An Introduction to Cu Chulainn: Our Agentic Workflows Platform

Links

- CS Prod environment
- CS Dev environment
- SaaS environment
- M&A environment

Example payload

CS Cu Chulainn is meant to be invoked by ATLAS Ticket Studio; however, you can also directly invoke the adapters. Below is an example using the Kayako adapter for the **CS dev** environment, which interfaces with the cs-sandbox Kayako environment.

```
None
API ENDPOINT:
https://dev.cu-chulainn.csaiautomations.com/api/external-adapter/kayako
REQUEST TYPE: POST
SAMPLE PAYLOAD:

{
    "testMode": true or false,
    "adapterParameters": {
        "ticket_url": full ticket URL in string format(i.e.
https://cs-sandbox.kayako.com/agent/conversations/4646),
        "package_name": cu chulainn package name
    }
}

IMPORTANT: This API endpoint requires a Key to be passed in the 'X-API-Key'
header for authentication. This key can be found in the AWS Secrets Manager
under cu-chulainn/dev/api-key.
```

1. What is Cu Chulainn?

Cu Chulainn is a platform designed to facilitate the deployment of Agent swarms to carry out tasks across various platforms. At its core, Cu Chulainn consists of two main components: the Marketplace and the Core.

- **Marketplace:** A web app where users can create workflows, Agents, and APIs, allowing for seamless integration and reuse across different challenges.
- **Core:** The heart of the system, which runs in Amazon ECS. It utilizes the AutoGen AgentChat framework to dynamically launch and configure Agent swarms with the designs created in the Marketplace.

This guide is intended to provide users with a foundational understanding of Cu Chulainn, guiding them through the process of developing for the platform and utilizing its capabilities to efficiently manage and execute tasks.

1.1 Scope and Intended Use

Cu Chulainn is designed to be entirely system-agnostic, meaning it can seamlessly interface with any platform or service provided an Agent with the required capabilities exists. This flexibility makes the framework highly adaptable, allowing developers to connect and operate across diverse environments.

As a framework, Cu Chulainn empowers users to build everything, from railroaded, structured processes to open-ended, exploratory solutions. Success ultimately depends on thoughtful design—crafting clear prompts, developing well-defined APIs, and creating Agents that effectively leverage the platform's features.

1.2 How Does Cu Chulainn Work?

Cu Chulainn operates through a set of core components—Skills, Agents, and Packages—that collectively define how tasks are approached and executed. An Adapter serves as the entry point for each task, setting up the environment in which these components interact.

1.2.1 Functional Elements

Skills

- Skills are APIs that Agents can call to perform specific actions.
- Each Skill points to an endpoint and includes defined input parameters and expected outputs.
- Skills are created by users and should support authentication headers and read-only modes.

Agents

- Agents are collections of Skills governed by a system prompt that shapes their behavior.
- By default, every execution includes three “system” Agents:
    - Manager – Coordinates all other Agents.
    - Auxiliary Agent – Has access to skills not assigned to other Agents.
    - Andon Corder – Monitors for errors and unexpected states, and can halt execution if necessary.
- Other Agents can be explicitly defined and assigned to a package.

Packages

- A Package determines which Agents will participate in a given task and how they will collaborate. As a rule of thumb, one package equals one intent.
- Each Package includes:
    - A workflow that describes the process or logic to be followed.
    - Optional approval criteria, defining conditions that must be met for write actions to take place.
    - A list of Agents that will be involved, beyond the default system Agents.
    - A list of skills accessible by the Auxiliary Agent at the Manager’s request.

1.2.2 The Execution Process

A standard execution run of Cu Chulainn looks as follows:

1. First, **an adapter is invoked** with the package to be launched and the input data for the task.
    - In the case of CS, ATLAS SRT is expected to select the package to be executed for a given ticket.
    - If the Kayako adapter is used, for example, only the ticket URL needs to be supplied to the endpoint, with the adapter gathering all the relevant ticket data before starting the core.

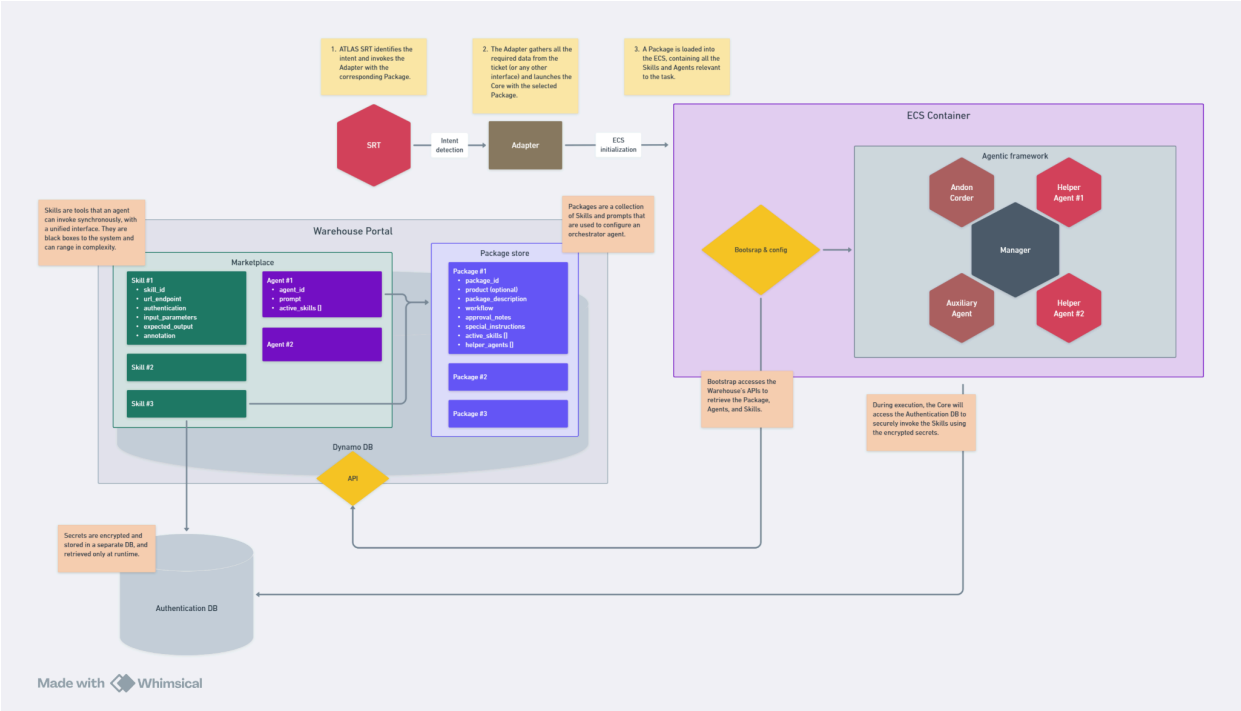
2. Afterward, **the Core is launched,** and the relevant data for configuring the Agent swarm is gathered.
    - The package, Agents, and skills are retrieved from the database and prepared to be loaded into the AutoGen framework, alongside the internal prompts.
    - Authentication data is also retrieved and decrypted within the ECS to call the skill endpoints.

3. **An AutoGen AgentChat session is launched** with the configuration from the previous step, and the Agents start collaborating to carry out the task.

4. They **continue their work** until one of the following happens:
    - The swarm completes the task and exits gracefully.
    - The andon cord is pulled. Different andon cord skills exist to hand over the task to a human Agent in each system that Cu Chulainn is set to operate with.

- The system goes into an execution loop and reaches the preconfigured message limit.

1.2.4 System Diagram



1.3 How Do I Start Using Cu Chulainn?

Users with a Trilogy (@trilogy.com) email address can access Cu Chulainn using Google SSO. Note that you will have a different account and, potentially, access level in each environment. Please contact your manager in either of these scenarios:

- You need to access Cu Chulainn, but you do not have a Trilogy email address.
- You need admin access in a certain Cu Chulainn environment.

2. Designing for Cu Chulainn

Design plays a major role in getting the best results from Cu Chulainn. Well-structured Skills, Agents, and Packages make it easier to handle complex tasks and deliver consistent outcomes. This section covers the basics of building these components effectively.

2.1 Creating Skills

Skills are endpoints that AI Agents can call to accomplish specific tasks. Each Skill includes a piece of “annotation” that provides the AI with essential details on how to use

it—such as inputs, outputs, and common use cases. Because Skills are exposed alongside their annotations, Agents can choose and invoke them whenever they detect a need that matches a Skill's functionality.

Skills need to be added to a Package to be used in resolving a task. A Skill can be included in a Package in two ways:

- **By assigning it to an Agent, so it can be reused across multiple Packages.** This approach allows for broader reusability of the Skill across different workflows.
- **By adding it directly to the Package, making it accessible to the Auxiliary Agent.** This option is often suitable for simple Skills that are tightly coupled to a specific workflow.

Skills are typically implemented as HTTP POST endpoints, often via serverless functions (e.g., AWS Lambdas) or microservices. Cu Chulainn treats these endpoints as "black boxes," sending the required parameters in the request body and expecting a structured response in return. As long as the endpoint accepts POST calls with defined inputs, it can be integrated into the system.

2.1.1 Skill Fields

- **Skill Name:** A meaningful, AI-facing name that clearly indicates what the skill does. Since the AI sees this name, it's important to keep it descriptive and unique.
- **Skill Description:** A short explanation of the skill's purpose and how it should be used. Think of this as the AI's reference for when to call the skill and what it does.
- **Skill Tags:** Tags for easier organization and searching within the Marketplace.
- **URL Endpoint:** The POST endpoint for the Skill, which Cu Chulainn calls at runtime.
- **Skill Authentication:** A token stored securely by the system and passed in the `authorization` header.
- **Skill Timeout:** The maximum time (in seconds) for the system to wait for a response. If the endpoint exceeds this time, the skill call is considered to have timed out.
- **Input Parameters:** The parameters expected in the POST request body.
- **Output:** The format and structure of the data returned by the Skill.
- **Annotation:** AI-facing documentation that explains the Skill's purpose, inputs, and outputs. This is initially AI-generated but can be refined manually for clarity and precision.

2.1.2 Best Practices

- Include detailed and accurate information in the annotation so AI can use the Skill effectively.
- Skills must account for all errors and return a descriptive output. AI will know that `404 - User not found` means that there's no such user, while a `500` or an empty response will usually lead to the system pulling the andon cord.

- Remember that the URL parameter `?test_mode=true` should be parsed: when it is true, no write action should take place.
- Ensure that your system expects the authentication string in the "authorization" header.
- Refer to the Marketplace tooltips for additional pointers on designing, documenting, and structuring Skills.

2.2 Creating Agents

Agents participate in the conversation to address a given issue. Each Agent is guided by its own system prompt and has access to their assigned Skills. In Cu Chulainn, users can explicitly add Agents to a Package, but there are also default Agents always present in any execution:

- **Manager:** Coordinates the flow of the conversation, requests actions from other Agents, and makes high-level decisions.
- **AuxiliaryAgent:** Has direct access to any Skills added to the Package itself.
- **AndonCorder:** Monitors for unexpected states and unrecoverable errors in Skill usage. Different Adapters may slightly adjust its behavior so outputs align with the platform's needs.

Users should create dedicated Agents when there's a common task that needs to be handled in different scenarios, or which should not be given to the AuxiliaryAgent.

2.2.1 Agent Fields

- **Agent Name:** A unique, descriptive name that helps AI identify the Agent's role.
- **Agent Description:** A detailed explanation of the Agent's purpose and capabilities. The AI uses this to understand the Agent's role and when it should be consulted.
- **Agent Tags:** Labels for organizing and searching Agents within the Marketplace.
- **Agent Prompt:** The system prompt that controls the Agent's behavior, guiding how it responds to tasks and uses their skills.
- **Skills:** The set of Skills assigned to this Agent. The AI knows the annotations for these Skills and uses them as needed.

2.2.2 Best Practices

- Avoid duplicating Skill information in the Agent's prompt, as the system already shares that data.
- Include any additional usage details in the prompt that are not covered by the Skill annotation.
- Keep the Agent description clear and focused, since the Manager relies on it to decide when the Agent's input is needed.

2.2.3 When should I create an Agent?

Note that you do not always need to create Agents to have skills assigned. While a KayakoAgent or a JiraAgent may be key to manage a large number of skills and reuse them across packages, skills to update a license do not usually need their own Agent. Think about the following when trying to determine if a separate Agent is needed for a task:

- **Do I need additional, complex logic that is not captured in the skills or workflows?** The AuxiliaryAgent can make use of skills based on the information in the Package's instructions and the Skill annotations. However, if there is complex logic about skill interaction, a specialized Agent can be of use.
- **Do I have a large number of skills that should be grouped together?** A separate Agent can be of great use when there's a large number of skills that should be grouped together, as it removes these from the Auxiliary Agent and allows related tasks to go through the same Agent. This is the case of the platform intermediaries, such as the KayakoAgent, JiraAgent, etc.
- **Will this Agent be used in multiple packages?** Compounding with the above is the usefulness of simply adding the same Agent to different packages, instead of each of the skills it can access. This should be secondary, however, as the gain can be minor.

2.3 Creating Packages

Packages are the most important element in Cu Chulainn because they tie everything together. They specify the instructions the AI should follow, determine which Agents participate, and define which Skills are accessible to those Agents (or directly to the Auxiliary Agent).

2.3.1 Package Fields

- **Package Name:** A unique, descriptive identifier used to launch the Package (not visible to AI).
- **Package Product Name:** A product label used mainly for housekeeping in the Marketplace.
- **Package Description:** An internal explanation of the Package's purpose (not visible to AI).
- **Package Tags:** Tags for organizing and locating the Package in the Marketplace.
- **Package Workflow:** The task(s) the agent swarm needs to carry out. This can be a detailed set of steps or an open-ended directive.
- **Special Instructions:** Any additional guidelines on how the Package should be executed.
- **Approval Notes:** Criteria or requirements the AI must meet before finalizing any action.
- **Package Active Skills:** Skills directly available to the Auxiliary Agent. Avoid including Skills here if they belong to a specialized Agent.
- **Package Helper Agents:** Additional Agents that join the Manager, AuxiliaryAgent, and AndonCorder. These Agents come with their own Skills and prompts.

2.3.2 Best Practices

- Use a clear, step-by-step format in Package Workflow, detailing when and how steps can be skipped.
- For open-ended tasks or investigations, list any required steps and define passing criteria.
- Ensure you understand how Cu Chulainn operates so you can define logical exit points (e.g., sending a pull request might be the final step).
- Reinforce approval requirements under Approval Notes if needed.
- Only include Skills in Package Active Skills if they aren't already tied to a specific Agent.

2.3.3 How to Create a Successful Package

- When a process is known in advance, break it down into clear steps: for example, "Skip to step 6 if the user is internal."
- Define precise exit criteria or completion triggers for open-ended tasks (e.g., a fixed number of attempts or a checklist of items to validate).
- Avoid ambiguous instructions in your workflow. Specific actions lead to better results.
- Don't tell Agents which Skills to use or how; instead, describe the process as if explaining it to a person ("Send a side conversation with X details").
- Make sure you fully understand how Agents interact with external systems. For instance, sending a PR or changing a ticket's status is typically a final step, and Cu Chulainn won't continue to monitor external systems without a new trigger.

3. Using Cu Chulainn

This section explains how to put your Packages into action. We'll look at how Adapters serve as the entry point into Cu Chulainn, how to launch the system with the right parameters, and how the agent swarm then executes the tasks you've defined. By the end, you'll have a clear understanding of how to run Cu Chulainn and integrate it with other platforms or services.

3.1 Adapters

Adapters are the primary way to launch a Cu Chulainn session. By calling an Adapter's endpoint, external systems or users can supply all the information needed to start an execution. Adapters are configured under Admin > Adapters in the Marketplace and define which parameters must be included in a POST request to trigger the system.

3.1.1 The Core Adapter

Every Cu Chulainn environment comes with a built-in Core Adapter. Although it can be called directly, most environments benefit from specialized Adapters that handle preprocessing before passing data to the Core. The Core Adapter expects the following parameters:

- **package_id *(optional if package_name is provided)*:** ID of the Package to be executed.
- **package_name *(ignored if package_id is provided)*:** Optionally, provide the name of the Package instead.
- **andon_cord_skill_id *(optional if andon_cord_skill_name is provided)*:** References the Andon Cord skill to be used to abort if errors arise. These skills are also configured under the Admin settings.
- **andon_cord_skill_name *(ignored if andon_cord_skill_id is provided)*:** Optionally, reference the Skill name instead.
- **input:** A string embedded into the task prompt, providing context or data for agents to use.

When these parameters are provided, the Core Adapter spins up the agent swarm in ECS to carry out the specified workflow.

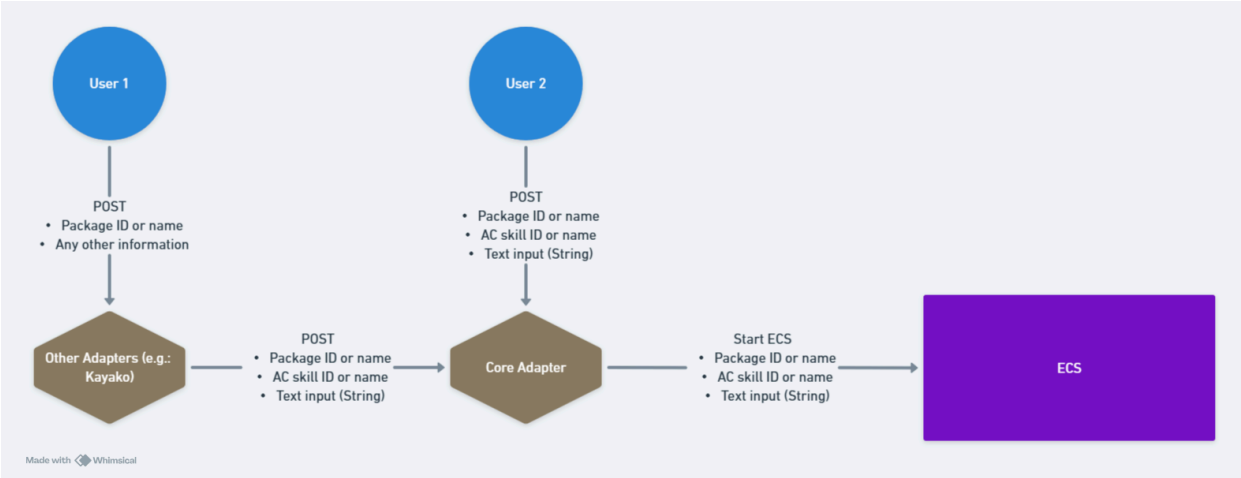
3.1.2 Custom Adapters

Although users can call the Core Adapter directly, it's common to set up environment-specific Adapters that gather and format input data first, and then pass it along to the Core Adapter. To better understand this, we will use the Kayako Adapter as an example. This Adapter requires these parameters:

- **ticket_url:** The full URL of the ticket that is subject to the action. All content of the ticket is used.
- **package_id *(optional if package_name is provided)*:** ID of the Package to be executed.
- **package_name *(ignored if package_id is provided)*:** Optionally, provide the name of the Package instead.

When invoked, the Kayako Adapter retrieves the relevant ticket details from Kayako, formats them into a usable string, and then calls the Core Adapter with these parameters:

- **Package_id or package_name:** The same value passed into the Kayako Adapter.
- **andon_cord_skill_id:** Defaults to the `pull_andon_cord_kayako` skill.
- **input:** Contains all the relevant data extracted from the ticket and formatted for AI to reference.



Adapters are added in the Admin area and contain information useful for other users. When creating an Adapter, the user will need to ensure that the Core Adapter is invoked in the code. Each Adapter added to the Admin area will generate an API endpoint with the following URL convention:

```
{env}.cu-chulainn.csaiautomations.com/adapters/{adapter_name}
```

3.2 An Example Run

1. **Kayako Adapter Invocation**
   A user or system calls the Kayako Adapter endpoint with two key parameters: the Kayako ticket URL and the name of the “AnswerhubLicenser” Package.
2. **Data Collection**
   The Kayako Adapter retrieves all relevant information from the ticket—comments, notes, side conversations, linked issues, and the requester’s details. It compiles these into a single string.
3. **Core Adapter Call**
   The adapter then sends the compiled ticket data, the Package name, and the Andon Cord skill reference to the Core Adapter. The Core Adapter starts the ECS (execution) process and replies with a success message and an execution ID, which the Kayako Adapter returns to the original caller.
4. **ECS Initialization**
   Within the ECS, Cu Chulainn loads the details of the “AnswerhubLicenser” Package, retrieves its Agents, and gathers any Skills linked to the Agents and Package. These resources are assembled, and the AgentChat session begins.
5. **Agent Swarm Execution**
   The Manager, AuxiliaryAgent, AndonCorder, and any additional Agents collaborate to follow the Package’s workflow. They use the ticket data and assigned Skills dynamically to complete the task, minding any approval requirements and special instructions.

6. **Resolution**
   The Agents conclude the session by sending a final update to the customer (as required by the Package) and post an internal note summarizing the actions taken. If everything is successful, the ECS ends naturally. If they encounter an unexpected situation or error instead, the AndonCorder triggers an emergency stop.
7. **Session Termination and Logging**
   Once the workflow completes or the AndonCorder is pulled, the ECS terminates. Cu Chulainn logs the entire AgentChat for future reference and review.

4. Addendum

This section covers a set of topics that extend beyond the core functionality of Cu Chulainn. It includes administrative settings, environment management practices, details on how the AI loop terminates, and guidelines for naming and logging. These insights are essential for system administrators and advanced users who want to ensure the platform remains well-organized, secure, and easy to maintain.

4.1 Admin Features

Access to the Admin menu is restricted to administrators. If you need admin privileges, refer to the contacts listed at the end of this document. The Admin menu includes the following areas:

- **Core Items:** A collection of hidden Skills, Agents, Packages, and Prompts that power Cu Chulainn's core functionality. Users should exercise great caution modifying these items, as they are included in every Package and define the system's overall behavior.
- **User Management:** A section where administrators can view, update, or disable user accounts. This is also where roles are managed to control the level of access users have within the platform.
- **Adapters:** A configuration hub for creating and managing Adapters. Administrators can define which parameters each Adapter accepts and the URL endpoint that will be invoked.

4.2 Environment Separation

Cu Chulainn can be deployed in multiple environments, each with its own set of Packages, Agents, and Skills. While the same core architecture underpins every environment, each environment maintains its own Marketplace data, user accounts, and Adapters. This separation allows different teams or use cases to operate independently.

While the ECS core is spun with AWS Fargate for each execution, all API endpoints are tied to an environment-specific domain; for example:

- cs-test.cu-chulainn.csaiautomations.com
- cs-dev.cu-chulainn.csaiautomations.com
- cs.cu-chulainn.csaiautomations.com
- saas.cu-chulainn.csaiautomations.com

4.2.1 Differences Between Environments

- **Packages, Agents, Skills:** Created and managed separately, so changes in one environment don't affect another.
- **Users:** Exist separately in each environment, with roles and access levels defined per environment.
- **Adapters:** Packages, Agents, Skills: Created and managed separately, so changes in one environment don't affect another.
- **Core Items:** Each environment has its own "base" agents, prompts, and Andon Cord skills that can be customized.

4.2.2 Designing a New Environment

When creating a new environment, it's crucial to set up the core items based on what the environment is intended to do:

- **Task Prompt**
  The main instruction given to the swarm. It includes the following placeholders:
  - `{{content}}` – The string received from the Core Adapter.
  - `{{package_workflow}}` – The Package's workflow.
  - `{{approval_notes}}` – The Package's approval notes.
  - `{{special_instructions}}` – The Package's special instructions.
  - `{{team_members_str}}` – Information about the agents in the session.

- **Manager Agent**
  Acts as the team leader. The prompt can use:
  - `{{jsons.package.packageWorkflow}}`
  - `{{jsons.package.packageApprovalNotes}}`
  - `{{jsons.package.packageSpecialInstructions}}`
  - `{{auxiliary_skill_names_and_descriptions_for_manager}}` – Skills available to the AuxiliaryAgent, pulled from the Package's skills.
  - `{{agent_list_for_manager}}` – List of agents in the session.

- **AndonCorder**
  Handles pulling the Andon Cord on critical errors. Its behavior can be tailored to each environment's needs.

- **AuxiliaryAgent**
  This agent uses the skills directly added to the Package. Its description references the following placeholder:
  - `{{auxiliary_skill_names}}`

  The system prompt uses this placeholder:
  - `{{auxiliary_skill_names_and_descriptions}}`.

- **BaseHelperAgent**
  A template for other agents. Its description includes the following placeholders:
  - `{{agent_skill_names}}`
  - `{{agent_description}}`

  The prompt uses these placeholders:
  - `{{agent_prompt}}`
  - `{{agent_skill_names_and_descriptions}}`.

- **Andon Cord Skills**
  Specifies which skill the AndonCorder calls to exit the loop when encountering an error. These skills must return the necessary termination strings (`TASK_COMPLETE`, `ANDON_CORD_PULLED` or `ERROR_PULLING_ANDON_CORD`) and should interact with external systems as needed.

- **Adapters**
  While every environment comes with its own Core Adapter, it is generally necessary to create specialized adapters for the user's needs.

4.3 Exiting the AI Loop

Since Cu Chulainn uses AutoGen AgentChat, other than by reaching the message limit, the AI loop is exited when one of the termination strings appears as a message. In Cu Chulainn, there are 4 termination strings that can be used:

- `TASK_COMPLETE`: Corresponds to a successfully completed loop.
- `ANDON_CORD_PULLED`: Results from the system pulling the Andon Cord.
- `ERROR_PULLING_ANDON_CORD`: Only appears when the system encounters an error during the Andon Cord pulling process.

These strings should be returned by the relevant skills that are intended to close the loop. In our base environment, for example, the `TASK_COMPLETE` is potentially returned by the KayakoAgent's wrap_up_ticket skill, while `ANDON_CORD_PULLED` and `ERROR_PULLING_ANDON_CORD` are returned by the `pull_andon_cord_kayako` skill. Make sure to take this into account when you create any new agents or adapters that should have this functionality.

4.3.1 Andon Cord Skills

While pulling the Andon Cord is a function that is always undertaken by the Andon Corder agent, it is possible to configure the skill that it should invoke in this event. This allows for configuring the exact mechanism that will be triggered upon pulling the Andon Cord, such as starting a separate flow, or logging the error to another system.

Andon Cord skills should always return one of the following strings in their success or failure messages:

- `ANDON_CORD_PULLED`: When the process of pulling the Andon Cord has completed successfully. The execution will be tagged as having been Andon Corded for human review as required.
- `ERROR_PULLING_ANDON_CORD`: When issues have occurred during the Andon Cord process that should tag the execution as errored for human review.

Andon Cord skills should not be added to an agent or Package, and are instead specified when starting a Package execution in one of the following parameters:

- **andon_cord_skill_id *(optional if andon_cord_skill_name is provided)*:** References the Andon Cord skill to be used to abort if errors arise. These skills are also configured under the Admin settings.
- **andon_cord_skill_name *(ignored if andon_cord_skill_id is provided)*:** Optionally, reference the Skill name instead.

Note that the system will automatically add this skill to the Andon Corder and instruct it on how it should be used, so that no further context is required by default.

4.4 Access Control

Cu Chulainn provides three levels of visibility and edit permissions for each Skill, Agent, or Package. Edit permissions cannot exceed view permissions—no one can edit an entity they aren't allowed to view.

- **Owner Only:** Only the entity's owner can view or edit the entity.
- **Allowed Users:** A provided list of users can view or edit the entity. Users with edit access can add more users to this list.
- **All Users:** Everyone in the environment can view or edit the entity.

Administrators can view and edit all entities, regardless of their assigned access level. They can also change an entity's owner, modify user roles, and access and update core items, which are hidden from non-admin users.

4.5 Naming Conventions

Cu Chulainn enforces certain naming rules for Skills, Agents, and Packages to keep them organized and consistent:

- **Skills:** Use `snake_case` and include the system name when possible.
  Example: `get_answerhub_license_details` instead of `get_license`.

- **Agents:** Use `CamelCase`, referencing the associated scope or product.
  Example: `KayakoAgent`.

- **Packages:** Allow flexible naming, though `CamelCase` is recommended.
  Example: `JiveHostedRollingRestart`.

- **Andon Cord Skills:** Follow `snake_case` with the form `pull_andon_cord_{system_name}`.
  Example: `pull_andon_cord_kayako`.

4.6 Logging

While CloudWatch logs exist for all elements that offer detailed runtime information for troubleshooting and performance monitoring, Cu Chulainn also stores information in dedicated logs that can be much more useful for understanding the execution flow:

Core Logs

- **AI Logs:** These logs capture the entire agent conversation, including skill usage. They reside in S3 under `cu-chulainn-files/agent-conversation-logs/`.
- **CSV Logs:** These logs record all actions at the Core level. They are stored in S3 under `cu-chulainn-files/csv-logs/` and can be accessed using Athena in the `cu_chulainn` database.

Marketplace Logs - WIP

4.7 Codebase

- Cu Chulainn Core
- Cu Chulainn Core Marketplace

5. Bugs and Feature Requests

If you find any bugs or have a suggestion for a feature, please add a card to either one of the following Notion projects:

- Bug Reports

- Feature Requests

6. Contacts

For urgent requests or if an outage has been reported for Cu Chulainn, please contact any of the following contacts:

- Fernando Perez
- Ahmet Metin Birgili
- Sanket Ghia

7. FAQ

- **Should I put information about processes in the Agent prompts?**
  No: processes should be introduced in the 'workflow' area of the Package. This is because the Manager, who is in charge of ensuring that the Agents follow the package's instructions, doesn't have access to each agent's prompt. Instead, use Agent prompts to give them context about their abilities, or fine-tune their responses; for example:

  ```
  Make sure that the exact error is recorded in the reason field
  when calling the Andon Cord.
  ```

- **Can I have Cu Chulainn schedule an action for later?**
  Currently, Cu Chulainn only supports direct invocation by third-party systems. This means that, at present, it is not possible to schedule tasks natively.
  - If you are using a system like Zendesk, Kayako, or Jira, consider using on-hold timers to trigger the same or a new package.
  - Note that a scheduler has been added as a feature request and will eventually be added to Cu Chulainn.

- **Can I have Cu Chulainn oversee a task?**
  While Agent chats occur in real time, and they can repeatedly call skills to check on the status of a task, Cu Chulainn presently restricts the length of time for which a package can run to prevent a variety of issues and unwanted costs. Because of this, it is instead recommended that you check on running tasks periodically, and keep a log somewhere that Cu Chulainn can write to and review.

- **Can Cu Chulainn run multiple packages at a time?**
  Cu Chulainn runs in ECS, which means you can have any number of containers running different packages at the same time; however, each container will only be able to run a single package. If you have use cases that require multiple packages to be run, consider starting a new package as part of a previous package's workflow via a skill, or splitting the scenario into its distinct intents.