentication, not Ethos User Provisioning (EUP).
* **Ethos User Provisioning (EUP)** is a cloud-based solution that supports both bulk and near real-time provisioning of users and groups from the Ellucian ERPs Banner and Colleague to third-party identity management services.
* **Data Connect** is available as part of Ellucian Experience, which provides the ability to define data workflows that include data transformations and mappings to third-party sources and formats.
* **Change Notifications and Business Events.**
  + **Change Notifications** enable Non Authoritative systems to be notified of changes to Data Model resources. This is enabled by Authoritative systems publishing Change Notification messages when Data Model resources change.
  + **Business Events** specifically inform about the modified columns in the Banner database. You have the option to assign an alias to the column name, which will be used instead of the original column name when publishing a message associated with a business event.

Knowledge Check Questions:

1. Ellucian Ethos provides a unifying framework for data, users, applications, insights, and actions that facilitate the integration of software applications at the data and user interface levels.

Ans: True

1. Which of the following is not an Ellucian Ethos Component?

Ans: Experience

**Lesson 2: Ethos Navigation Overview**

**Home Page**

**Authoritative Applications**

These are systems that **own and manage the official version** of specific data (called **resources** in the Ellucian Ethos Data Model). Think of them as the "source of truth" for certain types of data, such as student records, course information, or employee data.

**Responsibilities:**

1. **Publish Data Changes**:
   * When data changes, they **publish change notifications** to **Ethos Integration**.
   * These notifications **must include the most recent version** of the resource in the standard Ethos Data Model format.
   * This enables other systems to stay updated.
2. **Respond to Data Requests**:
   * They must **respond to API calls** from other systems that need data.
   * This is done via the **Ethos Integration Proxy API Service**.
   * They must expose both the **current version and previous supported versions** of their data via **RESTful APIs**.

**Subscribing Applications**

These are systems that do **not own** a particular data resource but **want to stay in sync** with the authoritative system.

**Responsibilities:**

1. **Consume Change Notifications**:
   * They retrieve changes from the **Ellucian Ethos Integration Message Queue service**.
   * This helps them **update their internal database** to reflect the latest data.
2. **Optional Direct Data Requests**:
   * If needed, they can **issue CRUD operations** (Create, Read, Update, Delete) via the **Proxy API service**.
   * Or they can **query data using GraphQL** via **Ethos Data Access**, especially for read-only operations.

| **Role** | **Description** | **Responsibilities** |
| --- | --- | --- |
| **Authoritative App** | Owns and manages the official data source | Publish updates Respond to data requests (RESTful API) |
| **Subscribing App** | Uses data but is not the source of truth | Sync via message queue Optionally request data (CRUD/GraphQL) |

**Applications Page**

**All Applications Page**

* Displays a **searchable table** listing all applications added to the Ethos Tenant Environment.
* Provides **detailed information** about each application.
* **No limit** to the number of applications you can add.

**Monitoring Dashboard (Monitoring Tab)**

* Shows the **connectivity status** of each application.
* Helps track whether apps are online, connected properly, or facing any issues.

**Application Overview Page: Key Configuration Tabs**

Each application in Ethos Integration has a set of tabs for managing and configuring how it interacts within the ecosystem:

**1. API Keys Tab**

* When an application is created in Ethos, it is assigned a **unique API key**.
* This key is used for **authentication**.
* The application uses the key to obtain a **JWT (JSON Web Token)** which is required to call APIs securely.

**2. Owned Resources Tab**

* Allows you to **specify the data resources** that the application **owns** (i.e., is authoritative for).
* Only the authoritative app should have a resource marked as "owned" — meaning it:
  + Publishes data changes.
  + Responds to API requests for that resource.
* **Do not add resources here** if the application only reads them via subscription, proxy API, or GraphQL.
* For Ellucian apps, some resources can be **auto-added from a catalog** during setup.

**3. Credentials Tab**

* List of login credentials this application needs to **access other applications**.
* You can:
  + Add new credentials.
  + Edit or delete existing credentials.

**4. Request Routing Tab**

* Used when an application makes **proxy API calls** to **multiple target applications**.
* If a resource has **multiple possible owners**, this page lets you:
  + **Manually route** requests to the correct application.
* Otherwise, requests are sent to the **default owner** defined in the Resources page.

**5. Resource Subscriptions Tab**

* Lists resources this application **subscribes to** (i.e., it listens for updates).
* You can:
  + Add or remove resource subscriptions.
  + Modify existing subscriptions.
* These are resources the application **does not own**, but keeps in sync via notifications.

**6. GraphQL Resources Tab**

* Define resources the application will use in **GraphQL queries**.
* You can also apply **data privacy rules** here.
* Security policies will be generated for secure GraphQL usage by this application.

**7. Tags Tab**

* Tags help you **organize and filter** applications by assigning metadata.
* Tags are simple **key-value pairs**.
* Example: Use a tag like integration-package = data-connect-api-package to associate related apps.

**Resources Page**

Data Resources are defined structures of related information using name/value pairs, that define how the data can be interacted with. This includes retrieving, creating, updating, and deleting that data. The Resources Page will display the Resource Name, Major Version, Resource Owner, Domain, and additional actions.

**Reports Tab**

The three reports that will be found under the Reports Tab are:

* Published Messages
* Errors
* Audit Logs

**Published Messages Report**

This report shows all **change notification messages** (data updates) that:

* Are currently **queued** for delivery to subscriber applications.
* Have been **archived** for reference.

**Features:**

* View **individual messages** with more info by clicking the ⓘ button.
* Use the **Filter icon** to search based on criteria:

| **Filter Criteria** | **Description** |
| --- | --- |
| Start and End Date | Filter messages within a time range |
| Change Notification ID | Filter by specific message ID |
| Publisher / Subscriber App | Filter based on sending or receiving app |
| Resource | Specify the type of data (e.g., student, term) |
| Retries | Number of retry attempts for delivery |
| Status | Delivered, failed, queued, etc. |

**Errors Report**

This report helps you **track and diagnose problems** related to message delivery, API calls, or system issues.

**Features:**

* View categorized error types:
  + **Errors** (critical failures)
  + **Warnings** (non-blocking issues)
  + **Info** (helpful messages)
* Click the **Info button** for more details on an error.
* Use the **Filter tool** to narrow down error results using:

| **Filter Criteria** | **Description** |
| --- | --- |
| Start and End Date | Time range of when the errors occurred |
| Application / Platform | Which system or package the error came from |
| Resource Name / ID | Resource involved (e.g., student or 1234) |
| Application Subtype | Further refine by type of application |
| Correlation ID | Useful for tracing specific request-response chains |

* Click the **Action button** to open detailed views of the error:
  + **Overview Tab**: Summary of what happened.
  + **Request Tab**: Technical details of the API call that failed.

**Audit Logs Report**

This page helps track **user activity and configuration changes** in the Ethos Integration system.

**Features:**

* Automatically records actions such as:
  + User login/logout
  + Resource changes
  + Subscription additions
  + Audit logging enable/disable
* **Audit logging is enabled by default** and retains records for **180 days** (older records are automatically purged).
* Shows up to the **latest 5000 records**.
* Use the **Filter** to narrow by:

| **Filter Criteria** | **Description** |
| --- | --- |
| Start and End Date | When the activity occurred |
| User | Who performed the action |
| Category / Sub Category | Type of action or configuration changed |
| Description of Change | Text-based match for action details |
| Status | Result of the action (e.g., success/failure) |

* Use **Exact Match** for precise filtering on User or Description fields.
* Use **Search** for keywords across multiple fields.

**Summary of the Reports Tab**

| **Report Type** | **Purpose** | **Key Use Case** |
| --- | --- | --- |
| **Published Messages** | View and filter change notifications sent to/from apps | Check if data updates were sent |
| **Errors** | View and diagnose issues or failed API calls | Troubleshoot integration problems |
| **Audit Logs** | Monitor user actions and configuration changes | Maintain security & compliance logs |

**Platform Components**

* Configure access to data managed by integration applications on the Credentials tab.
* Specify any APIs that will be able to access data from this Ethos platform component on the Application Access tab.
* View all resources owned by this Ethos platform component on the Owned Resources tab.
* Configure all requests on resources from the available owners on the Request Routing tab.

**Switch Ethos Tenants**

In order to switch between Ethos Tenant environments, click the⚙️gear icon in the upper right. The different Ethos Tenant environments will be displayed, and allow switching to the desired environment.

Knowledge Check Questions:

1. Which of the following pages is not accessible from the Ethos Integration Interface?

Ans: Customer Center

1. To gain access to the Ellucian Ethos Integration environments, first you must obtain an Ellucian Customer Center account.

Ans: True

**Lesson – 3: Ellucian Ethos Infrastructure**

The Ellucian Ethos Platform provides a unifying framework for data, users, applications, insights, and actions that facilitate the integration of software applications at the data and user interface levels.

Ellucian Ethos Integration connects data producers with data consumers and provides the infrastructure through which data and requests for data are exchanged.

**Data Integration Types**

**Point-to-Point Integrations (P2P)**

**What it is:**

* A **direct connection** between two systems that need to exchange data.
* Example: You connect your HR system directly to your Student Information System (SIS) with custom-built code.

**How it works:**

* For each pair of systems, you must:
  + Identify what data to share.
  + Build a **custom connector** for that pair.
* Each integration is **one-to-one**.

**Problems with P2P:**

* **Complex and fragile**:
  + Every connection has its own:
    - Technology
    - Configuration
    - Business logic
  + Updates in one system often break the integration.
* **Exponential growth**:
  + The more systems you add, the more connections you need.
  + For example:
    - 5 systems = 10 connections
    - 10 systems = 45 connections
    - 20 systems = 190 connections!
* **Expensive**:
  + High initial cost to build.
  + Ongoing costs for maintenance and upgrades.
  + Drains resources from other projects.

**Ethos Integration**

**What it is:**

* A **central integration hub** created by Ellucian.
* Acts like a **data and service middleman** for all applications.

**How it works:**

* Instead of each system connecting to each other directly, all systems connect to the **Ethos Integration Hub**.
* The hub handles:
  + Standardized communication (RESTful APIs).
  + Secure, reliable message delivery.
  + Data translation and routing.

**Benefits over P2P:**

**Fewer connections:**

* With Ethos:
  + Each app needs only **one connection** (to the hub).
* Instead of dozens or hundreds of unique links, there’s a single integration point.

**Lower cost and complexity:**

* Less custom code.
* Easier upgrades.
* Faster onboarding of new apps.

**Resilience to change:**

* Apps can evolve independently if they comply with Ethos standards.

**Configurable:**

* You can:
  + Decide which apps are authoritative for specific data.
  + Adapt integrations for your campus’s unique needs.
  + Integrate Ellucian and non-Ellucian apps seamlessly.

**Standards-based:**

* Ethos uses **REST APIs over HTTPS**:
  + Modern, widely supported technology.
  + Secure and scalable.

**Analogy**

Imagine:

* **Point-to-Point** → Like running private phone lines between every two houses in a town. Complex, messy, and expensive.
* **Ethos Integration** → Like a central telephone exchange where everyone has a single line, but can talk to anyone through the exchange.

**Key Advantages of Ethos Integration**

* Simplifies your IT environment.
* Reduces cost and time for integrations.
* Supports cloud and on-premise systems equally well.
* Future-proofs your environment for new applications and upgrades.

**Bottom line:**  
→ Point-to-point integrations are brittle and expensive.  
→ Ethos Integration is a modern, centralized, flexible way to connect all your systems reliably and efficiently.

**Ellucian Ethos Integration Architecture**

Imagine Ellucian Ethos Integration like the **electrical grid**:

* **Power plants** produce electricity → **authoritative applications** produce and publish data.
* **Consumers** use electricity → **non-authoritative applications** consume the data for their own purposes.
* The **electric grid** connects them → **Ethos Integration Hub** connects all applications, handling the flow of data and requests.

**Two Types of Applications**

**1. Authoritative Applications**

* These are the **“source of truth.”**
* They:
  + Own specific data (e.g., students, courses).
  + Expose **APIs** so other systems can access their data.
  + Publish **change notifications** to Ethos Integration when data changes.
* **Examples:**
  + ERP systems like **Banner** or **Colleague**.
  + CRM systems.

These applications are **data producers** — like power plants supplying electricity.

**2. Non-Authoritative Applications**

* These **do not own the data.**
* They:
  + Need data for processes or reporting.
  + Subscribe to **change notifications** to stay in sync.
  + Make **API requests** to authoritative apps to:
    - Read data
    - Create new data
    - Update data
    - Delete data
* **Examples:**
  + Applications like **Elevate** or a campus bookstore system.

These applications are **data consumers** — like households using electricity.

**How Ethos Integration Connects Everything**

Instead of dozens of individual connections (point-to-point), all apps connect to **Ethos Integration Hub**. The hub:

* Routes requests.
* Publishes data changes.
* Maintains standards so systems can integrate smoothly.

**Messaging Patterns in Ethos Integration**

Ethos supports **two key ways** for applications to exchange data:

**1. Publish and Subscribe**

* **Asynchronous.**
* Authoritative app publishes a message when data changes (e.g. a new course is created).
* Ethos Integration:
  + Receives the message.
  + Places it in a **queue** for each subscriber interested in that data.
* Subscribing apps periodically **pull messages** from their queue to update their own databases.

**Example Scenario:**

* A new course is created in Banner (authoritative app).
* Banner publishes a change notification to Ethos.
* Ethos puts it in queues for Elevate and the Bookstore (subscribers).
* Elevate and the Bookstore pull the message and update their systems with the new course.

**2. Request and Reply**

* **Synchronous.**
* Non-authoritative app makes a **live API call** via Ethos Integration to:
  + Create
  + Read
  + Update
  + Delete
* Ethos:
  + Determines which app owns the data.
  + Routes the request to the authoritative application.
  + Returns the response to the caller.

**Example Scenario:**

* Elevate wants to create a new course section.
* Elevate sends a request via Ethos.
* Ethos routes it to Banner (the authoritative app).
* Banner processes it and replies.
* Ethos returns the result to Elevate.

**Key Advantages of Ethos Integration Architecture**

**Simplifies integrations**

* No more dozens or hundreds of fragile point-to-point connections.

**Standards-based (REST, HTTPS)**

* Ensures compatibility and security.

**Handles change notifications**

* Keeps apps synchronized without each app needing to know all others.

**Flexible and configurable**

* Easy to add or replace systems without major rewrites.

**Supports both cloud and on-premises systems**

* Future-proof for evolving technology landscapes.

**Analogy**

Without Ethos:

* Each app builds a separate connection to every other app → **spaghetti mess.**

With Ethos:

* Each app plugs into a central hub → **clean and efficient grid.**

**In summary:**

* **Authoritative apps** = source of data, publish updates.
* **Non-authoritative apps** = consume or request data.
* **Ethos Integration Hub** = the smart middle layer routing data and requests efficiently.
* **Messaging patterns:**
  + **Publish/Subscribe** → for updates and sync.
  + **Request/Reply** → for live API operations.

**Ellucian Ethos Data Model (EEDM)**

Think of the **Ethos Data Model** like a **common language or dictionary** for all your campus systems.

Here’s why it’s needed:

* Different applications might call the same piece of data by different names:
  + One app says **“birthdate.”**
  + Another says **“DOB.”**

This causes confusion and makes integrations messy.

**The Ethos Data Model solves this.**

* It defines **standard names and structures** for data (e.g. “birthDate”) so all systems know exactly what each data element means.
* It’s the **translator** between systems.

**Key Features of the Ethos Data Model**

**A Common Language**

* Standard terms and definitions for data used across all institutions.
* Helps everyone (IT, developers, vendors) speak the same “data language.”

**Single Source of Truth**

* Ensures a consistent view of data, e.g. a single record for each student that all systems agree on.

**Open Platform**

* It’s community-driven, so institutions and partners can help evolve the standards.

**Extensibility**

* Institutions can **extend** the data model:
  + Add custom fields.
  + Adapt the model without changing core systems.

**How the Ethos Data Model Works**

When you use an Ethos API:

* You send or receive data **in the Ethos Data Model format** (JSON).
* No matter whether you connect to Banner, Colleague, or another system:
  + You get the data **in the same structure**.

**Example:**

* Banner stores a student’s birth date as **“birthdate.”**
* A bookstore system uses **“DOB.”**
* Ethos translates both to the same standardized field, like **“birthDate.”**

**Ethos APIs**

All **Ethos APIs** follow **REST standards**:

* They use the HTTPS protocol.
* Data is exchanged in **JSON** format.
* They reflect **real-time state** of data.

**Supported Methods**

| **Method** | **What It Does** |
| --- | --- |
| **GET** | Retrieve data (e.g. get student info) |
| **POST** | Create new data (e.g. add a new course) |
| **PUT** | Update existing data (e.g. change a student’s address) |
| **DELETE** | Delete data (e.g. remove a course) |

**Ethos Proxy API Service**

Here’s where it gets cool…

Imagine Elevate (an application) wants data from Banner but doesn’t know **who owns** the data.

* Instead of calling Banner directly, Elevate sends the request to the **Ethos Proxy API Service.**
* Ethos figures out:
  + Which system is authoritative.
  + Routes the request.
  + Returns the response back to Elevate.

This means:  
Applications **don’t need to know where data lives.**  
Systems are **decoupled**, making integrations simpler.

**Authentication with Ethos Integration**

All applications connecting to Ethos Integration must **authenticate**:

**How it works:**

1. Application uses its **API Key** to get an **access token** (a JWT).
2. That token allows it to:
   * Call Ethos APIs.
   * Publish change notifications.
   * Retrieve change notifications.
3. When the token expires → app requests a new one.

**Two types of API keys:**

* **Unrestricted API Key**:
  + Accepts calls from any app with the correct credentials.
* **Restricted API Key**:
  + Limits requests to specific IP addresses or ranges for extra security.

**Why This Matters**

| **Without Ethos** | **With Ethos** |
| --- | --- |
| Systems speak different languages | Systems share a **common data language** |
| Apps connect directly → complex web of connections | Apps connect through a **central hub** |
| Changes in one app break others | Ethos shields apps from change |
| High cost to build and maintain integrations | Faster, cheaper, more flexible integrations |

**Quick Example Scenario**

* **Banner** (authoritative app) stores course info.
* A **Bookstore system** needs new course info:
  + Instead of connecting directly to Banner, it calls the **Ethos Proxy API**.
  + Ethos knows Banner owns courses → sends the request to Banner.
  + Banner replies → Ethos sends the data back to the Bookstore in the Ethos Data Model format.

**Bottom Line:**

* The Ethos Data Model ensures everyone speaks the same language.
* The Ethos Proxy API simplifies connections.
* Integrations become easier, more secure, and future-proof.

**Lession – 4 Cloud Identity and Access Management**

**What is EIFS?**

**EIFS** stands for **Ellucian Identity Federation Service.**

**Cloud-based identity & access management solution.**  
Built specifically for **higher education.**  
Makes it easy for users to **log in once (Single Sign-On, SSO)** and access multiple Ellucian apps and services seamlessly.

**How EIFS Works**

Imagine your university has:

* Banner (Ellucian ERP)
* Colleague
* Ellucian Experience
* Other cloud apps

Without EIFS:

* Each app might have its own login screen.
* Users have to remember multiple usernames and passwords.

With EIFS:

* **One login → Access to everything.**

**How does EIFS make this possible?**

It works as a **“middleman”** between:

* **Your identity provider (IdP)**
  + e.g. Azure AD, Shibboleth, Google, etc.
* **Ellucian applications**
  + e.g. Banner, Experience

When a user tries to log in:

1. EIFS redirects them to your IdP (like Azure AD).
2. User logs in there.
3. IdP sends a **secure assertion** (e.g. SAML token) back to EIFS saying:

“This person is who they say they are.”

1. EIFS tells the Ellucian apps:

“Good to go — let them in!”

**Key Technology Used: SAML 2.0**

EIFS uses **SAML 2.0**, an open standard for exchanging login info between:

* Identity providers (**IdPs**) → the place users actually log in.
* Service providers (**SPs**) → the apps users want to use.

It’s super secure and uses HTTPS for all communication.

**Benefits of EIFS**

**Single Sign-On (SSO)**

* Users log in **once** → get into all Ellucian apps.

**Cloud-Ready & SaaS-Friendly**

* Designed for modern, cloud-based apps.

**Supports Federation**

* Connects your campus’s identity system to Ellucian apps.

**Multi-Tenancy**

* Supports multiple institutions in a single environment.

**No more managing user stores in Ellucian apps**

* Let your central IdP handle user info.

**Automatic upgrades and patches**

* No maintenance headaches.

**Helps with privacy & compliance**

* Centralizes identity management.

**EIFS vs. EEID**

| **Feature** | **EEID** | **EIFS** |
| --- | --- | --- |
| Federation services | ✅ | ✅ |
| Local user store | ✅ | ❌ |
| User & role management | ✅ | ❌ |
| SaaS support | ❌ | ✅ |
| Multi-Tenancy | ❌ | ✅ |
| Automatic upgrades | ❌ | ✅ |

**Simplified:**

* **EEID** → On-premises identity solution (older tech).
* **EIFS** → Modern cloud service built for SaaS.

**EEID is being sunset.**  
→ All customers are expected to move to EIFS.

**How EIFS Gets Configured**

When connecting EIFS to your systems (like Ellucian Experience):

1. The Experience admin enters **IdP details** into the Experience setup app.
2. Experience sends this info to EIFS.
3. EIFS establishes the trust relationship automatically.
4. EIFS tells Experience:

“Federation setup complete!”

1. The Experience app provides a **test connection** option.

All of this should happen **in near-real-time**, with no manual steps.

**EIFS Use Cases**

Here’s what EIFS helps institutions do:

Provide **one seamless login** for multiple apps.  
Use secure, open standards (like SAML 2.0).  
Reduce costs related to managing user data in multiple places.  
Help comply with privacy laws and regulations.  
Establish **trust relationships** between:

* Your campus’s IdP.
* Ellucian’s cloud apps.

**Real-Life Example**

* Your school uses **Azure AD** as its IdP.
* A student wants to use Ellucian Experience.
* Student clicks “Log In.”
* EIFS sends them to Azure AD to sign in.
* Azure AD verifies identity and sends a SAML token to EIFS.
* EIFS tells Ellucian Experience:

“This is Alex Smith, student #12345.”

* Student gets into Experience **without logging in again**.

**Key Takeaways**

✔ **EIFS = Cloud-based SSO and identity federation for Ellucian SaaS apps.**

✔ **Works with external IdPs** like Azure AD, Shibboleth, Google, etc.

✔ **Uses SAML 2.0** for secure data exchange.

✔ **Replaces EEID** (on-premises) with modern, scalable, cloud architecture.

**In short:**  
EIFS makes life simpler for both users and IT teams — **one login, many apps, and no more identity chaos!**

**Ethos User Provisioning (EUP)**

Imagine a university that uses Ellucian Banner or Colleague as its main system for managing:

* Student data
* Faculty/staff data
* Roles and classifications (e.g. undergrad, grad student, employee, etc.)

**EUP’s job is to take that user identity information and send it to other systems** — especially systems that handle **Single Sign-On (SSO)** like:

* Azure Active Directory
* Okta
* LDAP systems

So that everyone can log in **once** and access all their apps.

**Why Do We Need EUP?**

Here’s the problem:

* Banner or Colleague stores user details (like usernames, roles, affiliations, etc.)
* But those systems **aren’t connected automatically** to identity providers (IdPs) like Azure AD or Okta, which handle SSO.

So:  
Without EUP → IT staff manually copy data or build custom integrations. Painful!  
With EUP → It’s automated and happens **in real time or in bulk**.

**How EUP Works**

Think of EUP as a **bridge** between:

* Banner/Colleague (where the data lives)
* Identity management systems (where login and SSO happens)

**Example flow:**

* A new student enrolls in Banner.
* Banner stores:
  + Name
  + Email
  + Student ID
  + Classification (freshman, sophomore, etc.)
* EUP **pushes that user info** to Azure AD.
* Azure AD can now:
  + Authenticate that student for SSO.
  + Grant access to apps (like Microsoft 365, Ellucian Experience, etc.)

**What EUP Does (and Doesn’t Do)**

**EUP DOES:**

* Push identity data from Banner/Colleague to identity providers like:
  + Azure Active Directory
  + Okta
  + LDAP
  + SCIM endpoints
* Handle **bulk updates** (e.g. sync thousands of users overnight)
* Handle **near real-time updates** (e.g. as records change)

**EUP DOES NOT:**

* Handle logins or authentication itself → that’s EIFS’s job.
* Manage biographical data like addresses or personal info centrally.
* Provide a “master person database” (that’s handled by Ellucian Person Manager).
* Provision data to SPML endpoints (yet).

**EUP vs. EIFS**

It’s easy to confuse **EUP** with **EIFS**, so here’s a simple way to remember:

| **Feature** | **EUP** | **EIFS** |
| --- | --- | --- |
| Provisions user data to IdPs | ✅ | ❌ |
| Handles SSO login process | ❌ | ✅ |
| Sits between Banner/Colleague and IdPs | ✅ | ❌ |
| Sits between IdPs and apps for login | ❌ | ✅ |

**Think of it this way:**

* **EUP → pushes user data into your IdP.**
  + E.g. creating accounts in Azure AD.
* **EIFS → handles the login flow.**
  + E.g. letting users sign in once to access all apps.

You might need:

* **Only EIFS** → if your IdP already has your user info.
* **Both EUP and EIFS** → if you want to sync user data from Banner/Colleague into your IdP AND enable SSO.

**Why Institutions Use EUP**

**Save time**

* No manual export/import of user accounts.

**Consistency**

* The same data flows from Banner/Colleague to your IdP.

**Better security & compliance**

* Keeps your IdP in sync with the authoritative source.

**Flexibility**

* Works with different IdPs (Azure AD, Okta, LDAP, etc.)

**Cloud-native and scalable**

* Built for modern SaaS environments.

**Use Case Example**

Imagine a new faculty member joins:

1. HR enters the faculty member’s info in Colleague.
2. EUP detects the new user.
3. EUP pushes:
   * Name
   * Email
   * Faculty role
   * Department
   * Campus  
     …to Azure AD.
4. Azure AD:
   * Automatically creates the faculty’s account.
   * Assigns licenses for apps like Microsoft 365.
   * Enables SSO for Ellucian Experience, LMS, etc.

Without EUP, IT staff would have to **do this manually or build custom scripts.**

**Key Takeaways**

✔ **EUP bridges the gap** between Ellucian ERPs and your identity provider.  
✔ Automates **user provisioning** → fewer errors and less manual work.  
✔ Helps enable **SSO** by ensuring identity data is in your IdP.  
✔ EUP **doesn’t handle authentication** — that’s EIFS’s job.  
✔ Both EUP and EIFS are cloud-native, built for modern environments.

**In short:**  
**EUP = “Get user data from Banner/Colleague into my IdP.”**  
**EIFS = “Handle the login process and SSO.”**

**EUP ARCHITECTURE**

**What is EUP architecture designed to do?**

* EUP takes **user data from Ellucian’s systems (Banner or Colleague)**  
  ➡ and **creates or updates accounts** in other systems like:
  + **Azure Active Directory**
  + **Okta**
  + **LDAP / Active Directory**
  + **SCIM-compatible systems** (any standard identity platform)

This makes sure:  
- all your systems have the same users  
- with the same details and roles  
- so Single Sign-On (SSO) works smoothly

**How does data travel through EUP?**

**Step 1 — Data Origin: Banner or Colleague ERP**

* Your users exist in your ERP system:
  + Banner
  + Colleague

These store:

* usernames
* email addresses
* roles (student, faculty, staff, etc.)
* other identity data

**Step 2 — Ethos Integration Reads Data**

Ellucian has **Ethos Integration** — it’s like a hub between systems.

* It talks to Banner or Colleague using special APIs:
  + user-identity-profiles → pulls user data
  + identity-profile-roles → pulls list of roles

➡ So Ethos Integration knows whenever someone new is added or updated in Banner or Colleague.

**Step 3 — EUP Event Consumer Gets Notifications**

Instead of pulling data constantly, Ethos Integration can **send change notifications**.

Example:

“Hey, a new user got added in Banner!”

EUP’s **Event Consumer**:

* listens for these messages
* builds a payload with user info

**Step 4 — Data Enters the Cloud (AWS)**

The Event Consumer pushes the data to **AWS SQS (Simple Queue Service)** — a message queue.

This:

* decouples systems
* lets EUP handle big bursts of changes without crashing

**Step 5 — AWS Lambda Provisions the User**

From the queue:

* AWS Lambda functions:
  + grab the user data
  + transform it into the right format
  + send it to your Identity Provider (IdP)

➡ Different IdPs have different APIs. For example:

* Azure AD → uses Microsoft Graph API
* Okta → uses Okta REST API
* SCIM → standard SCIM API endpoints

So Lambda adapts the data for each target system.

**Two Different Architectures**

Here’s the big split:

**Architecture A — Cloud-Only IdPs (Okta, Azure AD, SCIM)**

These IdPs live in the cloud. No need for anything on-premises.

**Flow:**

1. ERP → Ethos Integration
2. Ethos → Event Consumer
3. Event → AWS SQS
4. Lambda → calls Azure AD, Okta, or SCIM
5. Done!

A diagram of a software company

AI-generated content may be incorrect.

**Architecture B — On-Premise IdPs (LDAP / Active Directory)**

If your IdP is on-premises (inside your network), it can’t just be called from the cloud directly.

Instead:

* EUP installs a **local Agent** on your network.
  + Written in Node.js
* This Agent:
  + connects to AWS over secure WebSockets
  + receives user data
  + provisions users into:
    - Active Directory
    - LDAP servers

**Flow:**

1. ERP → Ethos Integration
2. Ethos → Event Consumer
3. Event → AWS SQS
4. Lambda → pushes data to Agent
5. Agent → provisions users locally

A diagram of a cloud computing system

AI-generated content may be incorrect.

**Security**

Everything:

* travels over secure HTTPS or WSS (WebSockets)
* uses API keys and tokens for authentication
* ensures privacy of identity data

**Why EUP needs two architectures?**

* **Cloud IdPs**:
  + Easy — everything stays in the cloud
* **On-prem IdPs**:
  + Firewalls block direct cloud access
  + Need an on-prem agent to “bridge the gap”

**Example Scenarios**

**Scenario 1 — Azure AD**

* Student joins Banner
* Change event fires → Ethos picks it up
* EUP Event Consumer sends to AWS SQS
* Lambda reads it
* Lambda calls Microsoft Graph API
* Student appears in Azure AD → SSO ready!

No agent needed.

**Scenario 2 — Active Directory On-Premises**

* Employee added in Colleague
* Change event fires → Ethos picks it up
* Event Consumer sends to SQS
* Lambda queues provisioning
* EUP Agent connects from your data center
* Agent provisions user into local AD
* Done!

Agent needed because your AD server isn’t in the cloud.

**What EUP Does Not Do**

* **Does NOT handle logins or passwords directly.**  
  That’s EIFS’s job (Ellucian Identity Federation Service).
* **Does NOT store personal biographical data.**  
  It only moves data for provisioning.
* **Does NOT replace your Identity Providers.**  
  It connects them to your ERP data.

**Quick Recap**

* EUP moves users from Banner/Colleague → IdPs
* Uses:
  + Cloud-only architecture (Azure AD, Okta, SCIM)
  + Or on-premises agent architecture (LDAP, AD)
* Built on:
  + Ethos Integration
  + AWS SQS
  + Lambda
  + Optional EUP Agent
* Keeps your identities in sync → SSO works perfectly

**That’s the architecture of EUP in a nutshell!**

**Lesson – 5: Cloud Data Storage and Extensibility**

**Ethos Data Access**

Ellucian Ethos Data Access combines the power of the Ellucian Ethos Data Model and Ellucian Ethos Integration to store data from multiple tenants in the cloud-based data storage. Other applications can then consume that data through Ellucian Ethos Integration.

**What are Change Notifications?**

Suppose you change some data in your ERP—for example:

* You update a student’s address
* You add a new student
* You change someone’s privacy settings

These changes need to **flow into Ethos Integration** so any connected systems stay up-to-date.

→ That’s where **change notifications** come in.

Think of them as:

Little messages saying: **“Hey! This record changed!”**

**How Change Notifications are Processed**

Here’s how Ellucian handles these notifications step by step:

**➤ Step 1 – Stored in a Queue**

When your ERP system generates a change notification, Ethos Integration:

* **puts the message into a queue**

This ensures:

* changes don’t get lost
* they’re processed in order

**➤ Step 2 – Data Access Checks the Queue**

The Ethos component called **Data Access** regularly looks into this queue.

“Are there any new changes waiting for me?”

If yes—it grabs them.

**➤ Step 3 – Updates the Cloud Data Storage**

Once Data Access gets the changes:

* it **updates the data in the cloud**

This means:

* your cloud environment stays synced with your ERP
* applications using Ethos APIs see the latest data

**What Happens During a Reload?**

Now, sometimes you need to **reload entire data sets.**

Examples:

* you change privacy settings for a resource
* you change the authoritative application for a resource (e.g. moving from Colleague to Banner)
* you upgrade to a new version of the ERP

**When this happens:**

→ You tell Data Access to **reload the whole resource.**

But what if new changes come in during that reload?

**Pausing Change Notifications for Reloads**

While Data Access reloads a resource:

* it **pauses change notifications only for that specific resource.**

Other resources keep processing normally.

So:  
Address changes still process  
Student changes still process  
Person updates pause until reload finishes

Once the reload finishes:

* any queued changes for that resource **are processed afterward** so nothing gets lost.

**Scenario**

An admin decides:

“Let’s reload the **persons** data.”

* Maybe privacy settings changed
* Or they switched from one ERP to another

**Meanwhile, other change notifications are still arriving:**

* Students are being updated
* Addresses are being updated

**➤ What happens?**

* **Change notifications for persons → PAUSED.**  
  They go into a queue.
* **Change notifications for addresses and students → PROCESSED RIGHT AWAY.**

Once the person reload finishes:

* Data Access **processes all the queued changes** for persons
* So you don’t lose any updates

**When to Reload?**

Ellucian recommends a reload after changes like:

* resource privacy settings changed
* authoritative application changed
* ERP version upgraded

This ensures:  
data in Ethos matches your ERP  
no old data hanging around

**Outdated Status Tag**

When Ethos detects a reason to reload:

* it shows an **outdated status tag** next to the resource

Hover over the tag → you see why Ethos thinks a reload is needed.

**BUT…**

Ellucian warns:

**Don’t rely only on this tag.**  
Always evaluate whether you should reload data for your institution’s needs.

**Summary**

So, the core idea:

* Ethos Integration listens for change notifications from your ERP.
* Those changes go into a queue.
* Data Access processes them into cloud storage.
* If you reload a resource:
  + changes for that resource pause and queue up
  + changes for other resources continue
* After reload, queued changes get processed so nothing is missed.

This **keeps data consistent and reliable** between your ERP and cloud applications.

**Ethos Extend Overview**

**What is Ethos Extend?**

**Ethos Extend** is a **cloud-based tool** (SaaS) provided by Ellucian.

It’s part of the **Ellucian Ethos platform**, whose big job is:

➤ Helping different systems and applications share data in higher education.

**Ethos Extend** specifically:

* Allows you to **customize the data models** that define how data is structured and shared between systems.
* Lets you **create new data models entirely** for your unique needs.

Think of it as:

**A workshop where you customize the blueprint of your data.**

**How It Fits into Ellucian Ethos**

* Ethos uses **Ethos Data Models (EDMs)** for communication between systems.
* These data models define:
  + What fields exist (e.g. “birthDate” vs. “DOB”)
  + How data relates
  + Standard formats

But institutions often need:

* extra fields (e.g. “preferredLanguage” for students)
* completely custom objects (e.g. “Library Access Profile”)

This is why **Ethos Extend exists** — to let you adapt or extend those models.

**How Do You Access Ethos Extend?**

→ Only through **GraphQL APIs** managed by Ethos Integration.

Meaning:

* You can’t directly hit your extended models via REST APIs.
* GraphQL is the pathway for integrations using those extended models.

**What You Can Do with Ethos Extend**

Here’s what Ethos Extend allows:

**Extend Existing Data Models**

* Add new custom fields to existing Ellucian data models.
* For example:
  + Add a “Preferred Language” field to the standard Person model.

**Create Completely New Custom Models**

* For unique institutional data needs.
* For example:
  + Define a new “Parking Permit” data model if your ERP doesn’t track that.

**Do All This Without Breaking the Standard Models**

* You **don’t modify Ellucian’s baseline definitions.**
* Instead, your extensions layer on top, ensuring:
  + upgrades stay smooth
  + integrations don’t break

**Semantic Versioning**

* Data models have **version numbers.**
* Helps you track:
  + what changed between versions
  + whether your integrations might need adjustments

**Ethos Data Models — Key Points**

**Ellucian Ethos Data Models (EDMs):**

* Are **standardized**  
  → so all applications can “speak the same language.”
* Are **generic**  
  → not tied only to Ellucian apps; they can be used by other apps too.
* Are **extensible**  
  → designed so you can safely add:
  + new fields
  + new relationships
  + entirely new models

**Features of Ethos Extend**

Here’s the feature set summarized:

| **Feature** | **What It Means** |
| --- | --- |
| Extend existing Ethos Data Models | Add new properties (e.g. new fields) for your institution’s needs. |
| Create new custom data models | Define new models from scratch for unique business processes. |
| Data model-centric UI | Easy search, navigation, and editing through a dedicated user interface. |
| View properties by version | See exactly which fields exist in each model version. |
| Download JSON schemas | Export models as JSON to help integrate with external systems. |
| Bulk download | Download all latest models as a single zip archive. |

**A Practical Example**

Let’s say:

* Your institution uses the standard Person data model.
* The standard model doesn’t have a field for “Preferred Pronouns.”

With Ethos Extend:

* You can **extend** the Person model.
* Add a custom field: “preferredPronouns.”
* Your integrations can now include this data via GraphQL queries.

**Why This Matters**

Using Ethos Extend:

You stay in sync with Ellucian’s standards.  
Your custom needs are supported without breaking future upgrades.  
Your integrations become smoother and more maintainable.  
You avoid heavy customization directly in your ERP systems.

**Quick Analogy**

Imagine Ethos Data Models as:

**Blueprints for buildings.**

Ethos Extend:

Lets you:

* Add new rooms (custom fields)
* Build new types of buildings entirely (custom data models)
* Without tearing down the original architecture.

**Key Takeaways**

* Ethos Extend = Cloud-based toolkit to customize data models.
* Works only through GraphQL integrations via Ethos Integration.
* Allows safe extensions to standard models or entirely new models.
* Helps institutions integrate unique processes while staying upgrade-safe.

**Tenants, Environments, and Data ModelsTop of Form**

**Tenant**

A **tenant** = your institution’s “space” in the Ethos platform.

Example:

* University of Exampleton → has a single tenant.
* A multi-campus university → might have a tenant for each campus.

**Environments**

Inside a tenant, you have **environments** like:

* Test
* Production

These environments:

* Are completely **separate from each other**
* Let you safely experiment without breaking your live data

Example:

* In **Test**, you try adding “ldapUserId” to the Person model.
* Once it works, you copy those changes into **Production**.

**Ethos Extend in Action**

Here’s how you’d use Ethos Extend:

You look at the **Person** data model in Ethos.  
You decide you need a new field:

* e.g. **ldapUserId**  
  You extend the Person model:
* You **do not change the original model.**
* Your extension is layered on top.  
  Now, your APIs can read and write data to that new ldapUserId field.

Because:

* All your environments (Test, Production, etc.) are separate, you can:
  + Test changes safely
  + Avoid impacting live integrations
* Each environment **contains its own set of data models** (and extensions)

**Example Scenario**

**Example from your text:**

During testing in the test environment, you extend the Persons data model with a new property:

* **ldapUserId**

Why?

* Your institution uses LDAP for login and needs to store that ID.

How it works:

* You test this change in **Test.**
* Once verified, you move it to **Production.**
* Now all systems integrated via Ethos can read/write ldapUserId as part of the Person data model.

**Why Is This Useful?**

* You get **standardized data** across all systems.
* You avoid “one-off integrations” that break easily.
* You can **customize** without breaking future updates from Ellucian.
* You safely test changes without risking your live environment.

**Short Analogy**

Think of Ethos like:

* **A Lego set.**
* The Ethos Data Model = standard Lego bricks.
* Ethos APIs = tools to build things from those bricks.
* Ethos Integration = conveyor belt moving Lego builds between rooms.
* Ethos Extend = custom Lego bricks you design yourself.

**Key Takeaway**

Ellucian Ethos lets you:

**Standardize your data → Integrate systems easily → Customize where needed.**

* The Data Model = blueprint.
* The APIs = tools to work with the blueprint.
* Integration = the highway for moving data.
* Extend = your way to customize the blueprint safely.

Bottom of Form

**Lession – 6 : Ethos Integration Fundamentals for Administrators v2024**

* Non-Authoritative applications require the data produced by authoritative applications for their own functional processing and reporting.
* A **Proxy API consumer** is an app that need to create, update, or delete data they use but do not "own" can make requests to Ethos Integration, which is responsible for routing the request and returning the responsive from the authoritative application.
* Applications that are configured as subscribers to resources must request the messages that Ellucian Ethos Integration stores for them.

**What Is a Proxy API Request?**

A **proxy API request** allows **one application (the requester)** to access data **from another application (the target)** using **Ethos Integration** as the middleman.

**Why Do You Need Authentication?**

To ensure that **only trusted applications** can access data from another system, **authentication credentials** are required.

When App A wants to request data from App B:

* App A must prove it's allowed to make that request.
* This is done by supplying **user credentials** or an **API token** configured in App B.

**Types of Authentication**

Ethos Integration supports **two types** of authentication for proxy API requests:

**1. Basic Authentication**

* You create a **user** in the **target application** (e.g., Colleague).
* The requesting app includes:
  + **Username**
  + **Password**
* That user must have appropriate **permissions** (like READ, CREATE, UPDATE, DELETE).
* Commonly used for **Banner** or **Colleague** systems.

**2. User-defined Authentication**

* You create an **API token or key** in the **target application**.
* The requesting app includes that token in the request.
* This is more flexible and often used with **custom or third-party systems**.

**Example Scenario: Campus Card System Reads Housing Data from Colleague**

**Goal:**

Your **Campus Card System** (Requesting App) needs to **read housing assignments** from **Colleague** (Target App).

**Steps:**

1. **Set up Colleague in Ethos Integration**
   * Create an **Ethos application for Colleague**.
   * Ensure **Colleague owns** the housing-assignments resource.
2. **Create a Colleague User**
   * Go into Colleague and create a **user** with **READ access** to housing-assignments.
3. **Set up the Campus Card System in Ethos**
   * Create an **Ethos application** entry for the **campus card system**.
   * While setting it up (via the **Add Application Wizard** or by editing later), provide:
     + The **Colleague user’s credentials** (for Basic Auth)
     + OR the **API token** (for token-based authentication)
4. Now, when the campus card system makes a proxy API call:
   * Ethos Integration checks the credentials.
   * If valid, it routes the request to Colleague.
   * Colleague processes it and sends the data back.

**Why Do You Need a JSON Web Token (JWT)?**

* You want your app (e.g. a custom portal, integration tool, or reporting app) to call **Ethos proxy APIs** to access data (e.g. people, courses, housing info) in Ellucian systems like Banner or Colleague.
* **Ethos requires authentication.**
* To prove your app’s identity, you get a short-lived access token (JWT).
* You **send this token in the header** of every API call, so Ethos knows:
  + Who you are
  + What resources you’re allowed to access

**The Process in Simple Steps**

**STEP 1 – Set up an API key in the target app**

Before anything:

* You create an **API key or token** in the **target application** (e.g. Banner, Colleague).
* This key represents your app and its permissions.

Example:

* Your app “Campus Mobile” wants data from Colleague.
* You create an API key in Colleague, granting read access to persons data.

**STEP 2 – Check Ethos Application Settings**

In Ethos Integration:

* Go to the **Application Overview** page for your target app (e.g. Colleague).
* Confirm:
  + The auth type (Basic Auth or token-based) matches your setup.
  + The target app “owns” the resources you want to access.

Example:  
Authentication Type: Basic Auth  
Colleague owns housing-assignments resource

**STEP 3 – Get Your JWT Access Token**

You can manually test your token flow with an API tool like **Bruno** (an open-source alternative to Postman).

**➡ How to Get Your Token Using Bruno**

**1. Create a Collection**

Think of this like a folder to store all related requests.

**2. Add Folder: “Admin Requests”**

**3. Add Request: “Get an Access Token”**

* **URL:** https://integrate.elluciancloud.com/auth
* **Method:** POST
* **Header:** Authorization: Bearer {{apiKey}}
* The {{apiKey}} is your token from the target app.

When you send this POST request, Ethos returns a **JWT access token** in the response body.

**4. Save the Token**

* Copy the token from the response.
* Save it as an environment variable called {{accessToken}}.
* **Tokens expire quickly** (usually in 5 minutes). Refresh them often!

**STEP 4 – Test a Proxy API Call**

Now that you have a valid JWT, try to fetch data.

**➡ How to Test Proxy API Requests**

**1. Add Folder: “Proxy API”**

**2. Add Request: “Get Persons”**

* **URL:** https://integrate.elluciancloud.com/api/persons
* **Method:** GET
* **Headers:** Authorization: Bearer {{accessToken}}

Accept: application/vnd.hedtech.integration.v12+json

This makes a proxy API call to Ethos Integration to fetch “persons” data from the target system.

**What Should the Response Look Like?**

The response should:

* Contain JSON records for **persons**.
* Follow the data model schema (fields like id, firstName, lastName, etc.).
* Include useful headers:
  + x-max-page-size → how many records you can fetch per page.
  + x-total-count → total number of records available.

**Example: Expected API Flow**

1. You POST your API key → get back JWT access token.
2. You save the token.
3. You GET https://integrate.elluciancloud.com/api/persons → Ethos sends you JSON data for people.
4. Repeat as needed, refreshing your token every few minutes.

**Important Notes**

**Tokens expire quickly** → keep your app logic ready to refresh them.

Your token **determines what data you can see** (based on permissions in the target app).

Always include: Authorization: Bearer <your-access-token>

**Accept headers** define which version of the Ethos API you’re calling (e.g. v12).

**In Summary**

**Getting a JWT = your “ticket” into Ethos APIs.**  
Without it, no API calls will work!

**What is Ellucian Ethos Data Access?**

It’s a **cloud-based service** that:

Collects data from Ellucian systems like Banner and Colleague (on-premise or cloud)  
Stores that data in **secure cloud storage**  
Makes it available to other apps through **Ethos Integration**

Think of it as:

**A big cloud database for all your institution’s key data—ready to share securely across apps.**

**How Data Gets into Ethos Data Access**

There are **three ways data goes into Data Access:**

**① Initial Load**

* Used the **first time you set it up**.
* Data from Banner, Colleague, etc. is extracted and loaded into cloud storage.
* Populates the cloud database with all your institution’s data.

**② Updates via Change Notifications**

* When data changes in Banner or Colleague (e.g. a student changes address), the ERP sends a **change notification** via Ethos Integration.
* Ethos Data Access listens for these changes and updates only the **changed pieces of data** in the cloud storage.
* Keeps the cloud copy **fresh and in sync**.

**③ Reload**

* Sometimes you want to **reload everything**. For example:
  + If the Ethos Data Model changes
  + If privacy settings change
  + If you upgrade your ERP
* Reload pulls all data again from the authoritative source to ensure everything matches the new definitions.

**How Data is Consumed**

Apps can read this cloud data in **real-time** using:

* **GraphQL requests** → flexible, optimized queries that fetch only the data you need.
* Or sometimes direct requests to the authoritative source (depending on configuration).

So your apps don’t have to go directly to Banner or Colleague every time. They can talk to **Data Access** instead.

**Creating a GraphQL Consumer App**

Let’s explain the **activity steps** you shared!

Imagine you want an app to fetch data from Data Access via GraphQL—for example, a custom portal, reporting dashboard, or analytics tool.

Here’s how you’d set it up:

**Step 1 – Add a New App**

* Go to the **APPLICATIONS tab** in Ethos Integration.
* Click **Create New App** → choose **MANUALLY**.
* Check:

Configure GraphQL resources

* Click **CONTINUE**.

This signals:

“This app wants to fetch data using GraphQL.”

**Step 2 – Configure Details**

**Enter the App Details**

* Give your app a **name** (required).
* Enter a **description** (optional).

Example:

Name: Analytics Dashboard

Description: Dashboard consuming student and course data from Data Access

**Add GraphQL Resources**

* Choose the data you want your app to access.
* These resources must already exist in Data Access. For example:
  + persons
  + courses
  + academicPrograms

Think of this like:

“Telling Ethos what data this app will be allowed to query.”

**Step 3 – Finish**

**Configure Subscriptions? (Optional)**

* Skip this for now unless you want your app to get **automatic notifications** when data changes.
* Click **SKIP**.

You’ll see a **Confirmation Dialog**:

* Ethos generates an **API Key** for your app.
* You can go straight to the **Application Overview page** to view details.

**API Key** → You’ll use this to authenticate GraphQL requests from your app.

**How It All Connects**

Banner/Colleague → send data → Ethos Data Access → GraphQL API → Your App

**Benefits**

* Centralized data store → simplifies integrations
* Less load on Banner/Colleague → better performance
* GraphQL → fetch only what you need, improves efficiency
* Secure → multi-tenant, controlled access
* Near real-time → thanks to change notifications

**In Summary**

Ellucian Ethos Data Access:

**Loads data from Ellucian systems into the cloud → lets apps consume it via GraphQL APIs → keeps it fresh and secure.**

Creating a GraphQL consumer:

* Register an app
* Choose GraphQL resources
* Get an API key
* Start sending queries!

**What is the Available Resources API?**

This API is part of Ellucian Ethos Integration. It answers the question:

**“Which systems in my institution own which data, and how do I access it?”**

It’s like a **catalog** of all data resources your tenant (your institution’s cloud environment) knows about.

**What does it return?**

When you call the **/available-resources** API, it gives you:

* **List of authoritative applications** → e.g. Banner, Colleague, etc.
* The **resources** they own → e.g. persons, courses, sections, academicPrograms
* For each resource:
  + Resource name
  + Version number (e.g. v1.0, v2.0)
  + Supported methods (GET, POST, PUT, DELETE)
  + Properties that support filtering

So you can discover:

* Who owns “students”?
* Can I filter “courses” by department?
* What versions of “persons” exist?

**Why is this useful?**

* Helps developers **understand your data landscape**.
* Makes it easy to integrate applications correctly.
* Avoids errors from calling the wrong endpoints or resources.
* Provides details you’d otherwise have to look up manually.

Imagine you want to integrate your LMS with Banner. Instead of guessing which API endpoints to call, you query **/available-resources** and see:

**Banner owns persons v1.0**  
Supports **GET, POST, PUT**  
Allows filtering on lastName, personId, etc.

**How do you use the API?**

Here’s how to fetch the list of resources from Ethos Integration.

**➊ Build a GET request**

* **URL:** https://integrate.elluciancloud.com/admin/available-resources
* **Method:** GET
* **Header:** Authorization: Bearer {{accessToken}}

You need the access token from your previous Ethos authentication process.

**Optional: Filter by application**

If you only want resources for a single app, add a query parameter:

https://integrate.elluciancloud.com/admin/available-resources?applicationId=<id-of-the-app>

Where <id-of-the-app> is the **applicationId** assigned to Banner, Colleague, etc.

**Expected Response**

The API responds with JSON data like:

{

"x-total-count": 5,

"applications": [

{

"applicationId": "abc123",

"applicationName": "Banner",

"resources": [

{

"resourceName": "persons",

"resourceVersion": "1.0.0",

"methods": ["GET", "POST", "PUT"],

"filters": ["lastName", "emailAddress"]

},

...

]

},

...

]

}

So you can easily see:

* Which app owns which resources
* What actions you can perform on those resources
* Which fields you can filter by in queries

**Response Headers**

You’ll also get:

x-total-count: <number of applications listed>

Example:

x-total-count: 5

Meaning there are 5 authoritative applications registered.

**How to Test This (e.g. Bruno)**

If you’re using Bruno:

* Go to the **Admin Requests** folder in your collection.
* Create a request named **Available-resources**.
* Configure:
  + URL: https://integrate.elluciancloud.com/admin/available-resources
  + Method: GET
  + Header: Authorization: Bearer {{accessToken}}
* Send the request!

**Where Else Can You See This Info?**

Besides calling the API:

* You can also **log into Ethos Integration in your browser.**
* Go to the **Resources tab**.
* See the same list of applications and resources.
* Note: Sometimes the UI takes a couple of minutes to load because it’s pulling data from the backend.

**In Summary**

The **Available Resources API** is your **directory of resources** in Ethos.

It tells you:

* Who owns which resources
* How you can interact with those resources
* Technical details like versions, filters, and methods

Use it to:  
Plan integrations  
Build correct API calls  
Avoid surprises during development

**What are Audit Logs in Ethos Integration?**

Audit logs are **records of what happens** in your Ethos environment.

Think of them as a security and troubleshooting **trail of evidence**, capturing:

* Who did what
* When it happened
* What the action was
* Whether it succeeded or failed

They help you:  
Monitor system health  
Track changes across apps  
Investigate errors or suspicious activity  
Comply with security or auditing requirements

**The Audit Logs API**

Ellucian Ethos provides an **Audit Logs API** so you can:

* **Store logs** centrally from different applications
* **Retrieve logs** to review activities programmatically

Any app connected to Ethos can:

* **Publish logs** (by sending POST requests)
* **Retrieve logs** (by sending GET requests)

**How to Get Audit Logs**

You retrieve logs by sending a **GET request** to:

https://integrate.elluciancloud.com/audit-logs

**Headers**

You must provide your authentication token:

Authorization: Bearer {{accessToken}}

This verifies that your app has permission to read logs for your tenant.

**Optional Filters**

Instead of pulling **all logs**, you can filter your results using parameters in the request:

| **Parameter** | **What it filters** |
| --- | --- |
| limit | Max # of logs to return |
| offset | Skip N logs before starting |
| startDate | Only logs **after this date** |
| endDate | Only logs **before this date** |
| user | Only logs from a specific user |
| category | E.g. “authentication,” “data-change,” etc. |
| subCategory | More specific type of the category |
| description | Keyword search in the log’s description |
| status | E.g. “SUCCESS,” “FAILURE,” etc. |

These filters help you narrow down logs for:

* Specific time periods
* Specific users
* Specific types of activities

**What Will the Response Look Like?**

When you send the GET request, the response is a **JSON object** that includes:

* An array of **audit log entries**
* A header x-total-count showing how many logs were returned

For example, a log entry might look like:

{

"timestamp": "2025-06-25T14:03:02Z",

"user": "jsmith",

"category": "authentication",

"subCategory": "login",

"description": "User jsmith successfully logged in.",

"status": "SUCCESS"

}

So you know:

* **When** it happened
* **Who** performed the action
* **What** they did
* **Outcome** (success, failure, etc.)

**How to Test This (e.g. Bruno)**

In Bruno, you’d:

1. Go to your **Admin Requests** folder.
2. Create a new request called **Audit Logs**.
3. Configure:
   * URL: https://integrate.elluciancloud.com/audit-logs
   * Method: GET
   * Header: Authorization: Bearer {{accessToken}}
4. (Optional) Add query parameters for filters, e.g.:

startDate=2025-06-01&endDate=2025-06-26&status=FAILURE

1. Click **Send**.
2. Inspect the JSON response!

**Where Else Can You See Audit Logs?**

Besides calling the API:

* You can also log into the **Ethos Integration UI** in your web browser.
* Go to the **Reports tab**.
* There, you can browse audit logs visually without needing API tools.

Note: Depending on system activity, it might take a couple of minutes for logs to appear in the interface.

**Example Use Cases**

Here’s when you might check audit logs:  
Who changed a user’s roles?  
Were there failed login attempts?  
Did anyone delete resources via API?  
Was there suspicious activity during a certain period?  
Prove compliance for a security audit.

**In Summary**

**Audit logs** keep track of actions across your Ethos environment.

* You can **query logs via API** for integration into monitoring systems.
* Or **view logs in the UI** under the Reports tab.
* Filtering makes it easy to find specific events.

This helps keep your systems secure, traceable, and compliant.

**Errors – API**

The Ethos Integration Errors API is a public-facing API that allows applications participating in Ethos Integration to publish errors so Ethos Integration can provide centralized management and monitoring of participating applications and their conditions.

**Set up Permissions to Access Ethos Integration Experience Cards**

**What Are We Doing?**

We’re talking about **setting up permissions** for special dashboard cards in **Ellucian Experience** so that only the right users can see specific Ethos Integration information.

* These “cards” are widgets or tiles you see on the Experience dashboard.
* They show info from Ethos Integration, like:
  + Applications info
  + Error logs
  + Audit logs

For security, you don’t want everyone seeing these—only **admins** or specific roles.

**Two Key Parts**

There are **two main things you do**:

1. **Configure permissions**
   * Decide which people or roles are allowed to view certain Ethos Integration pages or cards.
2. **Set up the cards themselves**
   * Place the cards into the Experience dashboard and link them to the right roles.

Let’s go step by step.

**Part 1 – Set Up Permissions**

**➤ Step 1: Open Experience Setup**

* Go to the **Ellucian Customer Center.**
* Click **Tools** → find the Experience Setup option under “Experience.”
* You may see:
  + **Experience Setup (Test)** → for your testing environment
  + **Experience Setup** → for your live production environment

If you have extra Ethos environments (like Development), you’ll be able to choose those too in the **Environments dropdown** once inside Experience Setup.

**➤ Step 2: Go to Permissions**

* In the Experience Setup menu:
  + Click **Permissions**
  + Then choose **Ethos\_Integration**

This opens the page where you control who can **view** Ethos Integration content.

**➤ Step 3: Select Permissions You Want to Grant**

You’ll see options like:

**Applications > Configuration**  
→ Controls who can see applications info

**Reports > Error Messages**  
→ Controls who can view errors page

**Reports > Audit Logs**  
→ Controls who can view audit logs page

Check the boxes for what you want people to be able to see.

**➤ Step 4: Assign to Roles or Users**

* Next to each permission, there’s a **View checkbox**.
* Check this for the:
  + Roles (like “Admins”)
  + Or specific people  
    who should have permission.

**➤ Step 5: Save**

* Click **Save** to lock in your permissions.

Now, only those people can access the parts of Ethos Integration you’ve allowed.

**Part 2 – Set Up Ethos Integration Cards**

After permissions are set, you **place the cards into the dashboard**.

These are the cards you’re setting up:

| **Card Title** | **Card Type** | **What It Shows** |
| --- | --- | --- |
| Ethos Integration - Applications | EI Applications Card | Info about Ethos apps |
| Ethos Integration - Audit Logs | EI Audit Logs Card | Security & system logs |
| Ethos Integration - Errors | EI Errors Card | Errors from Ethos components |

**How to Add Cards**

Follow the **card setup wizard** in Ellucian Experience (this is a standard wizard for all cards).

**➤ Step 1: Search for Cards**

* In the wizard’s **search field**, type:
  + EI Applications Card
  + EI Audit Logs Card
  + EI Errors Card

→ This helps you find the specific Ethos cards you want.

**➤ Step 2: Edit the Card**

* In the **Card Management table**, find the card you want.
* Click to **edit** it.

**➤ Step 3: Assign Roles to the Card**

In the **Roles section** of the wizard:

* Pick the **same roles** you configured permissions for earlier.
* This connects the card to only those people who should see it.

Example:

If you gave “Admins” permission to see Audit Logs, make sure the EI Audit Logs Card is assigned to the “Admins” role in the wizard.

**In Summary**

So the process is:

Configure **permissions** for who can view:

* Applications
* Error logs
* Audit logs

Add the **cards** to the Experience dashboard:

* Search for the Ethos cards
* Edit them
* Assign them to the right roles

This ensures **only authorized users** see sensitive Ethos Integration info.

**Why It Matters**

* Prevents unauthorized access to sensitive data
* Lets different teams see just what they need
* Keeps your Ethos dashboard secure and organized

Top of Form

**Lesson – 7 : Ethos and Data Connect Cards in Experience**

**What Is Ellucian Experience?**

**Ellucian Experience** is like a **digital front door** for your college or university.

Imagine a single website or portal where:  
Students  
Faculty  
Staff  
Administrators

…can all **log in once** and see:

* Personalized information
* Links to key systems (like your ERP)
* Notifications
* Cards summarizing useful data
* Pages with more details

It brings together data and features from multiple systems into **one place** so people don’t have to jump between lots of different apps.

**➤ How Does It Work?**

**It Connects to Your Other Systems**

Experience pulls data from:

* Ellucian ERPs (Banner, Colleague, PowerCampus)
* Other Ellucian solutions
* Possibly third-party systems

It does this via **Ethos Integration** (Ellucian’s integration platform).

**What Is Displayed to Users?**

**➤ Two Main Things**

**1. Cards**

* Small widgets (like tiles) on the dashboard
* Show summaries or live info
* Examples:
  + Your class schedule
  + Account balance
  + Notifications about holds
* Some cards are:
  + Delivered by Ellucian
  + Custom-made by your institution using templates
  + Developer-built with Ellucian Experience SDK (requires Experience Premium)

**2. Pages**

* Full-page experiences you open from cards or menus
* Hold more **detailed information**
* Examples:
  + Detailed list of all your classes
  + Settings screens for applications
* Pages can be:
  + Created with the Page Designer (no-code tool)
  + Delivered by Ellucian as “extensions”
  + Built by your developers with the SDK (Premium feature)

**➤ How Is Content Personalized?**

Your institution sets up:  
**Roles** → like Student, Faculty, Advisor, etc.  
Permissions → who can see what cards/pages

* When a user logs in:
  + Experience **knows their role**
  + Shows only the cards and pages they’re allowed to see
* For example:
  + Students see classes and financial aid cards
  + Faculty see rosters and grade submission cards

**➤ Where Can Users Find Cards and Pages?**

**Experience Dashboard**

This is the user’s **Home Page.**

* Shows the cards the user has saved there
* Cards can be added from:
  + **Discover page** → see all cards available to the user
  + **Category pages** → focused groups of cards (e.g. Academics)

**Application Menu**

Some tools don’t show as dashboard cards. Instead, they appear in the:

* **Application Menu** (a navigation menu in Experience)

E.g.:

* **Person Manager** → accessed directly from the menu, rather than from a card

**➤ How Cards and Pages Work Together**

**Cards = summaries or quick snapshots.**

* E.g. the Classes card might say:
  + “You have 3 classes today.”

Click the card → opens a **Page** with more info:

* Full list of classes
* Class details
* Instructors

**Example Scenarios**

**Student**

* Logs into Experience
* Sees:
  + Card showing today’s classes
  + Card with financial aid status
  + Notifications for holds on their account
* Clicks a card → views a full page with more class details

**Staff Member**

* Logs into Experience
* Sees:
  + Card showing pending approval requests
  + Card with recent notifications
* Accesses:
  + Audit logs page via the Application Menu

**Why Use Ellucian Experience?**

**Single login** → No need to remember multiple passwords  
**Personalized** → Users see only what they care about  
**Consolidated data** → All info in one place  
**Modern interface** → Clean, tile-based UI  
**Customizable** → Create your own cards and pages

**Quick Comparison: Cards vs Pages**

| **Feature** | **Cards** | **Pages** |
| --- | --- | --- |
| Display Location | Dashboard tiles | Full content area |
| Purpose | Quick snapshot or summary | In-depth detail, configuration, actions |
| Can link to | Experience pages, external URLs | Open directly or from cards or menu |
| Customization | Delivered, templated, SDK-built | Page Designer, SDK-built |

**In Short**

* **Ellucian Experience** = Single portal for all users
* Shows info via:
  + **Cards** → quick views
  + **Pages** → detailed views
* Powered by:
  + Ethos Integration
  + ERP systems
  + Custom apps

It’s a **central hub** that makes life easier for everyone in your institution!

**What Is the Provisioning Metrics Card?**

The **Provisioning Metrics card** is a **dashboard widget** in Ellucian Experience.

It’s designed for administrators who are using **Ethos User Provisioning (EUP)** — the system that moves user and group data from Ellucian systems (like Banner or Colleague) to external identity systems (like Azure AD, Okta, LDAP, etc.).

Think of this card as a **status report** right on your Experience home page, telling you how well EUP is working.

**Parts of the Card**

Let’s explain each numbered part from your description:

**1 — Card Title**

* The default name is **Provisioning Metrics.**
* Your admins can **rename it** if they want, for clarity or branding.

Example:

Instead of “Provisioning Metrics,” you could rename it to “User Provisioning Health.”

**2 — Card Controls**

On the card, you’ll see icons:

* **Bookmark icon** → Remove the card from your home page.
* **Vertical dots (⋮)** → Change the card’s position on the home page.

This lets admins customize how the dashboard looks.

**3 — Time Period Controls**

The card lets you see provisioning metrics for different time frames:

* **Today** → Last 24 hours
* **Last Week** → Last 7 days
* **Last 30 Days** → Last 30 days

When you switch between these options, the graphs and numbers update to show data for that specific period.

**4 — Metric Overview Data**

If there’s data for the time frame, the card shows:

* A **progress bar** with:
  + Green portion = successful provisioning requests
  + Red portion = failed provisioning requests
* A **percentage** of how many provisioning requests succeeded.

Example:

Green 80% → successful  
Red 20% → failed

**If there were no provisioning requests**, you’ll simply see a message or icon saying so.

**5 — View Breakdown Button**

This is a button on the card labeled **View Breakdown.**

When you click it:

* It takes you to the full **Ethos User Provisioning Metrics page.**
* There, you’ll see details like:
  + The exact time of each provisioning request
  + The data (payload) that was sent
  + Which requests succeeded or failed
* The data shown initially matches whatever time frame you selected on the card.

So it’s a **deeper dive** into what’s happening behind the scenes.

**Why This Card Matters**

This card helps administrators:

* Know whether user provisioning is running smoothly.
* Quickly detect problems (e.g. if a lot of requests are failing).
* Drill into details to **troubleshoot issues.**
* Confirm that new configurations are working as expected.

**Real-Life Example**

Let’s say:

* Your school recently onboarded new students.
* EUP sends their accounts to Azure AD.

An admin might:  
Look at the Provisioning Metrics card → Sees 95% success, 5% errors.  
Click **View Breakdown** → See which student accounts failed.  
Investigate the failures → Maybe a missing email address in Banner caused the problem.

This allows proactive **error fixing** so students can log in without issues.

**In Short:**

* The **Provisioning Metrics card** is your quick health-check for user provisioning.
* It summarizes how many provisioning jobs succeeded or failed.
* You can drill down for details if there are errors.
* It’s customizable for different time frames and dashboard layouts.

It’s basically your **“Mission Control”** for keeping EUP running smoothly!

Bottom of Form

**Ethos P1 Assessment**

Q1. How many applications can exists within an environment?

Ans. No limit

Q2. Match the following

Q3. Which of following terms refers to a code passed in by the programs calling an API to identify the calling program, its developer, or its user to the website.

Ans. API Key

Q4. Ethos Integration introduces an integration hub to overcome the challenges of point-to-point integrations.

Ans. True

Q5. Match the following

Q6. Which of the following systems can be integrated with Ellucian Ethos Integration?

Ans. All of the above

Q7. True or False.  
Ellucian Ethos Integration allows applications to integrate in real-time regardless of whether they are deployed on premises or in the cloud.

Ans. True

Q8. True or False.

The Ethos Integration home page displays all of the applications that have been added to that Ethos Integration environment.

Ans. False

Q9. True or False.  
Individuals with access to the administrative interface of Ellucian Ethos Integration can configure and monitor the flow of messages between integrated applications.

Ans. True

Q10. Where would you find resources owned by an individual application on the Ethos Integration Hub Interface?

Ans. Ellucian Ethos Integration User Interface Application tab > Application overview > Resources tab

Q11. Select the two responsibilities of an authoritative source application.

Ans. → Publish data changes

→ Respond to requests for data

Q12. Match the following

Q13. Select the data exchange methods in Ethos Integration.

Ans. → Publish and subscribe

→ Proxy API requests

→ GraphQL requests

Q14. Ellucian Ethos Integration uses which of the following concepts: (select all that apply)

Ans. → Amazon Web Services

→ RESTFUL APIs

→ Ellucian Ethos Data Model data changes

Q15. Each tenant initially contains how many Ethos Integration environments?

Ans. 2

**Lession -11 : Connect the Ellucian ERPs to Ethos Integration**

**Explanation of the Components and Message Flows**

Ellucian Ethos Integration enables two major integration patterns:

**① Request/Reply (Proxy) Messaging**

* **Who uses it?**  
  Non-authoritative applications that want to look up data on-demand from the authoritative ERP system (e.g. Banner or Colleague).
* **Flow (Blue arrows in the diagrams):**
  1. A non-authoritative app sends a **service invocation** (API request) to the **Proxy Service** in Ethos Integration.
  2. The Proxy Service forwards the request to the ERP’s exposed **Ethos Data Model APIs**.
  3. The ERP responds via the API.
  4. The Proxy Service sends the response back to the calling non-authoritative application.
* **Key takeaway:**  
  This is real-time, on-demand data retrieval.

A diagram of a computer program

AI-generated content may be incorrect.

**② Publish/Subscribe (Messaging) Flow**

* **Who uses it?**  
  Systems that want to be notified when changes happen in Banner/Colleague instead of polling for data.
* **Flow (Green arrows in the diagrams):**
  1. BEP (Banner Event Publisher) or EDX (Envision Data Exchange for Colleague) **detects database changes** and generates business events.
  2. These events are **published to the Ellucian Messaging Service (EMS)**.
  3. The **Ellucian Message Adapter (EMA)** reads the event from EMS.
  4. EMA calls the ERP’s **Ethos Data Model APIs** to retrieve the full data payload for the entity that changed (e.g. a student, course, section).
  5. EMA **publishes the full data payload to the Publisher Service** in Ethos Integration.
  6. The Publisher Service distributes the change notification into the **Message Queue Service (MQS)**, where subscribers (non-authoritative applications) can poll and retrieve those messages.
* **Key takeaway:**  
  This is event-driven integration to keep external systems synchronized with changes in Banner or Colleague.

A diagram of a software application

AI-generated content may be incorrect.

**Component Roles**

| **Component** | **Purpose** |
| --- | --- |
| **BEP / EDX** | Listens for DB changes and publishes generic business events into EMS. |
| **EMS (Messaging Service)** | Holds events until consumed, decoupling event generation from processing. |
| **EMA (Message Adapter)** | Consumes events from EMS, calls APIs to build full Ethos Data Model payloads, and publishes these to Ethos Integration’s Publisher Service. |
| **Ethos Data Model APIs** | Expose ERP data in a standard model for consumption by EMA or external apps via Proxy. |
| **Proxy Service** | Routes on-demand API calls from non-authoritative apps to the authoritative system and returns results. |
| **Publisher Service** | Receives published data changes and stores them for subscribers. |
| **Message Queue Service** | Holds published messages for non-authoritative apps to poll and retrieve. |
| **Error Reporting** | Captures and reports errors encountered during processing. |

A diagram of a software application

AI-generated content may be incorrect.

**Green Path (Publish/Subscribe)**

* **(1)** BEP/EDX detects a DB change and creates an event.
* **(2)** Event published to EMS.
* **(3)** EMA consumes the event from EMS.
* **(4)** EMA queries ERP APIs for full data.
* **(5)** EMA publishes the complete entity payload to Ethos Integration’s Publisher Service.
* **(6)** Non-authoritative apps poll MQS to retrieve messages.

**Blue Path (Request/Reply)**

* **(1)** Non-authoritative app sends API request to Proxy Service.
* **(2)** Proxy Service forwards to ERP APIs.
* **(3)** ERP APIs respond with data.
* **(4)** Proxy Service sends response back to calling app.

**Summary**

* **BEP/EDX + EMS + EMA** → Used for **event-driven publish/subscribe** integrations.
* **Proxy Service + APIs** → Used for **on-demand request/reply** integrations.
* **Publisher Service + MQS** → Distribute changes to non-authoritative apps in **publish/subscribe**.
* **Error Reporting** → Helps diagnose failures in any of these flows.

These integrations allow Ellucian institutions to **modernize, decouple, and scale integrations** across cloud and on-prem environments.

**In short:**

* Use **Proxy Service** for on-demand lookups (Request/Reply).
* Use **Publisher Service and MQS** for data change notifications (Publish/Subscribe).

**Banner Setup Checklist**

Banner can integrate with other applications through Ethos Integration in **three ways**:

1. **Proxy API requests**
   * Real-time requests from external apps → Proxy → Banner.
2. **GraphQL requests / Ethos Data Access**
   * GraphQL queries against the consolidated Ethos Data Model.
3. **Publish/Subscribe**
   * Banner publishes data changes → subscribers receive updates automatically.

Depending on how you want to integrate, you **don’t have to install everything** — only the components relevant to your chosen integration method.

**Checklist Steps Explained**

Let’s go through each step in your table.

**1. Prepare for Ethos Integration with Banner**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Overall initial steps like:
  + Ensuring correct Banner versions.
  + Reviewing prerequisites.
  + Understanding integration architecture.

Basically, making sure your environment is ready for Ethos Integration.

**2. Install Supporting Banner Software**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Installs core components used by Banner APIs.
* Includes packages like:
  + Banner API frameworks
  + Integration libraries
  + Utilities for handling data models

Essential foundation for any integration approach.

**3. Install the Ellucian Messaging Service (EMS) with Banner**

**Required for:**

* ✔ GraphQL
* ✔ Publish/Subscribe
* **Not needed for Proxy API only.**

EMS is:

* A **queue system** between Banner and integration layers.
* Makes sure events (like student updates) flow reliably and in order.

If you’re only doing Proxy API calls → you don’t need this queueing layer.

**4. Install the Banner Event Publisher (BEP)**

**Required for:**

* ✔ GraphQL
* ✔ Publish/Subscribe
* **Not needed for Proxy API only.**

BEP:

* Detects changes in Banner’s database (e.g. new student).
* Sends a **business event** to EMS.

Without BEP, there’s no automatic “push” of changes from Banner.

**5. Install the Banner Ethos API DB Upgrade**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Database changes needed for:
  + Storing GUIDs
  + Supporting Ethos Data Model
* Prepares the Banner database for integrations.

This upgrade is mandatory regardless of the integration type.

**6. Set up the Banner Ethos APIs and Supporting Software**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Deploys Banner’s Ethos APIs.
* Makes Banner data available in:
  + REST APIs
  + GraphQL
  + Publish/Subscribe processes

No APIs = no integration.

**7. Open Ports in the Firewall**

**Required for:** ✔ Proxy API | ✔ Publish/Subscribe

* Allows communication between:
  + Banner APIs
  + Ethos Integration components
  + External apps

Not strictly needed for GraphQL Data Access if everything is internal.

**8. Create a Banner Ethos API User for Ethos Integration**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* A dedicated user account:
  + Secures API calls
  + Controls permissions

Every integration must authenticate somehow!

**9. Generate Banner GUIDs**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Ethos uses **GUIDs** to uniquely identify records across systems.
* Must generate GUIDs for existing Banner data.

Example:

* A student in Banner = one unique GUID across all systems.

**10. Specify API Configuration Settings in Banner**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Settings like:
  + Base URLs
  + Authentication methods
  + Logging levels

Fine-tunes how Banner APIs work.

**11. Set up Banner in Ethos Integration**

**Required for:** ✔ Proxy API | ✔ GraphQL | ✔ Publish/Subscribe

* Registers Banner as an authoritative source in Ethos Integration.
* Maps Banner APIs to the Ethos Data Model.

Example:

Tells Ethos Integration:  
*“Banner is authoritative for student data.”*

**12. Set up Banner with Data Access**

**Required for:** ✔ GraphQL only

* Connects Banner’s APIs to **Ethos Data Access.**
* Enables GraphQL queries against Ethos Data Model entities stored in Banner.

If you’re not using GraphQL, skip this step.

**13. Specify EMS Settings in Banner**

**Required for:** ✔ GraphQL | ✔ Publish/Subscribe

* Connects Banner to the Ellucian Messaging Service.
* Defines:
  + Queue names
  + Connection info
  + Message formats

No EMS settings needed if you only do direct Proxy API calls.

**14. Configure and Deploy the Ellucian Message Adapter (EMA) with Banner**

**Required for:** ✔ GraphQL | ✔ Publish/Subscribe

EMA:

* Consumes events from EMS.
* Calls Banner APIs to fetch the full data.
* Publishes the full Ethos Data Model message to Ethos Integration.

If you only want Proxy calls → you don’t need EMA.

**So, What Do You Actually Need?**

→ **If you only want Proxy API calls:**

Prepare  
Install Banner software  
Ethos DB upgrade  
Banner APIs  
Open firewall ports  
Ethos API user  
Generate GUIDs  
API configuration  
Set up in Ethos Integration

→ **If you want Publish/Subscribe:**

All the above, plus:  
EMS install  
BEP install  
EMS config  
Deploy EMA

→ **If you want GraphQL:**

All the above, plus:  
Set up Banner with Data Access

**Why Different Requirements?**

* **Proxy API** is a simple “real-time request → get answer” pattern.
* **Publish/Subscribe** needs:
  + Events → queues → adapters → publishing.
* **GraphQL** sits on top of the Ethos Data Model and may also use queues for consistency.

Hence, the additional components.

**Big Picture**

* Banner → authoritative system
* Ethos Integration → bridge to other apps
* Checklist ensures:
  + Security
  + Performance
  + Standard data model

**In short:**  
This checklist guides you through setting up the right mix of software and configurations so Banner can share data through Ellucian Ethos Integration, either via real-time APIs, GraphQL, or publish/subscribe events.

Let’s discuss **key differences between an on-premise Banner environment and Banner SaaS**, and the implications for migrating from on-prem to SaaS.

**Key Area #1 — Custom Code and Database Changes**

**What you cannot do in Banner SaaS:**

* **Custom code** you’ve written in your on-prem Banner system **cannot be moved** into SaaS.
* You **cannot**:
  + Add custom tables or columns directly to the Banner database (except via approved mechanisms like SDE).
  + Change baseline tables—for example:
    - making a column longer or wider
    - changing a data type
* Essentially:

**No direct modifications** to Ellucian’s SaaS database structures or core code.

**How to check for customizations:**

Before migrating, you need to identify:

* Where you have custom database objects
* Where you’ve altered Banner’s delivered tables
* Where custom code exists (e.g. PL/SQL jobs)

**Tool to help:**  
**Banner Environment Comparison Tool (BECT)**

* Compares:
  + your database schema
  + job submission code
* against:
  + baseline (standard) Banner SaaS schemas and code
* Detects:
  + Custom objects
  + Code changes
* Works using hash signatures (not requiring a live DB connection) for efficiency.

**Why this matters:**  
Anything customized needs to be:

* removed
* replaced with supported extensibility features
* or reimplemented using SaaS-compliant tools

**What you can do in Banner SaaS: Supported extensibility**

While SaaS restricts custom coding, **Ellucian provides tools for safe extensions** so you can still tailor the system:

**Extensibility Options:**

1. **Experience SDK**
   * Create custom pages or UI extensions (cards, tiles)
   * Control layout, navigation, etc.
2. **Data Connect and Ethos Integration**
   * Connect Banner with other applications.
   * Capture business events, run logic externally, and update records via APIs.
3. **Schema Extensibility**
   * Extend existing tables **safely**.
   * Add new tables via Ellucian’s supported mechanisms.
4. **Banner Event Publisher (BEP)**
   * Publish business events for **your custom schemas** as well.
   * Example: new table in SaaS triggers an event for integrations.
5. **Text Manager**
   * Manage multilingual labels, error messages, UI strings.
   * Support localization and translations.

The rule in SaaS is:

No touching the underlying database or core code directly — but **safe ways exist** to extend the product.

**Key Area #2 — Single Sign-On (SSO) and Email Configuration**

When moving to SaaS, Ellucian needs to:

* Integrate Banner with your organization’s identity system (e.g. Azure AD, Okta, etc.).
* Set up email services so Banner SaaS can send notifications, financial aid communications, etc.

**SSO setup**

* **Ellucian will reach out** during migration planning.
* They’ll gather:
  + Identity provider details (e.g. IdP URLs, certificates)
  + Attributes to pass for user authentication

This enables seamless login to SaaS using your existing accounts.

**Email Configuration**

* Ellucian configures SaaS to connect to your email server.
* Ensures:
  + Banner Financial Aid
  + Banner Communication Management  
    can send emails.

E.g. award letters, reminders, official notifications will still flow through your preferred mail service.

**Why These Differences Matter**

When migrating to SaaS:

* You **cannot simply “lift and shift”** your existing Banner environment.
* Any custom:
  + database objects
  + code
* must be **reviewed and re-engineered** using Ellucian’s SaaS-compliant extensibility tools.
* Authentication and email integrations **must be coordinated** with Ellucian for SaaS to function smoothly.

**In summary:**

→ You can’t carry over custom code or database changes.  
→ Instead, you must use Ellucian’s supported extension tools.  
→ SSO and email require coordination to work properly in SaaS.

**Lesson – 12: Create Authoritative and Non-Authoritative Applications in Ethos Integration**

**Resource Ownership in Ethos Integration**

When we talk about a **resource** in Ethos Integration, we mean a specific type of data entity — for example:

* **Persons**
* **Courses**
* **Sections**
* **Accounts**
* **Addresses**

These are the “data models” (resources) defined in the **Ellucian Ethos Data Model (EDM).**

**Ownership** means:

Which application is considered the “authoritative source” for the data in that resource.

**Scenario 1 — Single Owner**

**Description:**

* Only **one application** owns a particular resource.
* That application is the **authoritative system of record** for that resource’s data.

**Example:**

* Banner is the authoritative source for **courses.**
* Colleague is the authoritative source for **persons.**

When another application needs data about courses, Ethos Integration:

1. Receives the request.
2. Knows **exactly which app to forward the request to** (e.g. Banner).
3. Returns the result to the requesting application.

Other systems can:

* **Subscribe** to changes in that resource.
* **Request data** via API (proxy) or GraphQL.

But only one system **owns** it.

**Benefits:**  
Simpler architecture.  
No ambiguity about which system has the “truth.”  
Fewer configuration steps.

**Scenario 2 — Multiple Owners**

**Description:**

* **More than one application** owns the same type of resource.
* Each application owns its **own instance** of that resource.

**Why would this happen?**

→ Large institutions often have:

* **Multiple campuses** with separate ERP instances
* Parallel systems managing overlapping data domains
* Divided administrative functions (e.g. finance vs. academics)

**Example:**

* Both Banner at Campus A and Banner at Campus B own **courses.**
* Each campus has a separate catalog, different schedules, different instructors.

So, there are **multiple “instances” of the Courses resource.**

**How does Ethos handle it?**

When multiple apps own the same resource:

* Any app requesting data must **specify which instance it wants.**

Otherwise, Ethos wouldn’t know **which app to query.**

This applies to:  
Subscriptions  
Proxy API requests  
GraphQL queries

**Example Scenario**

Imagine:

* Banner (Main Campus) owns “Courses.”
* Banner (Satellite Campus) also owns “Courses.”

→ Both publish changes to the **Courses resource.**

If the LMS wants the courses from the Satellite Campus, it must:

**Specify the owning application instance** (e.g. the Satellite Banner environment) in the request.

Otherwise, Ethos has no way to know which campus’s courses to return.

**Benefits of Multiple Owners**

* Supports **multi-campus institutions.**
* Allows different departments or systems to manage their own data.
* Provides **data isolation** where needed.

But:

* Adds configuration complexity because you must **define which owner’s data** you want when requesting resources.

**Where This Matters**

The **main difference** is:

| **Scenario** | **Must specify owner when requesting resource?** |
| --- | --- |
| Single owner | ❌ No |
| Multiple owners | ✅ Yes |

**Short Version**

**Single Owner** → Only one app owns the resource → Simple integration.

**Multiple Owners** → Multiple apps own the same resource → You must specify **which one’s data** you want.

**TL;DR:**

In Ethos Integration, a resource can be owned by one app (simple) or multiple apps (more complex). With multiple owners, you must tell Ethos which app’s data you want when you subscribe, call APIs, or run GraphQL queries.

**Browsing Catalogs in Ethos Integration**

When you work in **Ellucian Ethos Integration**, you often connect applications to your Ethos environment. To make that easier and more standardized, **Ellucian provides catalogs.**

Think of a **catalog** like a **blueprint or template** for an application that wants to integrate with Ethos. It defines:

* The resources the app publishes or consumes (e.g. persons, courses, addresses)
* API endpoints
* Message types
* Supported versions

Instead of creating all this manually for every app, you can **browse the catalogs and choose one as a starting point.**

**Procedure Summary**

**Step 1 — Open the Catalog List**

**Go to: Applications → From Catalog**

* Log into the **Ethos Integration User Interface.**
* Go to the **Applications page.**
* Click the button **“From Catalog.”**

**Result:**

* You see a page listing all **available catalogs**:
  + Catalog names
  + High-level details (like the resources included)
  + Expiration date of the catalog (so you know if it’s current)

**Note:**  
If a catalog has expired, it **does NOT appear in this list.** Only active catalogs are displayed.

**Browse Catalog Window — What Are You Seeing?**

When you’re in the **Browse Catalog** window, you see:

* A list of catalogs
* Brief summaries of each
* Possibly buttons or icons to view details

You can usually:

* **Search** for a catalog
* Filter by app type
* See version info
* Read descriptions of what the catalog contains

Some interfaces provide an **ⓘ icon** (info icon) for each catalog. Clicking it will show:

* Which resources the app manages
* Supported API versions
* Specific endpoints

This helps you pick the **correct template** for your integration.

**Step 2 — Create an Application From a Catalog**

Optional but highly useful.

* Once you’ve found a catalog that matches your application’s purpose, click:

**Create App**

**What happens next?**

* Ethos creates an **application entry** using the catalog as a template.
* Pre-fills configuration:
  + Resources owned
  + Supported operations
  + Data model mappings
* Saves time versus configuring an app from scratch.

After that, you can:  
- Modify the application settings.  
- Add authentication details.  
- Configure messaging options.

**Why Use Catalogs?**

Using catalogs:

* Ensures **consistency** in how apps are integrated
* Helps you follow Ellucian best practices
* Reduces **setup time**
* Lowers the risk of configuration errors
* Supports standard resource definitions and versions

**Documentation Reference**

If you want to go deeper, check:

**Manage applications in Ethos Integration (single owner) (Internal-CE)**

This resource explains:

* How to create and manage apps
* How catalogs fit into the app setup process
* Detailed steps for using catalog templates

**Short Version**

* Go to **Applications → From Catalog** to browse integration templates (catalogs).
* Click **Create App** to start building an application using a catalog’s predefined settings.
* Saves you time and ensures consistent, correct setups for applications integrating via Ethos.

**How to View Catalog Details**

Here’s how to **view the details of available catalogs** in the Ethos Integration UI.

**Step 1 — Go to From Catalog**

* Log into the Ethos Integration User Interface.
* Go to the **Applications page.**
* Click:

**From Catalog**

This opens the list of all **available catalog templates.**

**Step 2 — Click a Catalog Application Name**

* In the list of catalogs, find the catalog you’re interested in.
* Click the **Catalog Application name.**

**Result:**

* You’ll see **detailed information** about that specific catalog template:
  + What it integrates
  + The resources it uses
  + How it connects to Ethos (Proxy, GraphQL, etc.)

This helps you decide whether to use it for your integration project.

**Step 3 — Optional: Create App**

* If you want to **build an app based on the catalog**, click:

**Create App**

This:  
Automatically sets up a new application  
Pre-fills it with the catalog’s resources, connections, and settings  
Saves time compared to starting from scratch

**Understanding the Catalog Details**

When you view a catalog, you’ll see several fields:

**Catalog Application**

* The **name** of the catalog application.
* E.g. “Banner Student API”

**Version**

* The **version number** of the catalog.
* Helps ensure you’re using the correct data model.

**Partner**

* Tells you whether this catalog is for an **Ellucian Partner application.**
* Helps you identify third-party vendor integrations.

**Integration Type**

* Whether the integration is:
  + **One-way** → data flows in a single direction (e.g. from app to Ethos)
  + **Two-way** → data flows both ways between app and Ethos

**Permissions & Resource Access**

You’ll see a grid showing:

* The resources this app uses (like persons, addresses, courses, etc.)
* Checkmarks indicating how the app interacts with each resource:

| **Type** | **Meaning** |
| --- | --- |
| **Owned** | App is the **authoritative source** for this resource. |
| **Proxy** | App needs Ethos to forward API requests to it. |
| **GraphQL** | App uses GraphQL Data Access for this resource. |
| **Subscription** | App subscribes to **change notifications** for this resource. |

This lets you quickly understand:  
What data the app controls  
How it integrates with Ethos  
Whether it publishes or just consumes data

**Example:**

Imagine a catalog shows this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resource** | **Owned** | **Proxy** | **GraphQL** | **Subscription** |
| persons | ✅ |  | ✅ | ✅ |
| courses |  | ✅ |  |  |

* This means:
  + The app **owns “persons”** data → it’s authoritative for person records.
  + Other apps might use Proxy or GraphQL to get person data from it.
  + The app **does not own courses** but needs Proxy access to them.

**Why This Matters**

Saves time configuring new applications  
Helps you avoid errors or incomplete integrations  
Ensures you’re aligning with Ethos best practices  
Helps manage complex integrations across multiple systems

**Short Version**

* Go to **Applications → From Catalog.**
* Click a catalog name → view its details:
  + Resources
  + Permissions
  + Integration types
* Optionally, click **Create App** to build an app from the catalog.

**Steps to Create an Authoritative Application from Catalog:**

**Step 1: Go to the Applications Page**

* On the **Ethos Integration header bar**, click:

**Applications**

This takes you to the list of apps already in your environment.

**Step 2: Select a Catalog**

* In the **Create New App** area, click:

**From Catalog**

* Then find your desired application and click:

**Create App**

Tip: The "Browse Catalog" view lets you search for apps supported by Ellucian, like Banner, Colleague, etc.

**Step 3: Enter Application Details**

You're now on the **Add Applications page**. Fill in:

* **Application name** (Required)  
  Give it a clear, unique name so you can easily recognize it later.
* **Description** (Optional)  
  Use this for extra clarity (e.g., “Authoritative app for student records”).

Click **Next** to proceed.

**Step 4: Define Connection Info**

Enter the technical details so Ethos can **talk to the app**:

* **Base URI**  
  The root API URL of the application (where Ethos will send calls).
* **Base QAPI URI** (optional)  
  Used **if your app supports POST-based querying**.
* **Username & Password**  
  Used for authenticating with the app’s API, so Ethos can **retrieve resources** from it.

Click **Next** once complete.

**Step 5: Configure GraphQL & Resources**

If your catalog has GraphQL-enabled resources:

* They will show up under the **GraphQL Resources** tab.
* By default, the **first app** added will be the owner.
* If resources are **unowned**, a warning will show. You can fix this by assigning ownership manually (click **Edit 🖉**).

**Optional: Add More Resources**

You can click **Add Resources** to include other entities this app should reference.

To **restrict access to specific fields** inside a resource (like hiding SSNs or grades), refer to the "Grant access to resource configurations..." documentation.

Click **Next** to proceed.

**Step 6: Set Up Subscriptions (Optional)**

You’ll see the **Add Subscriptions** page.

* You can subscribe this app to changes in resources owned by other apps.
* This is helpful for **data syncing** or triggering downstream workflows.

Click **Next** when done.

**Step 7: Confirmation**

* You’ll see a success message:

“Your application has been successfully added and configured.”

You can then:

* Click **View Application** → opens the full **Application Overview**
* OR click **Back to Applications** → returns you to the full list

**Summary**

| **Step** | **What You Do** | **Why It Matters** |
| --- | --- | --- |
| 1 | Open Applications | Start the creation process |
| 2 | Pick Catalog → Create App | Use a supported, pre-defined template |
| 3 | Name + Describe App | Identify it uniquely in your environment |
| 4 | Base URIs + Credentials | Let Ethos connect to the app |
| 5 | Manage Resources & GraphQL | Set ownership, add or hide fields |
| 6 | Add Subscriptions | Get updates from other apps |
| 7 | Confirm & Review | Finish and review the app’s config |

**How to Associate an Application to a Catalog**

Here’s the step-by-step explanation in simple terms:

**Step 1 — Open Your Application**

* In the Ethos Integration UI, click:

**Applications** (on the Ethos Integration header bar)

* Find and click on the name of your application (the one you want to associate).
* Look at the upper right section of the page:
  + If your app **is not associated** with a catalog, it will show something like:

“Not associated with a catalog application.”

* Click the dropdown called:

**Pick Catalog to Associate**

This dropdown lets you search and select from available catalogs.

**Step 2 — Pick the Catalog Application**

* From the dropdown list:
  + Find the appropriate **catalog application** that matches your app.
  + For example, if you’re working with Banner Student, pick the official **Banner Student** catalog.

Why pick a catalog?

* It defines which resources your app owns or can access.
* It ensures your app matches Ellucian’s standards.
* It enables better support from Ellucian.

**Step 3 — Save the Association**

* After selecting the catalog, click the:

**Save** icon (usually a small disk or checkmark)

This saves the connection between your app and the chosen catalog.

**Step 4 — Confirmation**

* Once saved:
  + The name of your selected catalog will appear under:

**Application Catalog**

Now your app is officially associated with the catalog!

**Why This Matters**

**Support and Maintenance**  
Ellucian can quickly see which apps you’re running and help troubleshoot issues.

**Integration Clarity**  
Catalogs document the resources an app owns or accesses. This prevents conflicts or confusion in integrations.

**Future Upgrades**  
Catalog associations help Ellucian guide you on upgrades or changes to resource structures.

**Optional — Disable the Prompt**

If you don’t want to be reminded every time:

* Toggle:

**Do not show this prompt for all applications**

But be careful — associating apps to catalogs is highly recommended for proper integration management.

**Example**

Imagine you have a custom integration called “My Student App,” but it’s really an implementation of Banner Student. You should associate “My Student App” to the official **Banner Student** catalog so:  
Ellucian knows what resources it uses.  
You avoid redundant setup.  
You get proper Ellucian support for resource changes.

**Summary Table**

| **Step** | **What you do** | **Why** |
| --- | --- | --- |
| 1 | Open the application’s page | Access the catalog dropdown |
| 2 | Pick catalog app | Link your app to the official definition |
| 3 | Save | Commit the association |
| 4 | Verify catalog name displays | Confirms the link is successful |

**How to Manually Create an Application**

**Step 1 — Start Manual Creation**

* In the Ethos Integration UI:
  + Click:

**Applications** (top navigation bar)

* Look for the section:

**Create new App**

* Click the button:

**Manually**

This tells Ethos Integration you’re not starting from a predefined catalog.

**Step 2 — Skip API Configuration (for this demo)**

* For this demonstration, you **do not need to configure REST API Proxy or GraphQL resources.**

Normally, when creating an app:  
You’d define:

* REST API Proxy (if the app will handle API requests via Ethos)
* GraphQL resources (if using Data Access)

But this example skips those to keep it simple.

**Step 3 — Enter Application Details**

* Enter:
  + **Application Name** → A unique name so you can recognize it in the UI.
  + **Description** → Optional notes describing the purpose or details of the application.

Then click:

**Next**

Example:

Application Name: My Custom Integration

Description: A custom integration for testing Ethos publishing.

**Step 4 — Optional Subscriptions**

* You’ll see a page for **Add Subscriptions.**
  + Subscriptions let your app receive change notifications from Ethos Integration for specific resources (e.g. Person, CourseSection).
* For this demo, **no subscriptions are added.**

However, if you were setting up a real integration, you might:  
Click **Add Subscriptions**  
Choose resources your app wants to listen for changes in.

Click:

**Next**

**Step 5 — Confirmation**

* Ethos Integration confirms your new application has been successfully created.

You can choose:  
**View Application** → Opens the Application Overview page, showing:

* Application details
* Resources it owns or proxies
* Subscriptions it has  
  **Back to Applications** → Returns you to the list of all apps in the system.

**Why You’d Manually Create an Application**

* **Custom apps**
  + E.g. an in-house reporting tool that will subscribe to Ethos data.
* **Prototyping**
  + Test integrations without impacting production systems.
* **Third-party apps** not in Ellucian’s catalogs
  + E.g. integrating niche products unique to your institution.

**Quick Recap Table**

| **Step** | **What you do** | **Why** |
| --- | --- | --- |
| 1 | Click **Manually** under Create new App | Start manual app creation |
| 2 | Skip API config (for demo) | Keep it simple |
| 3 | Enter app name & description | Identify your app |
| 4 | Optionally add subscriptions | Choose resources to monitor |
| 5 | View success message | Confirm setup |

**Example Outcome**

After this procedure, your new app appears in Ethos Integration’s Applications list. It can later:

* Be configured for REST or GraphQL access
* Subscribe to resources
* Act as an authoritative source

**What Does “Owning a Resource” Mean?**

In Ethos Integration:

* A resource = A specific type of data (e.g. person, student, housing-assignment).
* Owning a resource means:
  + The application **is authoritative** for that data.
  + It’s the **system of record** (i.e. the “truth”).
  + It **publishes data changes** for that resource.
  + It can respond to API calls for that data.

Example:

Colleague might own “housing-assignments” because it’s where the official housing data lives.

**Don’t mark an app as owning a resource unless it’s truly authoritative.**  
If it just reads the data, it should subscribe instead.

**Why You Might Add Owned Resources**

Two reasons:

1. **Catalog import** (automatic)
   * If you create an app from the catalog (e.g. official Colleague app), all owned resources can be automatically loaded.
2. **Manual addition**
   * If your application is custom or only authoritative for specific resources, you’ll add them individually.

**Types of Authentication for Proxy API Requests**

If one app makes a proxy API request to another, you need credentials.

**Two types exist:**

**➤ Basic Authentication**

* Username + password of a user in the target system.
* Example: Proxy API calls to Banner or Colleague.

**➤ User-defined Authentication**

* An API key or token instead of a username/password.
* Example: Many modern APIs prefer tokens for security.

**Scenario Example**

Your campus card system wants to look up housing assignments in Colleague.  
Colleague owns the housing-assignment resource.  
You:

* Create a user in Colleague with READ access.
* Save those credentials in Ethos Integration.
* Now the campus card system can safely query Colleague through Ethos.

**Adding Owned Resources in Ethos Integration**

**How to add owned resources for an authoritative application:**

**Step 1 — Open the Application**

* Go to:

Applications tab

* Click the **Application Name** for the app that will own resources.

**Step 2 — Open Owned Resources Tab**

* Click:

**Owned Resources Tab**

Here you’ll see:

* Any resources the app currently owns.
* Options to add more.

**Step 3 — Autoconfigure Resources (optional)**

Instead of adding resources one by one, you can let Ethos Integration **auto-configure** them.

* Click:

**Autoconfigure Resources**

* Ethos Integration connects to the app’s API and:
  + Detects what resources the app exposes.
  + Adds them as owned resources.

**Credentials Required!**

* You’ll enter a username/password to allow Ethos to connect to the app’s API.

**Step 4 — Manually Add Resources**

Instead of autoconfigure, you can add resources yourself:

* Click:

**Add Resources**

Then choose:  
**Add Ellucian Ethos Resource**  
**Add Custom Resource**

**Step 5 — Add Ellucian Ethos Resource**

* Click:

**Add an Ellucian Ethos Resource**

* Either:
  + Select all resources
  + OR choose resources individually

Click:

**Add**

**Step 6 — Add Custom Resource**

If your application owns a custom entity:

* Click:

Add A Custom Resource

In the dialog box:

* Enter:
  + Resource Name → e.g. campus-events
  + URI Override (optional)
  + QAPI URI Override (optional)

**Why Override the URI?**

* Normally:

http://baseuri/resource-name

* But if your custom resource lives somewhere else, you override the default.

Example:

http://baseuri/custom/persons

Click:

**Add**

**Step 7 — Confirm Resources Added**

Once resources are added, you’ll see them listed:

* Resource name (e.g. employee-leave-plans)
* Whether it’s an Ellucian Ethos Data Model
* Version (semantic versioning, e.g. v1.0)
* Deprecated versions marked with \*

**How It All Comes Together**

**Imagine this scenario:**

You have Colleague managing housing data.  
A campus card system needs to know housing assignments.  
In Ethos:

* Colleague **owns** the housing-assignments resource.
* Campus card system makes **proxy API requests** to Colleague.
* Authentication is configured (username/password or token).

→ This is why you define owned resources—to clarify:

* Who’s authoritative
* Who publishes data
* Who can be queried for it

**In Summary**

→ **Only add a resource as “owned” if the app is authoritative.**  
→ Use autoconfigure for convenience, or add resources manually for precise control.  
→ Set up authentication if you’ll be making proxy API calls.  
→ Resources you add appear under the Owned Resources tab.

**Creating a Proxy API Consumer Application**

**Use case:**  
An app wants to read or change data in Banner or Colleague **live** via API calls.

**Steps to Create It in Ethos Integration UI**

**➤ Step 1 — Start App Creation**

* Go to:

Applications tab → Create New App → Manually

* Select:  
  **Configure REST API proxy**
* Click:  
  → **Continue**

**➤ Step 2 — Enter Application Details**

* Enter:
  + **Application Name**
  + (Optional) Description
* Click:  
  → **Next**

**➤ Step 3 — Add Source Applications**

This is the critical step:

* Click:  
  → **Add Source Application**
* Select the **authoritative application** (e.g. Banner or Colleague).
* Enter the **authentication credentials** needed to access the APIs in the authoritative app.
  + Could be:
    - Username/password (Basic Auth)
    - Token/API key (User-defined Auth)

Click:  
→ **Add**

**➤ Step 4 — Configure Subscriptions (Optional)**

* You could subscribe to change notifications for resources.
* In this example, skip it.
* Click:  
  → **Skip**

**➤ Step 5 — Finish**

* Ethos shows a **confirmation message**.
* An **API key** is generated for your proxy consumer app.
* You can:
  + Go to **Application Overview**
  + Or return to Applications list

**End Result:**

* Your new application can now make **proxy API calls** through Ethos Integration.
* You don’t have to code directly to multiple back-end systems—it’s abstracted by the Proxy Service.

**Creating a Subscriber Application**

**Use case:**  
An app wants to **receive notifications** when data changes in Banner or Colleague.

**Steps to Create It in Ethos Integration UI**

**➤ Step 1 — Start App Creation**

* Go to:

Applications tab → Create New App → Manually

* Ignore all checkboxes (because it’s not a proxy app).
* Click:  
  → **Continue**

**➤ Step 2 — Configure App Details**

* Enter:
  + **Name** (required)
  + **Description** (optional)
* Add **Subscriptions**:
  + Click “Add Subscriptions.”
  + Choose resources (e.g. students, housing-assignments, etc.).
  + For each resource, you’re telling Ethos Integration:

“I want notifications any time this resource changes.”

Click:  
→ **Next**

**➤ Step 3 — Finish**

* A **confirmation message** appears.
* Ethos generates an **API Key** for your subscriber app.
* You can:
  + Go to **Application Overview**
  + Or return to Applications list

**How It Works at Runtime**

* Banner or Colleague detects a data change → publishes an event (BEP or EDX).
* EMS → EMA fetches the data → sends it to Ethos Publisher Service.
* Ethos places the change into a **queue** for your subscriber app.
* Your app **polls the Message Queue Service (MQS)** to retrieve events.

**Creating a Proxy API Consumer Application**

**Use case:**  
An app wants to read or change data in Banner or Colleague **live** via API calls.

**Steps to Create It in Ethos Integration UI**

**➤ Step 1 — Start App Creation**

* Go to:

sql

CopyEdit

Applications tab → Create New App → Manually

* Select:  
  ✅ **Configure REST API proxy**
* Click:  
  → **Continue**

**➤ Step 2 — Enter Application Details**

* Enter:
  + **Application Name**
  + (Optional) Description
* Click:  
  → **Next**

**➤ Step 3 — Add Source Applications**

This is the critical step:

* Click:  
  → **Add Source Application**
* Select the **authoritative application** (e.g. Banner or Colleague).
* Enter the **authentication credentials** needed to access the APIs in the authoritative app.
  + Could be:
    - Username/password (Basic Auth)
    - Token/API key (User-defined Auth)

Click:  
→ **Add**

**➤ Step 4 — Configure Subscriptions (Optional)**

* You could subscribe to change notifications for resources.
* In this example, skip it.
* Click:  
  → **Skip**

**➤ Step 5 — Finish**

* Ethos shows a **confirmation message**.
* An **API key** is generated for your proxy consumer app.
* You can:
  + Go to **Application Overview**
  + Or return to Applications list

**End Result:**

* Your new application can now make **proxy API calls** through Ethos Integration.
* You don’t have to code directly to multiple back-end systems—it’s abstracted by the Proxy Service.

**Creating a Subscriber Application**

**Use case:**  
An app wants to **receive notifications** when data changes in Banner or Colleague.

**Steps to Create It in Ethos Integration UI**

**➤ Step 1 — Start App Creation**

* Go to:

Applications tab → Create New App → Manually

* Ignore all checkboxes (because it’s not a proxy app).
* Click:  
  → **Continue**

**➤ Step 2 — Configure App Details**

* Enter:
  + **Name** (required)
  + **Description** (optional)
* Add **Subscriptions**:
  + Click “Add Subscriptions.”
  + Choose resources (e.g. students, housing-assignments, etc.).
  + For each resource, you’re telling Ethos Integration:

“I want notifications any time this resource changes.”

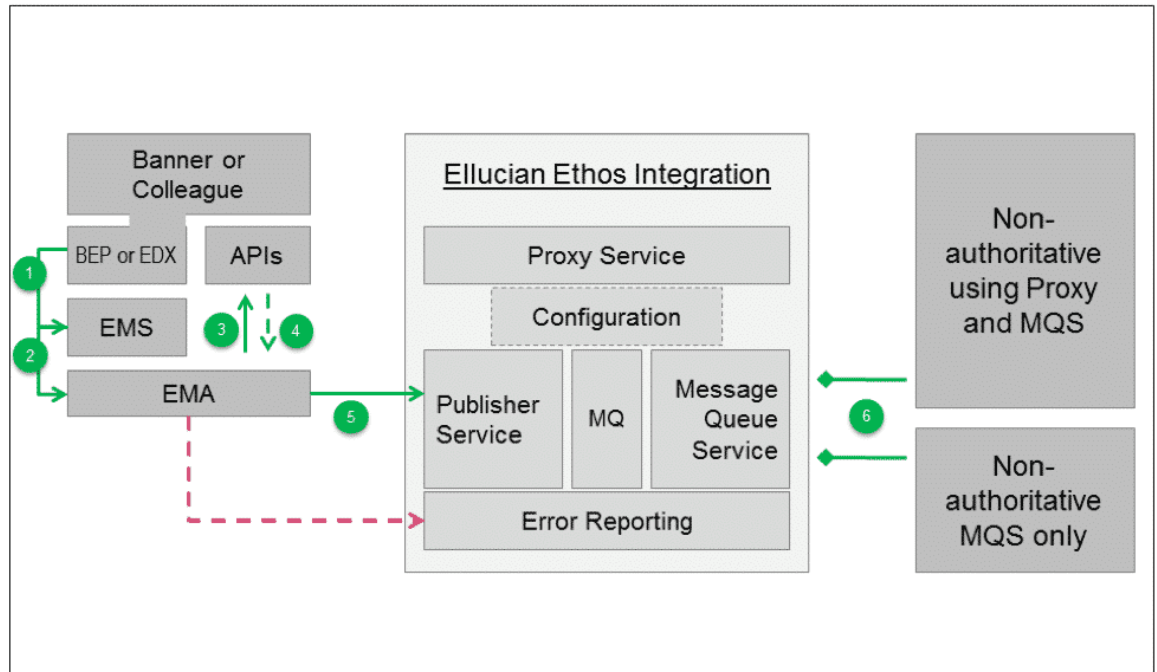
Click:  
→ **Next**

**➤ Step 3 — Finish**

* A **confirmation message** appears.
* Ethos generates an **API Key** for your subscriber app.
* You can:
  + Go to **Application Overview**
  + Or return to Applications list

**How It Works at Runtime**

* Banner or Colleague detects a data change → publishes an event (BEP or EDX).
* EMS → EMA fetches the data → sends it to Ethos Publisher Service.
* Ethos places the change into a **queue** for your subscriber app.
* Your app **polls the Message Queue Service (MQS)** to retrieve events.



1. **BEP/EDX** (Banner or Colleague’s event publisher) detects a change.
2. Sends an event to **EMS** (Ellucian Messaging Service).
3. **EMA** (Ellucian Message Adapter) consumes the event.
4. EMA calls APIs to fetch full entity data.
5. EMA sends that data to Ethos Integration → Publisher Service queues it.
6. Subscriber apps retrieve the message from Ethos MQS.

**Quick Comparison**

| **Feature** | **Proxy API Consumer** | **Subscriber App** |
| --- | --- | --- |
| Used for? | Real-time API calls (GET, POST, PUT, DELETE) | Event-driven data sync |
| Messaging Pattern | Request / Reply | Publish / Subscribe |
| Data ownership | Reads from authoritative apps | Responds to published changes |
| Example | Campus card system reads housing info | Reporting tool subscribes to student changes |

**Why Both?**

* Use **Proxy API** if:
  + You need on-demand data.
  + You want to change data via APIs.
* Use **Subscriber** if:
  + You want **notifications of changes** without polling APIs constantly.
  + You want to sync data efficiently.

Many integrations **combine both** for:  
→ Real-time reads/writes (Proxy)  
→ Change notifications (Subscription)

**In short:**  
Proxy = real-time access  
Subscriber = get updates as data changes

**Ellucian Ethos Data Access & GraphQL Integration — Explained**

**💡 What is Ethos Data Access?**

**Ellucian Ethos Data Access** bridges data from different systems (like SIS, ERP, CRM, or other data sources) into a single cloud-based data store. It:

* **Extracts data** from authoritative systems (e.g. Banner, Colleague).
* Stores data in secure, multi-tenant cloud storage.
* Makes that data available in a standardized, real-time format via **GraphQL**.

**Why it matters:**  
Instead of connecting directly to many systems, apps can integrate once with Ethos Data Access, reducing complexity and enabling faster data-driven solutions.

**How Data Moves Through Ethos**

A diagram of a network

AI-generated content may be incorrect.

1. **Authoritative Data** (e.g. SIS, ERP, CRM) feeds into Ethos Data Access.
2. Data flows through **Ethos Integration** — the hub for managing APIs, proxies, messaging, etc.
3. Applications query the data via **GraphQL** for bulk reads or specific data sets.

So instead of hitting multiple backend systems, integrations funnel through Ethos for a unified experience.

**Data Loading & Updates**

Ethos Data Access keeps data current by:

**Loading** initial data from source systems.  
**Updating** when source systems send change notifications.  
**Reloading** data if the data model version changes.

This ensures Ethos always reflects near real-time data from source systems.

**GraphQL Access via Ethos Integration**

Instead of many REST calls, GraphQL enables:

* **Bulk reads** (pulling large sets of data in one call).
* Flexible queries tailored to exactly what you want.

Your second screenshot shows how Ethos Integration lets you **manage which GraphQL resources** each application can use.

For example:

* The academic-periods resource is LOADED and ready for queries.
* The buildings resource is NOT LOADED.
* The persons resource FAILED its load and needs investigation.

**GraphQL Resource Status Meanings**

From your chart:

| **Status** | **Meaning** | **What to do** |
| --- | --- | --- |
| **LOADED** | Data is loaded and available for GraphQL. | Ready for queries. |
| **LOADING** | Data is still loading into Ethos. | Wait until LOADED. |
| **OUTDATED** | Data loaded, but using an older model version. | Reload to refresh. Queries work but may be inaccurate. |
| **FAILED** | Load into Ethos failed. | Check Data Loads tab for issues. |
| **NOT LOADED** | Data not yet imported. | Set up loading from source system. |

**How to Create a GraphQL Consumer App**

When building a new app that will use GraphQL to query data in Ethos, you:

**Step 1 – Start a New App**

* Go to **APPLICATIONS → Create New App → Manually.**
* Check **Configure GraphQL resources.**
* Click **Continue.**

**Step 2 – Configure App Details**

* Enter an application name (required).
* Optionally add a description.
* Add your GraphQL resources (from the pool of already-loaded resources in Ethos).

**Step 3 – Add Subscriptions (Optional)**

* You can subscribe to change notifications if your app wants real-time updates rather than just periodic queries.

Otherwise, click **Skip**.

**Step 4 – Finish**

* A confirmation dialog appears showing your new app’s **API key**.
* You can go directly to the Application Overview page to review details.

**Attribute Filters for Data Privacy**

When setting up GraphQL resources, you can control what data is visible. For example:

* **Hide all dates of birth** from the persons resource.
* Hide dates of birth **only if a person’s record is flagged as restricted.**

This allows compliance with privacy rules (like FERPA, GDPR) without blocking access to entire records.

**Selecting Shared Resources**

If you share a tenant with other organizations, you can choose which resources you want to query:

Go to the app’s GraphQL Resources tab.  
Click **Add Resource.**  
Select a tenant and pick resources shared with you.  
Optionally, pick which app owns the resource if there are multiple owners.

**Benefits of GraphQL with Ethos**

* Retrieve only the data you need → **simpler, faster queries.**
* Consolidate many API calls into **one flexible query.**
* Bulk-read records for data analytics, reporting, or integrations.
* Protect sensitive data with fine-grained attribute filters.

**In Short:**

→ Ethos Data Access loads, stores, and manages data from your systems.  
→ GraphQL via Ethos Integration lets apps retrieve that data easily and efficiently.  
→ Status indicators help you know when your data is ready.  
→ Attribute filters keep your data secure and compliant.

This architecture makes integrations simpler, safer, and faster across your institution’s ecosystem.

**Ellucian Ethos Integration — API Keys Explained**

**What is an API Key?**

* An **API Key** is like a **password** that identifies and authenticates your application when it connects to Ethos Integration.
* It tells Ethos:
  + **Who you are** (the application)
  + **What you’re allowed to do** (permissions)

Without an API key, your application cannot talk to Ethos Integration’s APIs.

**Types of API Keys**

There are two types:

**1. Unrestricted API Key**

* No restrictions on where the request comes from.
* Can be used from **any IP address.**
* Easier to set up—but less secure if exposed publicly.

Good for:

* Development environments
* Internal systems behind firewalls

**2. Restricted API Key**

* **Restricted to specific IP addresses** or ranges.
* Only requests coming from those addresses are accepted.
* Adds a strong layer of security.

Good for:

* Production integrations
* Security-sensitive environments

**How to Add an Unrestricted API Key**

Let’s say you want to add another unrestricted API Key to your Ethos app.

**Steps:**

1. **Login:**
   * Go to:

Tools > Ethos > Integration

* + Choose your environment.

1. On the Ethos header bar, click: Applications
2. Find your app in the list and click its name.
3. On the Application Overview page, click: API Keys tab
4. Click: Add API Key
5. Click: Add API Key with no IP address restrictions
6. (Optional) Enter notes to describe the key’s purpose.
7. Click: Add

You’ve added a new unrestricted API key!

**How to Add a Restricted API Key**

For greater security, you might want your API key to only accept requests from specific servers.

**Steps:**

1. **Login:**
   * Go to: Tools > Ethos > Integration
   * Choose your environment.
2. Click: Applications
3. Find and click your app’s name.
4. Go to the: API Keys tab
5. Click: Add API Key
6. Click: Add API key with IP address restrictions
7. Enter allowed IP addresses or ranges.
8. (Optional) Add notes to document why this key exists.
9. Click: Add

Now only requests from the allowed IP addresses can use this API key.

**Managing API Keys Safely**

When you create an app in Ethos Integration, **one unrestricted API key** is automatically created.

For security and operational reasons, you should manage your API keys carefully.

**How to Change an API Key Safely**

If you need to replace an API key (e.g. it’s been exposed or you’re rotating credentials):

**Steps:**

1. **Create a new API key** for the app. Both keys will work for now.
2. **Update all systems** (code, configs, integrations) to use the new key.
3. Once everything uses the new key, **delete the old key.**

**Important:**

* Ethos won’t let you delete the last remaining API key for an app. Every app must have at least one key.

**Best Practices for API Key Security**

**DO NOT embed keys directly in code.**  
→ Instead, store them in:

* Environment variables
* Secure vaults (e.g. HashiCorp Vault, AWS Secrets Manager)

**DO NOT check API keys into source control.**  
→ Keep key files out of your repo (especially public ones like GitHub).

**Restrict API keys by IP address** whenever possible.  
→ Greatly limits exposure if a key is leaked.

**Limit API keys to specific APIs only.**  
→ Never grant unnecessary access.

**Rotate keys regularly.**  
→ Helps reduce security risk if a key is compromised.

**In Summary**

* API keys are essential credentials for Ethos Integration.
* Use **Unrestricted** keys for flexibility; use **Restricted** keys for security.
* Always follow secure handling practices:
  + Keep keys secret.
  + Rotate keys regularly.
  + Restrict access wherever possible.

This helps ensure your Ethos integrations remain secure and stable!

Bottom of Form

**Lesson 13: Support for multiple ownership of resources**

**What Is Multiple Ownership in Ethos Integration?**

**Multiple ownership** means that **more than one authoritative system or application** is responsible for maintaining data for the same Ethos resource (e.g. constituent-event-participations, email-types).

Instead of only one “owner” application, you might have:

* Two vendors managing the same data
* Two instances of the same vendor application (e.g. Main Campus vs. North Campus)
* Banner Multi-Entity Processing (MEP), where Banner is deployed for multiple institutions sharing the same system.

Ethos Integration **supports this scenario** through special capabilities:

* Multiple apps can own the same resource
* Consuming apps can subscribe to the specific instance of the resource they want
* Proxy API requests can be routed to the correct owner

**Use Cases Illustrated**

**① Multiple Vendors Manage the Same Information**

**Scenario**

* You have two different vendors (Vendor A and Vendor B) each managing event participation data for events they run on campus.
* Both vendors write to the same Ethos resource: constituent-event-participations.

**Diagram #1** (your first image) shows:

* **Vendor A → Ethos Integration ← Vendor B**
* Both owning constituent-event-participations.

**Why?**  
This allows flexibility so:

* Either vendor can **publish** data about events they manage.
* Other applications can **consume** this consolidated data from Ethos, regardless of which vendor created it.

**② Multiple Instances of a Vendor Application**

**Scenario**

* You have one vendor’s event management application installed separately:
  + One instance for Main Campus
  + Another instance for North Campus
* Both instances maintain their own data about event participation.

**Diagram #2** (your second image) shows:

* **Vendor A Main Campus → Ethos Integration ← Vendor A North Campus**
* Both owning constituent-event-participations.

**Why?**  
This supports **distributed management**:

* Each campus operates independently.
* But you can still bring all data together centrally in Ethos for reporting or cross-campus integrations.

**③ Banner Multi-Entity Processing (MEP)**

**Scenario**

* Banner is set up in a Multi-Entity Processing (MEP) environment, e.g. for:
  + Main Campus
  + North Campus
* Each Banner instance owns its own Ethos resources (e.g. email-types).

**Diagram #3** (your third image) shows:

* **Banner Main Campus Integration API → Ethos Integration ← Banner North Campus Integration API**
* Both owning email-types.

**Why?**

* Each campus’s Banner instance manages **its own data**.
* But Ethos provides a way to aggregate or query data from one or both instances as needed.
* Banner requires **two Ethos applications** per MEP instance:
  + One for the Integration API
  + One for the Student API
* Plus, there might be shared resources managed by shared apps.

**Ethos Integration Capabilities for Multi-Owner Scenarios**

Here’s how Ethos handles multiple ownership:

| **Feature** | **Multi-Owner Capability** |
| --- | --- |
| **Resource Ownership** | Multiple apps can own the same resource. One app is the default owner, but you can change it. |
| **Subscriptions** | Consuming apps can subscribe to specific instances of a resource from the desired owner. |
| **Proxy API Requests** | When a consuming app makes an API request, you can define which authoritative source to route it to. Otherwise, it goes to the default owner. |

**Practical Example**

Suppose you have:

* **Banner Main Campus Integration API** → owns email-types
* **Banner North Campus Integration API** → owns email-types

A consuming app wants email types:

* It can subscribe specifically to **Main Campus** or **North Campus**.
* Or make a proxy API call directed to a specific campus’s Banner instance.

**Why Multi-Owner Support Matters**

Without multi-owner support:

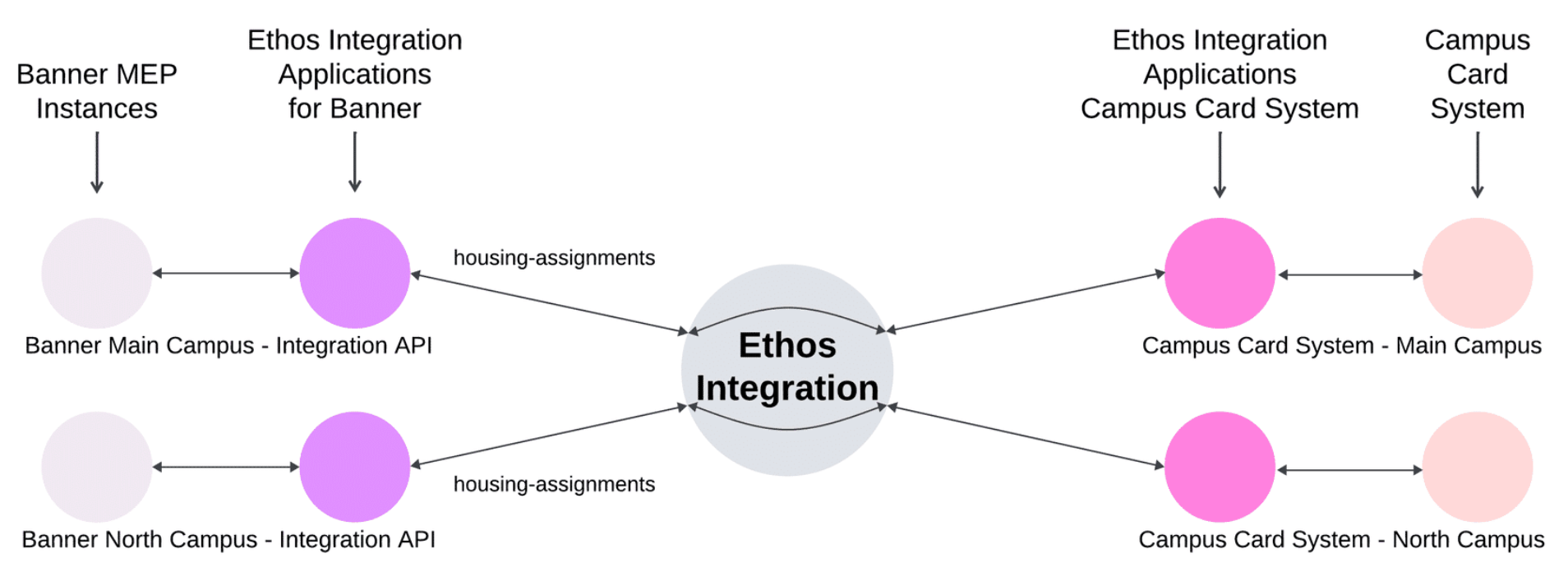
* Data would either be duplicated in different systems, or You’d have no way to reconcile data from multiple systems in one place.

With Ethos multi-owner support:  
- Each system can independently own and manage its data.  
- Centralized integration remains possible.  
- You avoid conflicts and maintain data integrity.

**In summary:** Ethos Integration’s multi-owner features make it possible to integrate systems in **complex, multi-tenant, or distributed environments**—like Banner MEP, or campuses using different vendors for overlapping functions—while preserving a single integration architecture.

**One-to-one relationship Example: Banner MEP and Campus Card System**

This example assumes that you have installed two Campus Card System application instances, one for each campus. In this example, the Main Campus instance of the Campus Card System application needs access to data in the Main Campus MEP instance, while the North Campus instance of the Campus Card System application needs access to data in the North Campus MEP instance.



The scenario is about **one-to-one relationships** between:

* **Banner MEP instances (authoritative systems)**
* **Campus Card System instances (consuming systems)**
* Connected via **Ellucian Ethos Integration**

**Context: Banner MEP**

**Banner MEP (Multi-Entity Processing)** = One Banner environment used by multiple campuses or institutions.

* E.g. Main Campus and North Campus are separate “instances” within the same Banner system.

Each MEP instance:

* Keeps its own separate data
* Needs separate integrations

**The Scenario**

**Systems Involved**

* **Banner Main Campus**
* **Banner North Campus**

Both run:

* Integration API
* Student API

And:

* **Campus Card System Main Campus**
* **Campus Card System North Campus**

**Why Integrate?**

The **Campus Card System** needs:

* **Housing Assignments** from Banner → to control door access, etc.

**Banner** is the authoritative owner of housing-assignments data.

**How Integration Happens**

There are **two ways** the Campus Card System can access Banner data through Ethos:

**Option 1 — Subscriptions**

**Scenario:**

* Banner publishes housing-assignments data updates into Ethos Integration.

Campus Card System:

* Subscribes to those events
* Ethos Integration sends notifications when data changes

Example:

* A student changes dorm room → Campus Card System automatically gets notified and updates door access.

**Key Details:**

* Each Campus Card instance subscribes only to its matching Banner instance:
  + Main Campus Card → subscribes to Main Campus Banner
  + North Campus Card → subscribes to North Campus Banner

Because **housing-assignments** is owned by both Banner instances, Campus Card System **must choose which Banner instance’s data to subscribe to.**

**Option 2 — Proxy API Requests**

Instead of waiting for updates:

* Campus Card System directly **queries Banner** through Ethos.

Campus Card System calls Ethos:

* “Give me housing-assignments for this student.”

Ethos Integration:

* Forwards (proxies) the API request to the correct Banner system.

**Key Details:**

* Again, each Campus Card instance makes proxy API requests only to its campus-specific Banner Integration API.

If a resource is owned by multiple systems (like housing-assignments), you may have to configure which system the proxy request goes to.

**Why Separate Instances?**

* **Data separation:** Main Campus and North Campus keep separate housing data.
* **Security:** Prevents cross-campus data leaks.
* **Flexibility:** Each campus can evolve independently.

Hence, you need:  
- Separate Ethos Integration applications  
- Separate API keys  
- Separate subscriptions or proxy configurations

**Steps Summarized**

**Step 1** — Create Ethos Integration applications:

* Banner Main Campus Integration API
* Banner Main Campus Student API
* Banner North Campus Integration API
* Banner North Campus Student API
* Campus Card System Main Campus
* Campus Card System North Campus

**Step 2** — Assign resources:

* housing-assignments owned by both Banner campus Integration API apps

**Step 3** — Set up subscriptions:

* Main Campus Card → subscribes to Main Campus housing-assignments
* North Campus Card → subscribes to North Campus housing-assignments

**Step 4** — Set up proxy API requests (if needed):

* Configure requests so Ethos Integration forwards queries to the right Banner campus Integration API

**Diagram Recap**

* Left side: **Banner Main & North campus**
* Center: **Ethos Integration**
* Right side: **Campus Card System Main & North campus**

Arrows:

* Represent data flow of **housing-assignments**
* Go both ways (read/write capability)

**Why Use Ethos?**

Without Ethos Integration:

* Campus Card System would have to integrate directly with Banner’s APIs
* High coupling → brittle integrations
* Harder to manage multiple campus contexts

With Ethos:  
- Decoupled  
- Standardized APIs  
- Simplified subscriptions and proxy calls  
- Clear separation of data per campus

**Short Summary**

This is a **one-to-one integration** between:

* Banner campus-specific data
* Campus Card campus-specific systems

Using:

* Ethos Integration as the middleware
* Either subscriptions or proxy API requests to sync data

This ensures:

* Data security
* Campus-specific control
* Scalability

**Many-to-one relationship Example: Banner MEP and Campus Card System**

**Situation:**

* You have **one Campus Card System application instance**.
* That one application needs **data from multiple Banner MEP instances** (Main Campus + North Campus).

So, unlike the previous example (where each campus had its own Campus Card System), here you have:

**One consuming system connected to multiple authoritative systems.**

**What’s Different From the One-to-One Scenario?**

**One-to-One:**

* Campus Card Main Campus only talks to Banner Main Campus.
* Campus Card North Campus only talks to Banner North Campus.

**Many-to-One:**

* **One** Campus Card System needs access to **both** Banner MEP instances.
* It has to **keep the data separate** inside itself so it knows which Banner instance the data came from.

**Why Do You Still Need Multiple Ethos Integration Applications for Campus Card?**

This is the part that confuses people.  
Even though you only have **one** Campus Card System app, you must create:

* One Ethos Integration Application for **Main Campus**
* One Ethos Integration Application for **North Campus**

**Reason:**

* Each Ethos Integration Application holds:
  + Its own **API Key**
  + Its own **resource ownership and subscriptions**
  + Its own **routing to the correct Banner MEP**

This separation ensures:  
- Secure separation of Main/North Banner data.  
- You can control subscriptions separately.  
- You can control which Banner MEP the request goes to.

**How It Works**

Let’s break this down step by step:

**Systems Involved**

**Authoritative Systems (Data Owners):**

* Banner Main Campus Integration API
* Banner North Campus Integration API

**Consuming System:**

* **One** Campus Card System application

**Ethos Integration Applications:**

* 2 for Banner (Main and North)
* 2 for Campus Card System (Main and North)

**Steps in Detail**

**Step 1 — Create Ethos Integration Applications**

Even though you have **one Campus Card System**, you create:

* 2 applications for Banner (Main and North)
* 2 applications for the Campus Card System (Main and North)

**Why?**

* So each Campus Card Integration Application can separately:
  + Subscribe to the correct Banner data
  + Make proxy requests to the correct Banner system

**Step 2 — Assign Resources**

You define that the **housing-assignments** resource is:

* Owned by **Banner Main Campus Integration API**
* Also owned by **Banner North Campus Integration API**

**Step 3 — Subscriptions**

If you want **notifications when Banner data changes**:

* For Main Campus data:
  + In the Ethos Integration application for **Campus Card System Main Campus**, create a subscription to housing-assignments **owned by Banner Main Campus**.
* For North Campus data:
  + In the Ethos Integration application for **Campus Card System North Campus**, create a subscription to housing-assignments **owned by Banner North Campus**.

When Campus Card System gets notified, it knows which subscription and API key it came from—so it can tell **which campus** the data belongs to.

**Step 4 — Proxy API Requests**

If you want to **query data directly**:

* To get Main Campus data:
  + The Campus Card System uses the **API key** from **Campus Card System Main Campus Integration App**.
  + This routes the request to **Banner Main Campus Integration API**.
* To get North Campus data:
  + The Campus Card System uses the **API key** from **Campus Card System North Campus Integration App**.
  + This routes the request to **Banner North Campus Integration API**.

This keeps the data flow clean and separate.

**Why Do It This Way?**

Imagine you didn’t have separate Integration Applications:

* One API key wouldn’t know which Banner MEP to call.
* You could mix Main and North Campus data.
* Subscriptions wouldn’t be scoped to the right Banner instance.

By creating **separate Integration Applications for each pairing**, you maintain:  
- Data isolation  
- Correct routing  
- Granular permissions and subscriptions

A diagram of a company

AI-generated content may be incorrect.

**Left Side:**

* Banner Main + North Campus Integration APIs

**Center:**

* Ethos Integration (the hub)

**Right Side:**

* Two separate Ethos Integration Applications for Campus Card System
  + Main Campus
  + North Campus

Even though the **actual Campus Card System is one single app**, you **split the logical connections in Ethos**.

**Simple Analogy**

Think of this as:

* You have **one warehouse** (Campus Card System).
* You have **two suppliers** (Banner Main, Banner North).
* You create **two purchase orders**, one per supplier, even though everything ends up in the same warehouse.
* The purchase orders (Ethos Integration Applications) tell each supplier **which items to ship and where they came from**.

**Key Takeaway**

This setup ensures:

* The Campus Card System can **fetch or subscribe to data** separately from Main and North Campus Banner instances.
* Each interaction is **properly identified and isolated** via its dedicated Integration Application and API key.

**Lesson – 14: Ethos Integration Fundamentals for Developers-v2024**

**Data Integration Types — Simplified Explanation**

There are **two main ways to integrate applications** so they can share data:

**1. Point-to-Point (P2P) Integration**

**What is it?**

* Think of **direct wires** connecting one application to another.
* If App A needs data from App B → you build a **custom connector** just for those two.
* You must **repeat this for every pair** of systems that need to talk.

**Example:**

* Banner ↔ Campus Card System
* Banner ↔ Library System
* Campus Card System ↔ Housing System

Each arrow means:

A separate, unique integration you have to build, test, and maintain.

**Problems with P2P**

1. **Exponential Growth**
   * The more systems you have, the more connections you need.
   * For 10 systems, you could have up to 45 connections! (combinatorial explosion)
2. **Redundant Work**
   * Many integrations do similar things (e.g. moving student info).
3. **Maintenance Nightmare**
   * Change one system → potentially break many integrations.
   * Updates become costly and time-consuming.

**2. Ethos Integration**

**What is it?**

* Instead of direct wires, **all apps plug into a central hub.**
* The hub becomes the “middleman.”
* Applications don’t talk directly → they talk to the hub.

**Analogy:**

* Think of **an airport hub**:
  + Instead of flying directly between every two cities, flights go through a central hub city.
  + Fewer routes → easier logistics → cheaper and more efficient.

**How Ethos Helps**

**One connection per app**

* App A → Hub
* App B → Hub
* App C → Hub
* No need for A↔B, A↔C, B↔C separately.

**Standardized API**

* Uses REST APIs over HTTPS.
* All apps speak the same “language” with the hub.

**Flexibility**

* You can:
  + Swap out apps without breaking everything.
  + Integrate cloud apps or on-prem apps in one architecture.

**Authoritative Data**

* You can define:

“Banner is the official source for student housing assignments.”

* All apps pull that data from the same place.

**Easier Maintenance**

* Update one integration in the hub → all apps benefit.

**Quick Comparison**

| **Feature** | **P2P** | **Ethos Integration** |
| --- | --- | --- |
| Connections Needed | Many (exponential growth) | One per app |
| Complexity | High | Lower |
| Maintenance Cost | Expensive and redundant | Much cheaper and easier |
| Standardization | Custom per connection | Hub enforces standards |
| Flexibility | Low | High |

**Key Takeaways**

* P2P = lots of custom “wires” → fragile and costly.
* Ethos = plug everyone into a central hub → efficient, standardized, and future-proof.

That’s why Ellucian and many modern systems are moving away from P2P toward hub-based integration like Ethos.

**Lesson – 15 : Ethos Integration APIs and Services**

**Developing Custom Apps for Ethos Integration**

When you build custom apps that connect to Ellucian’s **Ethos Integration**, you have to follow certain **rules and standards** so your apps can talk smoothly with others in the ecosystem.

Think of it like following **traffic rules** so everyone can safely drive on the same roads!

These rules fall into two groups:

**1. Ethos Integration Governing Principles**

These are the **big-picture rules** guiding how apps connect to Ethos.

Here’s what they mean, in plain English:

**Adhere to REST principles and semantics**

* **REST** = Representational State Transfer
* Apps should communicate using **HTTP methods** like:
  + **GET** → fetch data
  + **POST** → create data
  + **PUT/PATCH** → update data
  + **DELETE** → remove data
* Makes integrations predictable and standard.

**Quantify data as entities or resources**

* Think of **data as objects**:
  + Student
  + Course
  + Housing Assignment
* Each object is a **resource** accessible via an API URL.

Example: GET /students

**Use JSON schema to define data structures**

* Data shared via Ethos APIs is in **JSON** format (JavaScript Object Notation).
* JSON schema defines:
  + What fields exist
  + Their types (e.g. string, number, date)
  + Rules (e.g. required fields)

This makes it clear:

“Here’s how data should look so everyone understands it.”

**Employ the concept of an authoritative source**

* For every piece of data, **one system is the “source of truth.”**
* Example:
  + Banner is authoritative for housing-assignments.
  + Another app shouldn’t randomly overwrite this data.

This avoids:

Confusion about who’s right when data conflicts happen.

**Use standard messaging integration patterns**

* Ethos uses **standard ways to share data**, like:
  + **Event Notifications:**  
    “Hey, housing-assignment changed!”
  + **API calls:**  
    “Give me the latest housing-assignment.”

Keeps apps loosely coupled and easier to maintain.

**2. Ethos Integration API Standards**

These are **detailed technical rules** for building APIs that work with Ethos.

Here’s a simple summary of each:

**Role of authoritative source, data model, and the subscriber**

* Every API:
  + Defines who owns the data (authoritative source)
  + Uses a standard data model (e.g. housing-assignments)
  + Lets other apps **subscribe** to updates

**API naming, operations, behaviors, versioning, filtering, endpoints, and headers**

* APIs must:
  + Have consistent names
    - e.g. /students, not /getStudentData
  + Support standard HTTP methods
  + Allow **versioning**
    - e.g. /v1/students
  + Offer filtering:
    - e.g. GET /students?lastName=Smith
  + Use standard headers for security and data formats.

**Change notification**

* Apps can subscribe to:

“Tell me when housing-assignment data changes.”

So apps stay in sync without constantly polling for updates.

**Documentation**

* APIs must include **clear documentation**:
  + How to use them
  + What data fields exist
  + Example requests and responses

**JSON schema**

* Data must be structured in **standard JSON schemas** so everyone knows what data looks like.

**Error message schema**

* When things go wrong:
  + Errors should follow a **standard format.**
  + So apps can handle problems reliably.

Example:

{

"errors": [

{

"title": "Not Found",

"detail": "Student record does not exist.",

"status": 404

}

]

}

**Batches**

* APIs should handle **bulk data operations** efficiently.
* E.g. create multiple students in one request.

**HTTP status codes**

* Use standard codes like:
  + 200 → OK
  + 201 → Created
  + 400 → Bad Request
  + 404 → Not Found
  + 500 → Server Error

Makes it easier for apps to know what happened.

**Change log**

* Document changes to the API:
  + What changed?
  + When?
  + Why?

So developers stay informed and avoid surprises.

**Why These Rules Matter**

By following these principles and standards:

* Apps integrate **smoothly** with Ethos.
* Data stays **consistent and trustworthy.**
* Maintenance becomes **easier and cheaper.**
* New apps can plug in **without major rewrites.**

In short:

“Play by these rules so your apps fit into the Ethos ecosystem—and everything just works.”

**Whitelisting Ethos Integration IP Addresses**

When you use **Ellucian Ethos Integration**, your campus systems and firewalls need to allow traffic between:

**Your systems** (like Banner, ERP, Campus Card, etc.)  
and

**Ethos Integration services** running in the cloud (AWS)

Otherwise, Ethos Integration **can’t reach your systems** to send or receive data.

**Why Does Ethos Integration Need Firewall Rules?**

* Ethos uses **Proxy Services** to:
  + Call APIs on your campus systems (inbound requests)
  + Let your campus systems send data back to Ethos (outbound requests)
* These connections go through the **public internet**, so your firewall must know:

“These IPs are safe—let their traffic through.”

**Two Sides of Network Traffic**

There are **two kinds of IP addresses** you need to configure:

**1. Inbound IPs**

These are the **Ethos Integration IPs that connect INTO your network**.

* For example, Ethos wants to:
  + Call your Banner Integration API
  + Retrieve housing-assignments data
  + Push updates to your applications

So, your firewall needs to **allow incoming connections** from these IPs.

**Inbound Static IPs to Allow:**

| **Region** | **IP Addresses** |
| --- | --- |
| **USA (us-east-1)** | 50.16.144.60 52.0.163.27 |
| **Australia/APAC (ap-southeast-2)** | 52.65.227.164 54.66.193.28 |
| **Europe (eu-west-1)** | 52.210.253.252 52.19.193.15 |
| **Canada (ca-central-1)** | 35.182.43.187 35.182.100.39 |

**2. Outbound IPs**

These are the **IP addresses your campus systems need to reach OUT TO** if they call services in Ethos Integration.

* For example, your Banner system might:
  + Send updates to Ethos via the proxy service
  + Call Ethos to retrieve integration data
  + Send error messages, authentication requests, etc.

So your firewall must **allow outbound traffic to these IPs.**

Here’s how they’re organized:

* **admin-static.integration.elluciancloud.com**  
  For admin services
* **errors-static.integration.elluciancloud.com**  
  For error reporting
* **integrate-static.elluciancloud.com**  
  For integration services
* **messages-static.integration.elluciancloud.com**  
  For messaging services
* **proxy-static.integration.elluciancloud.com**  
  For proxy connections
* **tokens-static.integration.elluciancloud.com**  
  For authentication and token handling

Each domain has **two static IPs** for redundancy.

**Outbound Static IPs to Allow (Example for USA region):**

| **Hostname** | **IPs** |
| --- | --- |
| admin-static.integration.elluciancloud.com | 99.83.237.160, 75.2.4.231 |
| errors-static.integration.elluciancloud.com | 13.248.200.205, 76.223.88.189 |
| integrate-static.elluciancloud.com | 76.223.72.162, 13.248.209.218 |
| messages-static.integration.elluciancloud.com | 99.83.200.129, 75.2.44.239 |
| proxy-static.integration.elluciancloud.com | 76.223.2.146, 13.248.193.245 |
| tokens-static.integration.elluciancloud.com | 99.83.234.134, 75.2.32.220 |

Each other region (Australia, Europe, Canada) has similar lists with their own regional IPs.

**Why Regional IPs?**

* Ethos Integration uses AWS Global Accelerator.
* This means traffic routes to the **nearest AWS region** for:
  + Lower latency
  + Better performance
* You need to whitelist IPs for the region where your instance is deployed.

**What You Need To Do**

* Check which AWS region your Ethos services run in (e.g. us-east-1).
* **Inbound rules:**
  + Allow the Ethos inbound IPs **into your campus network**.
* **Outbound rules:**
  + Allow traffic **outbound from your network** to all the Ethos IPs listed for your region.
* Test connections to:
  + Proxy services
  + Integration APIs
  + Authentication services

**Why Whitelisting Matters**

Without whitelisting:

* Ethos Integration can’t connect to your systems.
* Your apps might fail to push or pull data.
* Integrations and event notifications break.

Whitelisting ensures:

* Secure, reliable communication
* Smooth integration flows
* Compliance with security best practices

**Security Note**

* These IPs are **static and won’t change** (per Ellucian’s statement).
* Safe to configure permanently in your firewall.
* Helps keep your network secure while enabling integrations.

**In short:**

“Ethos Integration needs your firewall to allow specific IPs for inbound and outbound traffic so it can connect to your campus systems and provide integration services.”

**How to Retrieve an Access Token for Ethos Integration APIs**

When your app wants to **talk to Ellucian Ethos Integration**, it first needs a “key” to unlock the door:  
→ **An access token.**

This token is like a temporary “badge” that proves who your app is and what it’s allowed to access.

**How Ethos Integration Works**

Instead of connecting every app directly to every other app (which creates a big mess of connections), all apps connect to a central hub:

Hub = **Ethos Integration**

This hub:

* Standardizes data formats (using the Ethos Data Model)
* Uses **REST APIs** over HTTPS
* Makes it easier to swap or change applications
* Improves security

**Why You Need an Access Token**

Ethos Integration uses security to ensure only authorized apps can talk to it.

* Your app **must authenticate** using its unique **API Key**.
* In return, Ethos gives you an **access token** (a JWT token).
* You **include this token in API calls** to:
  + Read data
  + Publish events
  + Subscribe to changes
  + Use proxy services

**How to Get an Access Token (Step-by-Step)**

Let’s say you’re using an API tool like:

* **Bruno**
* **Postman**
* **Insomnia**

**① Make a POST Request**

* **HTTP Method:** POST
* **URL:** https://integrate.elluciancloud.com/auth

Note: If you’re in another region (like Europe or Canada), replace:

integrate.elluciancloud.com

with your region-specific domain.

**② Set Authorization Header**

In the **Headers** section of your request:

| **Header Key** | **Value** |
| --- | --- |
| Authorization | Bearer YOUR\_API\_KEY\_HERE |

* The **API Key** comes from the Ellucian Ethos Integration configuration for your application.
* Think of this like a password identifying your app.

**③ Send the Request**

* Click **Send** in your API client.

If successful, you’ll receive:

**A JWT Access Token**  
→ a long string starting with **ey...**

**What’s Inside the JWT Token?**

JWT stands for **JSON Web Token**. It securely carries identity and authorization information.

**Example Payload Data**

When you decode the token (e.g. using [jwt.io](https://jwt.io/)), you’ll see JSON data like:

{

"sub": "abcd-1234-5678-efgh",

"tenant.id": "a1b2c3d4",

"tenant.accountId": "abc-123",

"tenant.alias": "my-institution-alias",

"tenant.name": "My University Prod",

"tenant.label": "Production",

"apiKeyPrefix": "XXXX-XXXX-XXXX-XXXX",

"iat": 1711200000,

"exp": 1711200300

}

**Meaning:**

* **sub** → Your Ethos Integration App ID
* **tenant.id** → Your unique tenant ID in Ellucian Cloud
* **tenant.accountId** → Cloud account ID
* **tenant.alias** → Alias used in Experience Dashboard
* **tenant.name** → Your environment name (e.g. “University Prod”)
* **tenant.label** → Usually “Test” or “Prod”
* **apiKeyPrefix** → The API Key that generated the token
* **iat** → “Issued At” time (when token was created) — epoch time
* **exp** → “Expires” time (when token becomes invalid) — epoch time

**How Long is the Token Valid?**

**5 minutes.**

* After that, it **expires** and you must request a new token.

This short expiration keeps things secure.

**Security Reminder**

* **JWT tokens are sensitive credentials.**
* Don’t paste them into public chats, documents, or unsafe tools.
* Tools like [jwt.io](https://jwt.io) are safe because:

“JWT.io does not record tokens. All validation happens on your machine.”

**Next Steps After Getting a Token**

→ You include this token in the Authorization header for **ALL your API calls** to Ethos services, like:

* Publishing events
* Subscribing to changes
* Calling the proxy
* Fetching resources

Example usage: Authorization: Bearer eyJhbGciOi...

**Summary:**

Your app authenticates with its API Key → receives a JWT access token → uses that token to securely call Ethos APIs for 5 minutes.