LangChain Docs

Get started › Quickstart

Get started

Quickstart

Copy page

This quickstart takes you from a simple setup to a fully functional AI agent in just a few minutes.

LangChain Docs MCP server

If you're using an AI coding assistant or IDE (e.g. Claude Code or Cursor), you should install the **LangChain Docs MCP server** to get the most out of it. This ensures your agent has access to up-to-date LangChain documentation and examples.

Requirements

For these examples, you will need to:

- **Install** the LangChain package
- Set up a **Claude (Anthropic)** account and get an API key
- Set the `ANTHROPIC_API_KEY` environment variable in your terminal

Although these examples use Claude, you can use **any supported model** by changing the model name in the code and setting up the appropriate API key.

Build a basic agent

Start by creating a simple agent that can answer questions and call tools. The agent will use Claude Sonnet 4.5 as its language model, a basic weather function as a tool, and a simple prompt to guide its behavior.

```
  (input) => `It's always sunny in ${input.city}!`,
  {
    name: "get_weather",
    description: "Get the weather for a given city",
    schema: z.object({
      city: z.string().describe("The city to get the weather for"),
    }),
  }
);

const agent = createAgent({
  model: "claude-sonnet-4-5-20250929",
  tools: [getWeather],
});

console.log(
  await agent.invoke({
    messages: [{ role: "user", content: "What's the weather in Tokyo?" }],
  })
);
```

To learn how to trace your agent with LangSmith, see the **LangSmith documentation**.

Build a real-world agent

Next, build a practical weather forecasting agent that demonstrates key production concepts:

1. **Detailed system prompts** for better agent behavior

Create tools that integrate with external data

6. **Create and run the agent** to test the fully functional agent

Let's walk through each step:

1 Define the system prompt

The system prompt defines your agent's role and behavior. Keep it specific and actionable:

```
const systemPrompt = `You are an expert weather forecaster, wh

You have access to two tools:

- get_weather_for_location: use this to get the weather for a specific loc
- get_user_location: use this to get the user's location

If a user asks you for the weather, make sure you know the location. If yo
```

2 Create tools

Tools are functions your agent can call. Oftentimes tools will want to connect to external systems, and will rely on runtime configuration to do so. Notice here how the `getUserLocation` tool does exactly that:

```
  (input) => `It's always sunny in ${input.city}!`,
  {
    name: "get_weather_for_location",
    description: "Get the weather for a given city",
    schema: z.object({
      city: z.string().describe("The city to get the weather for"),
    }),
  }
);

type AgentRuntime = ToolRuntime<unknown, { user_id: string }>;

const getUserLocation = tool(
  (_, config: AgentRuntime) => {
    const { user_id } = config.context;
    return user_id === "1" ? "Florida" : "SF";
  },
  {
    name: "get_user_location",
    description: "Retrieve user information based on user ID",
  }
);
```

in mind that JSON schemas **won't** be validated at runtime.

Example: Using JSON schema for tool input

```
const getWeather = tool(
  ({ city }) => `It's always sunny in ${city}!`,
  {
    name: "get_weather_for_location",
    description: "Get the weather for a given city",
    schema: {
      type: "object",
      properties: {
        city: {
          type: "string",
          description: "The city to get the weather for"
        }
      },
      required: ["city"]
    },
  }
);
```

3 Configure your model

Set up your **language model** with the right parameters for your use case:

```
  { temperature: 0.5, timeout: 10, maxTokens: 1000 }
);
```

Depending on the model and provider chosen, initialization parameters may vary; refer to their reference pages for details.

4 Define response format

Optionally, define a structured response format if you need the agent responses to match a specific schema.

```
const responseFormat = z.object({
  punny_response: z.string(),
  weather_conditions: z.string().optional(),
});
```

```
import { MemorySaver } from "@langchain/langgraph";

const checkpointer = new MemorySaver();
```

In production, use a persistent checkpointer that saves message history to a database. See **Add and manage memory** for more details.

6 Create and run the agent

Now assemble your agent with all the components and run it!

```
  systemPrompt: systemPrompt,
  tools: [getUserLocation, getWeather],
  responseFormat,
  checkpointer,
});

// `thread_id` is a unique identifier for a given conversation.
const config = {
  configurable: { thread_id: "1" },
  context: { user_id: "1" },
};

const response = await agent.invoke(
  { messages: [{ role: "user", content: "what is the weather outside?" }] },
  config
);
console.log(response.structuredResponse);
// {
//   punny_response: "Florida is still having a 'sun-derful' day ...",
//   weather_conditions: "It's always sunny in Florida!"
// }

// Note that we can continue the conversation using the same `thread_id`.
const thankYouResponse = await agent.invoke(
  { messages: [{ role: "user", content: "thank you!" }] },
  config
);
console.log(thankYouResponse.structuredResponse);
// {
//   punny_response: "You're 'thund-erfully' welcome! ...",
//   weather_conditions: undefined
// }
```

LangChain Docs

Get started › **Quickstart**

LangChain Docs

Get started › Quickstart

```
// Define system prompt
const systemPrompt = `You are an expert weather forecaster, who speaks in

You have access to two tools:

- get_weather_for_location: use this to get the weather for a specific loc
- get_user_location: use this to get the user's location

If a user asks you for the weather, make sure you know the location. If yo

// Define tools
const getWeather = tool(
  ({ city }) => `It's always sunny in ${city}!`,
  {
    name: "get_weather_for_location",
    description: "Get the weather for a given city",
    schema: z.object({
      city: z.string(),
    }),
  }
);

type AgentRuntime = ToolRuntime<unknown, { user_id: string }>;

const getUserLocation = tool(
  (_, config: AgentRuntime) => {
    const { user_id } = config.context;
    return user_id === "1" ? "Florida" : "SF";
  },
  {
    name: "get_user_location",
    description: "Retrieve user information based on user ID",
    schema: z.object({}),
  }
);
```

```
);

// Define response format
const responseFormat = z.object({
  punny_response: z.string(),
  weather_conditions: z.string().optional(),
});

// Set up memory
const checkpointer = new MemorySaver();

// Create agent
const agent = createAgent({
  model,
  systemPrompt,
  responseFormat,
  checkpointer,
  tools: [getUserLocation, getWeather],
});

// Run agent
// `thread_id` is a unique identifier for a given conversation.
const config = {
  configurable: { thread_id: "1" },
  context: { user_id: "1" },
};

const response = await agent.invoke(
  { messages: [{ role: "user", content: "what is the weather outside?" }]
  config
);
console.log(response.structuredResponse);
// {
//   punny_response: "Florida is still having a 'sun-derful' day! The suns
//   weather_conditions: "It's always sunny in Florida!"
```

```
  config
);
console.log(thankYouResponse.structuredResponse);
// {
//   punny_response: "You're 'thund-erfully' welcome! It's always a 'breez
//   weather_conditions: undefined
// }
```

To learn how to trace your agent with LangSmith, see the **LangSmith documentation**.

Congratulations! You now have an AI agent that can:

Understand context and remember conversations

Use multiple tools intelligently

Provide structured responses in a consistent format

Handle user-specific information through context

Maintain conversation state across interactions

LangChain Docs

Get started › **Quickstart**

LangChain Docs

Resources

Forum

Changelog

LangChain Academy

Trust Center

Company

About

Careers

Blog

Powered by mintlify

LangChain Docs

Get started › **Quickstart**