Unit 1 - Planning the AI Agent

- 1.1 Unit Introduction
- 1.2 How to identify tasks that need agents
- 1.3 Types of agents
- 1.4 Identify which information the agent needs
- 1.5 Identify and plan the tools
- 1.6 Wrap up

Unit 1
Planning the AI Agent

1.1 Unit Introduction

You are at the first unit of the course Design and build a production-ready AI agent.

Now that you know how to build and use an AI agent in Make, it's time to learn **how to plan and design an agent you can deploy in your organization**.

In this unit you will learn:

- how to identify a good use case for your agent
- how to choose the right way for users to interact with your agent
- how to plan the knowledge and the tools your agent needs

Let's begin!

Continue to 1.2: How to identify tasks that need agents

1.2 How to identify tasks that need agents

In the course AI, AI agents and automation, you learned when to use traditional automation and when to use agentic automation.

REVIEW COURSE

Now you will learn how to **identify which tasks in your organization are good candidates for AI agents**.

Start by learning what not to do.

Continue to 1.2.1: When you shouldn't use an agent

1.2.1 When you shouldn't use an agent

Before you identify good use cases, you need to **understand what to avoid**.

Learning to recognize these situations helps you **focus your efforts on tasks where agents truly add value**.

You shouldn't use agents for:

1. Processes with simple logic
2. Tasks requiring guaranteed outcomes
3. High-stakes decisions without review
4. Vague goals without clear success criteria
5. Tasks without tool access

Let's explore each one.

1: Processes with simple logic

1

1: Processes with simple logic

Do not build agents for processes you can fully capture with true/false rules.

If every step in your process follows the same order and requires the same actions every time, **use traditional automation in Make** instead.

For these kinds of tasks, the agent adds complexity without adding value.

Sending scheduled emails does not need an agent.

You can build a scenario to send email messages and schedule it to run at specific times.

2

2: Tasks requiring guaranteed outcomes

Do not build agents when you need guaranteed, predictable outcomes.

Agents introduce variability because they reason about situations differently: the same input might produce different outputs depending on the context. This flexibility is valuable for complex tasks, but it creates problems when you need **consistency**.

For these kind of tasks, **use traditional automation in Make** instead.

If your process requires **approving expense reports under $100**, traditional automation will approve every report under that amount.

An agent might consider additional factors like spending patterns or budget status, which creates unpredictable outcomes.

3: High-stakes decisions without review

3

3: High-stakes decisions without review

Avoid agents for high-stakes decisions where errors could be extremely costly.

You need to consider **what happens if the agent makes a mistake**. Try asking these questions:

- Could it lose a major customer?
- Could it create legal liability?
- Could it damage your reputation?

For these kind of tasks, build a **hybrid approach** where the agent analyzes the situation and recommends a decision, then a human reviews and approves it.

For example, if an agent approves a fraudulent **$50,000 transaction**, the consequences could be severe.

4: Vague goals without clear success criteria

4

4: Vague goals without clear success criteria

Avoid agents when you have vague goals for what you want them to do.

The agent needs a **specific goal to understand when it has completed the task successfully**.

Before you build an agent, you must be able to answer this question:

How will I know if this agent succeeded?

If you cannot answer that question clearly, you do not have a good use case yet.

Improve customer satisfaction is too vague. The agent doesn't know whether to respond faster, provide more detailed answers, or escalate issues.

Resolve customer support tickets within 2 hours with a satisfaction rating above 4 stars gives the agent clear targets to work toward.

5: Tasks without tool access

5

5: Tasks without tool access

Do not build agents when you cannot provide tool access.

An agent without tools can **only generate text**, it **cannot take actions** or retrieve information from your applications.

For these kind of tasks, **ensure you can provide the necessary tools before building the agent**:

- identify which systems the agent needs to access and
- verify you can connect those systems through Make modules or APIs

If you cannot provide tool access, the use case is not ready for an agent yet.

An agent helping with **customer refunds** needs **access to your order system** to look up purchase history and your **refund system** to process refunds.

Without these tools, the agent can only write suggestions that a human must execute manually.

Now you know when you should not build an agent and when your use case is not ready.

Next, let's look at when it makes sense to use an agent.

Continue to 1.2.2: When you should use an agent

1.2.2 When you should use an agent

Good agent use cases are too complex for simple automation but not too risky to automate completely. They are tasks that require intelligence and judgment but follow repetitive patterns.

For example, **lead research** requires you to investigate and decide whether a company is a good fit.

You read websites, understand their business model, and assess alignment with your product, all activities that require judgment. While each lead is different, you follow the **same investigation and decision-making process for every lead**.

Look for these characteristics when identifying good agent candidates:

Both conversation and action

The agent needs to **gather information through conversation** and then **take action** based on

the information it gets.

👉 For example, **customer support** requires asking clarifying questions to understand the issue (conversation) and then processing a refund or updating account settings (action)

Complex decision-making

Tasks with decision criteria you can't fully capture in a step-by-step process. These tasks

involve:

- **Multiple variables** that affect the outcome (purchase history, customer sentiment, refund policies, special circumstances).
- **Shifting context** where the same inputs require different approaches depending on the situation.
- **Edge cases and exceptions** that don't fit standard rules and require judgment rather than rigid logic.

👉 For example, **customer refund approvals** need to consider many factors: a $50 refund request from a loyal customer who's never complained needs different handling than the same request from someone with a history of frequent returns

Unstructured data

Processes involving **natural language**, **documents**, **images**, or **messy data** like emails and chat

logs.

👉 For example, **insurance claims processing** requires interpreting medical reports written in different styles. Each document is different and requires the agent to extract relevant information and make sense of it

High ROI tasks

Tasks that offer **significant time savings or cost reductions**.

ROI (Return on Investment) is a value used to measure the benefit you gain compared to the cost you invest.

👉 For example, **lead research** requires manually reading company websites, understanding business models, and assessing fit with your product. This process might take 30 minutes per lead. Automating this research for 100 leads per week reclaims hundreds of hours that your team can spend on higher-value activities like closing deals

Note that if the task follows the characteristics above but is business-critical (for example, processing refunds), you can still use an agent.

Plan to include a step where the user must review and approve the agent's decision for final approval rather than letting the agent act autonomously.

Continue to 1.2.3: Find use cases in your organization

1.2.3 Find use cases in your organization

Let's apply what you have learned so far to identify processes in your organization by following this systematic process.

Catalog existing processes

Document current workflows throughout your organization, paying special attention to processes with manual steps and activities people repeat frequently, pain points where people struggle, and bottlenecks that slow down work.

Create a list of candidate processes. For each one, write down **what the process does**, **who performs it**, **how often**, and the **time and cost involved**.

Evaluate against criteria

Evaluate each process against the criteria you learned earlier:

- Does it involve **complex decision-making** with multiple variables, shifting context, and edge cases?
- Does it involve **unstructured data** like emails, documents, or images?
- Can you define **clear success criteria**?
- Can you provide **tools** through Make modules or APIs?
- If business-critical, can you include a **human in the loop for review**?

Eliminate mismatches

Remove processes that **don't fit** according to the criteria above.
Eliminate processes that follow clear, fixed steps and conditions, for these use traditional Make automation instead.
Also remove processes that need guaranteed outcomes, high-stakes decisions without review, vague goals, or tasks where you cannot provide necessary tools.

Calculate potential ROI

For each remaining process, **estimate current costs including time**, **errors**, **and delays**. Study **implementation costs** including building and testing in Make, and **compare benefits against costs**.
Target processes that offer significant time savings and assess feasibility based on available data and integrations. Also, consider organizational readiness and potential resistance.

Start small with quick wins

Begin with **lower-risk opportunities** with **clear metrics for success**. Build capabilities and confidence with each implementation while learning from your first agent before building the second.
Remember that the best agent implementations start with a **clear problem to solve**, so look for problems that AI agents can solve effectively.

Continue to 1.2.4: Define the agent's objective

1.2.4 Define the agent's objective

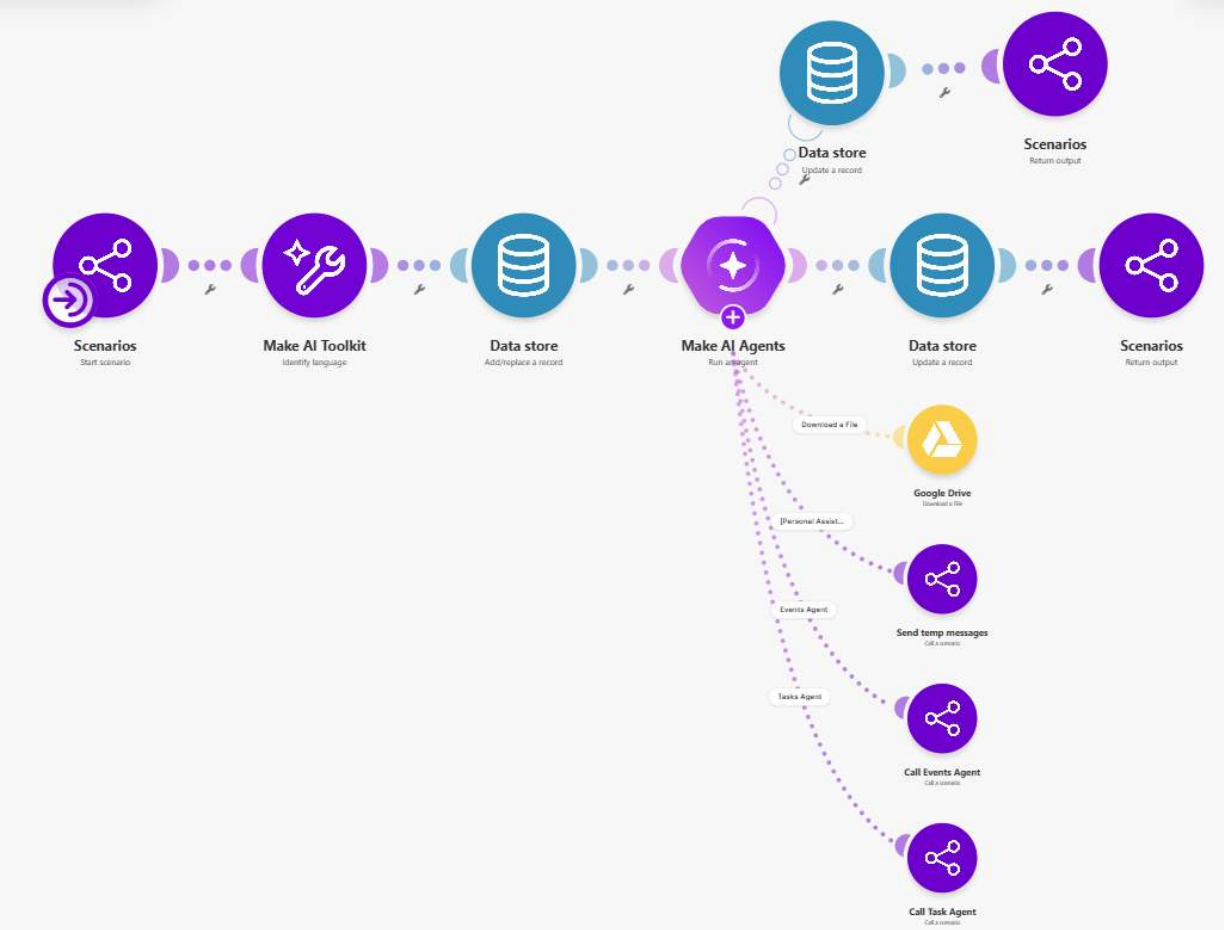
The next thing you need to take into account when planning your agent is defining its objective.

From the information from your previous analysis, write down your agent's objective clearly, the **specific goal your agent should achieve**.

For the agent to work properly, the objective needs to be narrow and specialized in a single job. Don't create one agent that does everything.

This directly affects the agent's accuracy and reliability. A **specialized agent** becomes **expert** at its objective, makes better decisions, and works faster.

If you have more than one objective, build multiple specialized agents with a single objective in Make.



Use a **main agent** to **coordinate** these specialized agents.

The main agent acts as the **decision-maker**.

It receives the user's request and routes it to the appropriate specialized agent.

Each specialized agent does its specific job.

Now you know how to identify good agent use cases in your organization.

Next, you will learn how to plan the type of agent that fits each use case.

Continue to 1.3: Types of agents

1.3 Types of agents

The next step in planning is choosing the agent type.

Agent type determines how users will access and interact with your agent.

This choice affects how you build your agent in Make and what considerations you need to address.

There are three main types of agents:

- **Chatbot agents** that allow direct conversation with users.
- **Autonomous agents** that run on schedules or triggers without direct user interaction.
- **Agents embedded in workflows** that handle specific reasoning steps within larger automated processes.

The type you choose depends on your use case and how users need to interact with the agent.

Let's look at each one in more detail and at their different implementation requirements.

Continue to 1.3.1: Chatbot agents

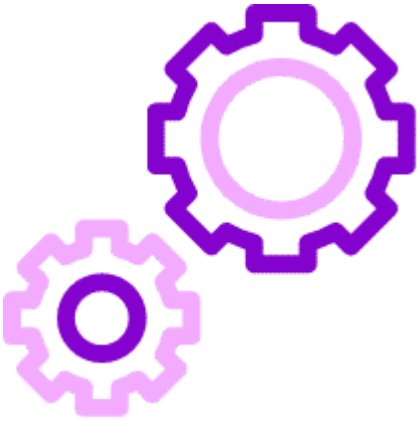
1.3.1 Chatbot agents

Chatbot agents allow direct interaction with users through conversation.

Users ask questions or make requests, and the **agent responds in real-time**.

Users access chatbot agents through **messaging apps** like Slack, Discord, Telegram, or other web applications that connect to your Make scenario. The user types a message, and the agent responds directly in the **same interface**.

You build chatbot agents using **webhook scenarios** in Make that receive messages from the chat interface and trigger the **AI agent** to process them. The scenario processes the message, runs the AI agent, and sends the response back to the user.



When should you use chatbot agents?

See the answer

Use them for customer support and user-facing applications.

When you build a chatbot agent in Make, you need to manage several key aspects:

💬 Conversation history

The agent needs **access to previous messages in the conversation** to help it understand what the user is referring to and maintain continuity. Store and retrieve conversation history using **conversation IDs**.

⚡ Real-time response handling

The agent must respond when the user sends a question. Use a **webhook scenario** that the user message triggers.

🔓 User identification

You need to identify **who** is talking to the agent to access the right customer data and personalize responses.

Keep in mind the security considerations discussed in the course Information management and security when building chatbot agents.

A **customer support chatbot agent** helps customers **check order status** and **request refunds**.

A webhook scenario in Make receives a message from a website chat widget. It then identifies the user and retrieves the conversation history. Finally, the scenario calls the AI Agent module with relevant tools and sends the response back. The agent accesses the order management and refund systems through Make modules to take actions during the conversation.

Continue to 1.3.2: Autonomous agents

1.3.2 Autonomous agents

Autonomous agents run on schedules or triggers without direct user interaction.

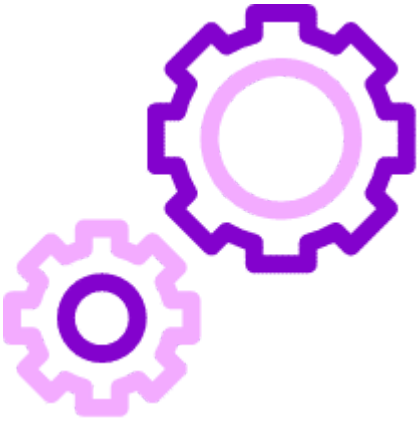
Users do not interact directly with autonomous agents.

The agent executes tasks **automatically** based on **time-based schedules or system events** and sends notifications, reports, or updates when tasks complete.

You build autonomous agents in Make using:

- **Scheduled scenarios** run at specific times (daily, hourly, weekly).
- **Event-triggered scenarios** run when something happens in your systems, like a new order arriving or a deadline approaching.

The scenario runs the AI agent module with the necessary context and tools, processes the information, and returns the results.



When should you use autonomous agents?

See the answer

Use them for recurring tasks that require intelligence but don't need human initiation.

Autonomous agents work well for tasks like **monitoring**, **analysis**, and **notifications that the agent sends automatically**. They can review data, identify issues or opportunities, and take action to generate reports, update records, or send notifications.

When you build an autonomous agent in Make, you need to manage several key aspects:

🕹️ Trigger configuration

Determine when the agent should run by setting up schedules for time-based execution or configuring event triggers for webhook scenarios. Consider how often the agent needs to run and whether timing matters for the task.

📌 Data context

Provide the agent with the data it needs for each execution. The agent doesn't have conversation history from previous runs unless you explicitly store and retrieve it using the conversation ID.

📤 Output handling

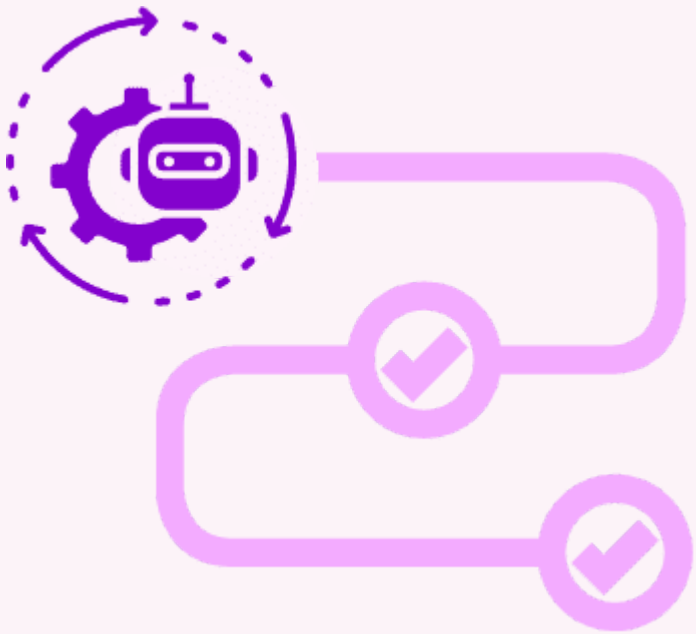
Decide where results should go. Configure the agent to: send notifications, update databases, create tasks, trigger other workflows based on findings, or return a custom response upon completion.

A **lead qualification agent** runs every morning to review new leads from the previous day. A scheduled scenario in Make runs at 8 AM daily and retrieves new leads from the CRM. It then calls the AI agent module with tools to search company websites and assess fit. Based on the agent's findings, the scenario updates lead scores in the CRM and posts a summary of high-priority leads to the sales team's Slack channel.

Continue to 1.3.3: Agents embedded in workflows

1.3.3 Agents embedded in workflows

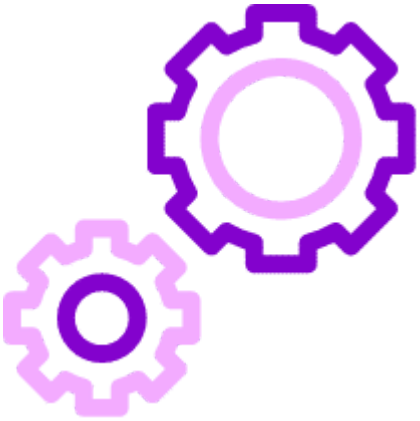
Agents embedded in workflows handle specific reasoning steps within larger automated scenarios.



The agent performs intelligent decision-making at specific points while **traditional automation handles the deterministic steps**.

This type **combines** the reliability of fixed workflows with the flexibility of AI reasoning.

You build embedded agents by adding **AI agent modules** at specific points within your existing Make scenarios.



When should you use agents embedded in workflows?

See the answer

Use them when you have a mostly deterministic process with specific steps that require intelligent decision-making.

When you build an embedded agent in Make, you need to manage several key aspects:

🔗 Integration points

Identify where in your workflow the agent should intervene. Determine what data the scenario passes to the agent and what output it expects back.

🔀 Data flow

Prepare the data that the scenario sends to the agent. Format data appropriately for both the agent's input and the scenario's needs.

⚠️ Exceptions management

Plan what happens if the agent cannot make a decision. Build fallback logic or human escalation into your workflow.

A Make scenario regularly checks for new **expense reports**. When new reports are available, it retrieves the details and passes the data to the the AI agent. The agent reviews the expenses against company policy and recommends approval or rejection, then either auto-approves or routes to a manager for review. The scenario updates the expense system and notifies the employee.

Continue to 1.3.4: Choosing the right agent type

1.3.4 Choosing the right agent type

Choose the agent type based on your use case and how users interact with the agent:

🤖 **Choose chatbot agents when users need to start conversations and get immediate responses.**

This is useful for use cases like **customer support**, where people ask questions and expect real-time answers.

🔔 **Choose autonomous agents when the system should initiate work based on schedules or events.**

This is useful for tasks like **daily reporting**, **monitoring**, or **proactive notifications** that run without human input.

🔗 **Choose embedded agents when you have an existing workflow that needs intelligent decision-making at specific points.**

This is useful for use cases like **approval processes** where most steps are deterministic but some require reasoning.

Now you understand how to choose the right agent type for your use case.

Next, you will learn how to identify and plan the tools your agent needs.

Continue to 1.4: Identify which information the agent needs

1.4 Identify which information the agent needs

The next planning step is identifying which information the agent needs beyond its training data.

In the course Information management and security, you learned that depending on your use case, the agent may need additional context specific to your business.

REVIEW COURSE

For example:

- **Company-specific information**: internal policies, procedures, guidelines, and past case resolutions
- **Domain-specific expertise**: specialized technical or industry knowledge required for your field
- **Frequently updated information**: data that changes regularly like prices, inventory, or schedules
- **Product and service details**: specifications, features, and information about what you offer

---

Planning these knowledge requirements is critical because the agent cannot perform effectively without the right information.

Knowledge comes in different formats and sources.

The format you choose depends on what information you have and how it's organized.

Here are some examples:

Documents	FAQs & knowledge base articles	Databases & structured data

📝

PDFs, Word files, or text files containing policies, procedures, and guidelines.

For example:

- refund policies
- shipping guidelines
- product manuals

📖

Structured question-and-answer content with established answers.

For example:

- troubleshooting steps
- account management guides
- feature explanations

🛢️

Information in databases, spreadsheets, or tables.

For example:

- product catalogs with specifications, pricing, and

Steps to plan which specific knowledge your agent needs and how to organize it:

🎯

Start with your agent's objective

Refer back to the agent's objective you defined earlier. **What information does the agent need to know to accomplish this objective?**

👉 For a customer support agent helping with refunds and shipping questions, the agent needs your refund policy, shipping timeframes, and return procedures.

🚧

Identify specific knowledge gaps

List the questions your agent will need to answer or **decisions** it will need to make. For each one, identify **what information the agent needs that it doesn't have from training data**.

👉 For the customer support agent, questions might include *What is the refund window?* or *How long does international shipping take?* For each question, identify that the agent needs your specific refund policy document and shipping guidelines.

📌

Locate existing information sources

Determine **where this information currently exists in your organization**. Is it in documents, databases, spreadsheets, or knowledge base articles? **Identify the specific files or sources** you'll need to connect.

👉 For the customer support agent, your refund policy might exist as a PDF in Google Drive, shipping guidelines might be in your knowledge base articles in Zendesk, and return procedures might be in a shared document.

🖌️

Organize and prepare your knowledge

Ensure the information you provide is **concise** and **relevant** by **removing unnecessary details** that might confuse the agent and **excluding sensitive data** unless absolutely necessary for the agent's task. Keep information **current** by updating it when changes occur.

👉 For the customer support agent, extract only the customer-facing refund policy sections and remove internal notes about exceptions, approval processes or customer details. Update shipping timeframes when carriers change their schedules.

Pro Tip

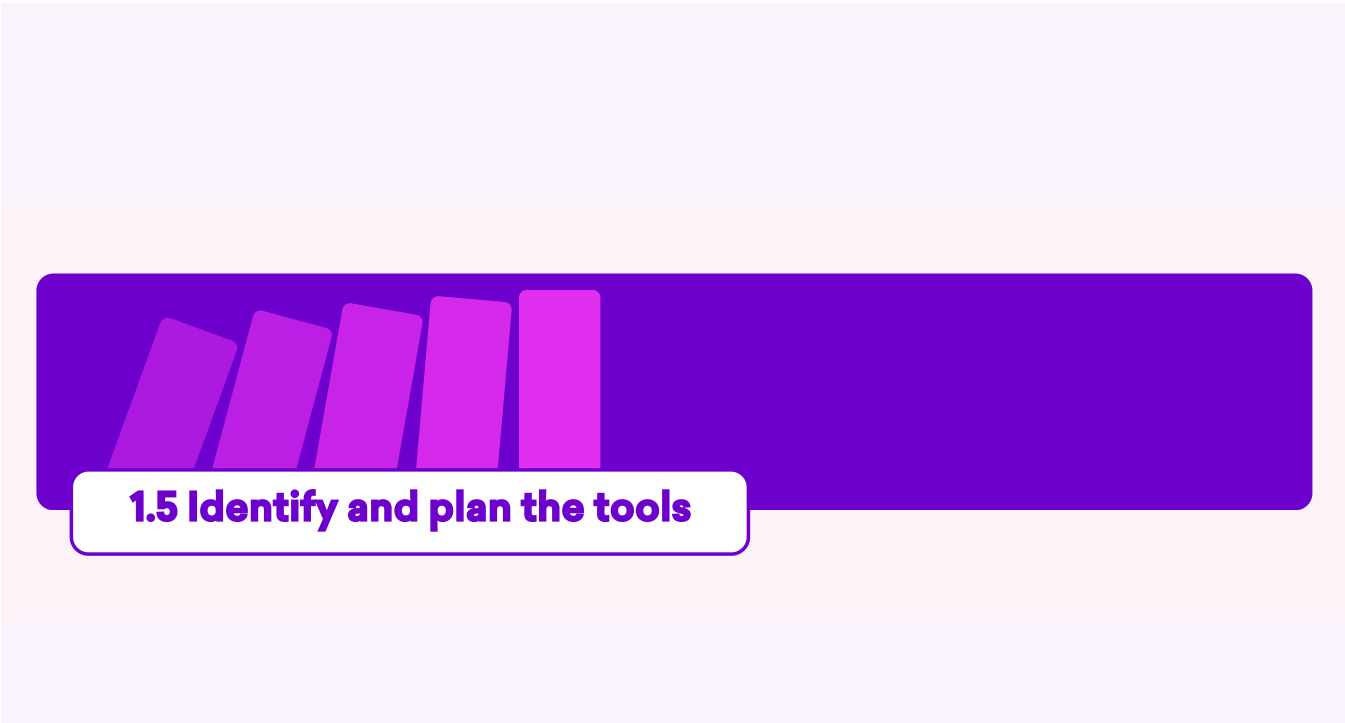
Create a **clear list** showing **what information the agent needs**, **where it currently exists**, what **format** you'll provide it in, and **how you'll keep it updated.**

This becomes your **implementation guide** when you connect knowledge sources to your agent.

Great!

Now you have properly planned your agent and you're ready to build it.

Continue to 1.5: Identify and plan the tools



The last planning step is identifying which tools your agent needs to accomplish its tasks.

Proper tool planning is critical because too few tools limit what the agent can do while too many tools overwhelm it and degrade performance.

Let's look at what you need to consider when planning your agent's tools.

Continue to 1.5.1: Tool categories

1.5.1 Tool categories

Tools fall into three main categories based on what they enable your agent to do.

Understanding these categories helps you identify which tools your agent needs.

Click each one to learn more.

📊 Tools that provide data

Data tools enable agents to **retrieve context and information**.

Use them for:

- database queries
- document retrieval

- search capabilities
- knowledge base access

👉 **Example:**

A customer support agent might use a data tool to find a user's profile information from the CRM.

🎬 Tools that perform actions

Action tools allow agents to **interact with systems and take actions**.

Use them for:

- sending messages
- updating records
- creating content
- managing resources
- initiating processes

👉 **Example:**

A support agent might use an action tool to send a Slack alert to the support team.

🪄 Orchestration tools

Orchestration tools trigger other AI agents in Make when a task requires specialized expertise.

Use them to route specific subtasks to agents that handle them better.

👉 **Example:**

A main agent might use an orchestration tool to trigger a document agent for interpreting complex reports.

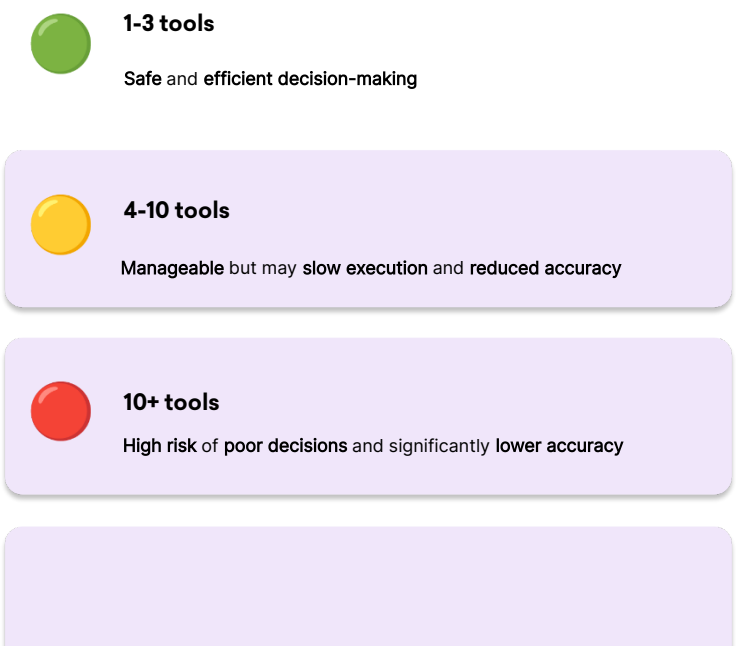
Continue to 1.5.2: How many tools to provide

1.5.2: How many tools to provide

Next, you need to think about how many tools to give your agent.

A high number of tools reduce the agent's decision-making quality because the agent has too many options and can make mistakes choosing the right one.

Follow these guidelines:



For this reason, only build tools that are critical for the agent's tasks.

One strategy to reduce the number of tools is to consider **how much functionality each tool provides**.

There are two types of tools according to how much they do:

- Fine-grained tools perform **one specific operation**, like getting a project ID.
- Coarse-grained tools are Make scenarios that **combine multiple operations into a single tool**. For example you

could create a tool that receives the workspace and project name, finds the project ID, and creates a ticket.

Build coarse-grained tools when possible to keep your tool count low.

If you build fine-grained tools, you need many of them since each performs a single operation. This increases the risk of incorrect sequencing, duplicate calls, or wrong tool selection as the agent orchestrates multiple calls.

Coarse-grained tools, however, reduce the total number of tools you need while allowing you to perform everything you need within a single tool, making it more reliable and efficient since your Make scenario handles all the internal logic.

Use fine-grained tools only when the agent truly needs independent access to each operation.

Creating a task in Asana with **fine-grained tools** requires three separate tool calls: list workspaces, get projects, and create task.

With a **coarse-grained tool**, you build a single Make scenario that accepts the workspace name, project name, and task details, then handles all steps internally. This approach reduces three tools to one.

Continue to 1.5.3: Identifying tools for your agent

1.5.3: Identifying tools for your agent

Follow this process to identify which tools your agent actually needs.

Identify the tasks your agent needs to perform

Start by defining **what the agent needs to do to achieve its objective**. List the **specific tasks and actions** required.

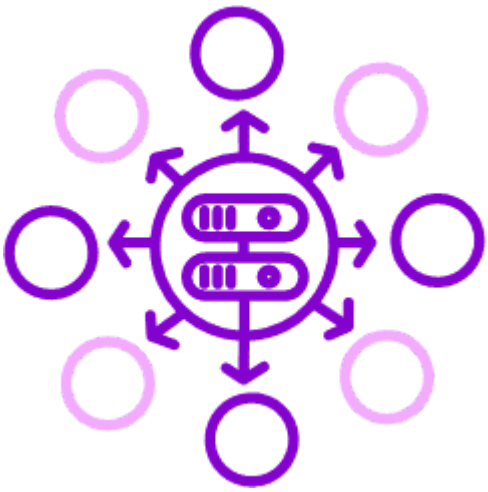
👉 **Example:**
A **marketing lead generation agent** needs to search for contacts in HubSpot, create new contacts, update contact information, create campaigns in Brevo, add contacts to campaigns, notify the marketing team in Slack, and create tasks in Trello for follow-up.

List the systems your agent needs to access

From your task list, identify the **third-party applications** and **systems your agent will connect to**. Create a simple list of all systems involved.

👉 **Example:**

For the marketing agent example, the systems are HubSpot, Brevo, Slack, and Trello.



Identify the actions for each system

For each system, identify the **specific actions your agent needs to perform**: for example create, update, delete, get, or search. You can use a **table** to **organize** this information, especially when you have many actions and systems.

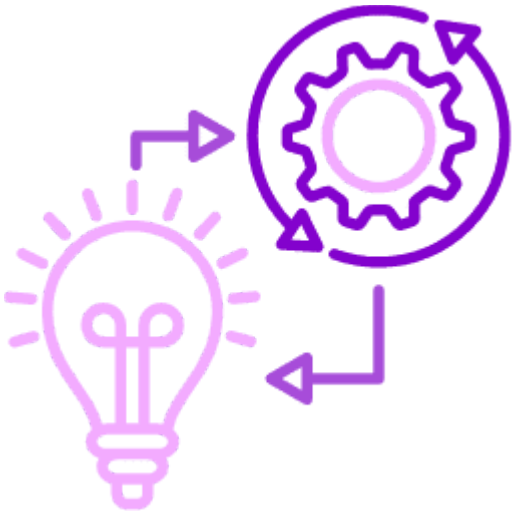
👉 **Example:**

For the marketing agent, in HubSpot you need to search for contacts, create contacts, and get contact details.

Plan your implementation approach

Decide how to implement each action. Check whether Make has modules available for the systems you need and verify you can connect those systems through Make modules or APIs.

👉 **Example:**
For the marketing agent, check Make's app directory for HubSpot. Verify each module supports the specific actions you need, for example, confirm the HubSpot module can search and create contacts.

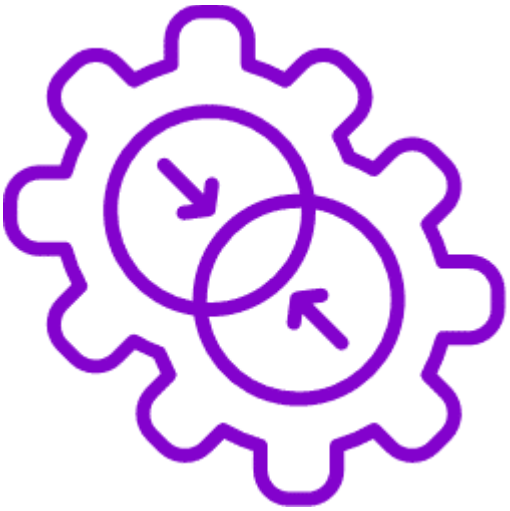


Group related actions into scenario tools

Look for opportunities to **combine related actions** into coarse-grained scenario tools, as you learned earlier. Actions that always happen together should become a single scenario tool rather than multiple separate tools.

👉 **Example:**

For the marketing agent, if you always search for a contact in HubSpot and then retrieve their full details, build a single Get contact information scenario tool instead of separate tools for each action.



Pro Tip

You can create a clear document or table that shows which tools you'll build, which systems they connect to, what actions they perform, and how you'll implement them in Make.

This becomes your **implementation guide** when you start building.

After planning your tools, you need to build them in Make.

To learn best practices for building tools, go to the course Tools and AI agent settings optimization.

GO TO COURSE

Continue to the wrap up for this unit

1.6 Wrap up

1

A **good agent use case** sits between too simple and too risky: don't build agents for tasks you can capture with simple rules. Good candidates involve **complex decision-making with multiple variables**, **unstructured data**, and **clear success criteria**. Define your agent's **objective** as **narrow** and **specialized**: one agent should do one job well.

2

Agent type determines **how users access your agent**. **Chatbot agents** allow direct conversation and respond in real-time, **autonomous agents** run on schedules without user interaction, and **embedded agents** handle

reasoning steps within larger workflows. **Choose based on who initiates the work**: users, systems, or existing processes.

3

Tools connect your agent to external systems while **knowledge** provides context beyond its training data. **Keep tool count low** by building **coarse-grained tools** that **combine multiple operations into single Make scenarios**. Identify what information your agent needs, locate where it exists, and organize it clearly before connecting it to your agent.

Unit complete!

Good job!

By now you should have an understanding of how to:

- Identify good agent use cases
- Choose the right agent type
- Plan the tools and knowledge your agent needs

In the next unit, you'll learn some best practices to build your agent in Make step by step.

make | academy

Mark this task complete to continue to the next unit.