Quickstart for AI Configs

Overview

This topic explains how to get started with the LaunchDarkly AI Configs product.

You can use AI Configs to customize, test, and roll out new large language models (LLMs) within your generative AI applications. AI Configs may be right for you if you want to:

- Manage your model configuration and messages outside of your application code
- Upgrade your app to the newest model version, then gradually release to your customers
- Start using a new model provider, and progressively move your production traffic to the new provider
- Compare the performance of different messages and models

With AI Configs, both the messages and the model evaluation are specific to each end user, at runtime. You can update your messages, specific to each end user, without redeploying your application.

Working with AI Configs is available to members with a role that allows AI Config actions. The LaunchDarkly Project Admin, Maintainer, and Developer project roles, as well as the Admin and Owner base roles, include this permission.

Follow the steps below to incorporate AI Configs into your app, or use the in-app onboarding experience to set up your first AI Config directly in the LaunchDarkly UI.

If you'd prefer to learn from an example that's built on a specific generative AI application, read one of our guides:

You can use AI Configs with any model provider, including Gemini, OpenAI, and Anthropic. To learn how to use OpenAI or Anthropic models with AI Configs, read the examples in Customize the AI Config. To learn how to use Gemini models, including how to cache prompts, read Use Gemini prompt caching with AI Configs.

Try the LaunchDarkly sandbox

You can also use the LaunchDarkly demo sandbox.

Step 1, in your app: Install an AI SDK

First, install one of the LaunchDarkly AI SDKs in your app:

.NET AI SDK | Go AI SDK | Node.js AI SDK (TypeScript) | Python AI SDK | Ruby AI SDK

```
$ Install-Package LaunchDarkly.ServerSdk
> Install-Package LaunchDarkly.ServerSdk.Ai
```

Next, import the LaunchDarkly AI client in your app and initialize a single, shared instance of it:

.NET AI SDK | Go AI SDK | Node.js AI SDK (TypeScript) | Python AI SDK | Ruby AI SDK

```
using LaunchDarkly.Sdk.Server.Ai;
using LaunchDarkly.Sdk.Server.Ai.Adapters;
using LaunchDarkly.Sdk.Server.Ai.Config;

var baseClient = new LdClient(Configuration.Builder("sdk-key-123").Start
var aiClient = new LdAiClient(new LdClientAdapter(baseClient));
```

project and environment. They are available from **Project settings**, on the **Environments** list in the LaunchDarkly UI. To learn more, read Keys.

Then, set up the context. Contexts are the people or resources who will encounter generated AI content in your application. The context attributes determine which variation of the AI Config LaunchDarkly serves to the end user, based on the targeting rules in your AI Config. If you are using template variables in the messages in your AI Config's variations, the context attributes also fill in values for the template variables.

Here's how:

.NET AI SDK | Go AI SDK | Node.js AI SDK (TypeScript) | Python AI SDK | Ruby AI SDK

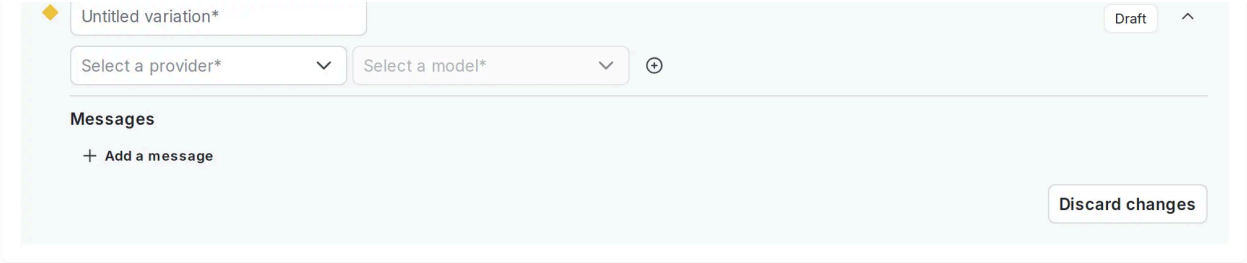
```
var context = Context.Builder("example-user-key")
  .Name("Sandy")
  .Build();
```

Step 2, in LaunchDarkly: Create an AI Config

Next, create an AI Config in LaunchDarkly:

1. Click **Create** and choose **AI Config**.
2. In the "Create AI Config" dialog, give your AI Config a human-readable **Name** and, optionally, a **Maintainer**.
3. Click **Create**.

The empty **Variations** tab of your new AI Config displays:



The Variations tab of a newly-created AI Config.

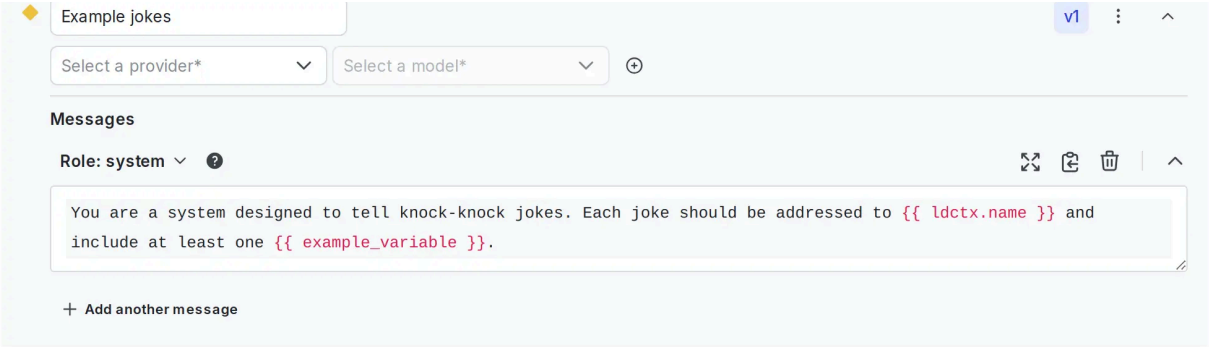
Then, create a variation. Every AI Config has one or more variations. Each variation includes a model configuration and, optionally, one or more messages.

Here's how:

1. In the create panel in the **Variations** tab, replace "Untitled variation" with a variation **Name**. You'll use this to refer to the variations when you set up targeting rules, below.
2. Click **Select a model** and choose the model to use.
   - LaunchDarkly provides a list of common models, and updates it frequently.
   - You can also choose **+ Add a model** and create your own. To learn more, read Create AI model configurations.
3. (Optional) Select a **message role** and enter the message for the variation. If you'd like to customize the message at runtime, use `{{ example_variable }}` or `{{ ldctx.example_context_attribute }}` within the message. The LaunchDarkly AI SDK will substitute the correct values when you customize the AI Config from within your app.
   - To learn more about how variables and context attributes are inserted into messages at runtime, read Customizing AI Configs.
4. Click **Review and save**.

You can also select **Import from playground** in your variation to bring your model, model parameters, and messages from an external playground.

Here's an example of a completed variation:



The Variations tab of an AI Config with one variation.

Expand to copy variation message

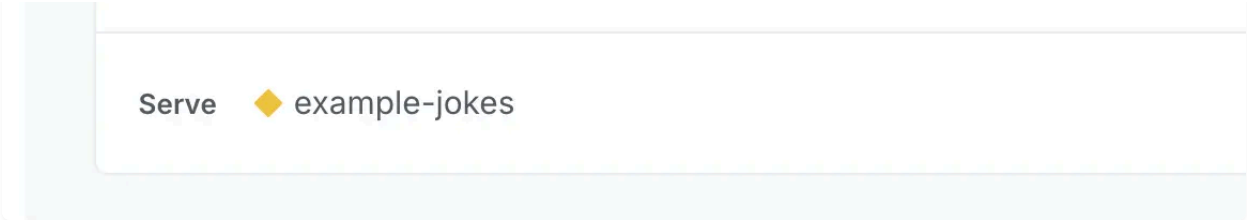
Step 3, in LaunchDarkly: Set up targeting rules

Next, set up targeting rules for your AI Config. These rules determine which of your customers receive a particular variation of your AI Config.

To specify the AI Config variation to use by default when the AI Config is toggled on:

1. Select the **Targeting** tab for your AI Config.
2. In the "Default rule" section, click **Edit**.
3. Configure the default rule to serve a specific variation.
4. Click **Review and save**.

Here is an example of a default rule:



An AI Config's default rule.

The AI Config is enabled by default. After you've added code to your application to pull the customized messages and model configuration from LaunchDarkly, your AI Config will be active.

When an end user opens your application, they'll get the AI Config variation you've defined, either in the default rule or in a custom targeting rule. Your app's AI content generation will use the model and messages from the AI Config variation, and the messages will be customized for each end user.

When you're ready to add more AI Config variations, come back to this step and set up additional targeting rules. If you are familiar with LaunchDarkly's flag targeting, the process is very similar: with AI Configs, you can target individuals or segments, or target contexts with custom rules. To learn how, read Target with AI Configs.

You can release variations gradually

You can release a new variation gradually using a guarded rollout. Guarded rollouts reduce risk and let you monitor key metrics as the change rolls out.

Step 4, in your app: Customize the AI Config, call your generative AI model, track metrics

Now that your AI Config is set up, you can use it in your app:

1. Customize the AI Config: First, use the `config` function from the LaunchDarkly AI SDK to customize the AI Config. The `config` function returns the customized

2. Call your generative AI model, track metrics: Then, call your generative AI model, passing in the result of the `config` function. In the LaunchDarkly AI SDKs, you use a `track[Model]Metrics` function to record metrics from your AI model generation. This function takes a completion from your AI model generation.

Customize the AI Config

In your code, use the `config` function from the LaunchDarkly AI SDK to customize the AI Config. **You need to call `config` each time you generate content from your AI model.**

The `config` function returns the customized messages and model configuration along with a tracker instance for recording metrics. Customization means that any variables you include in the AI Config variation's messages have their values set to the context attributes and variables you pass to the `config` function. Then, you can pass the customized messages directly to your AI. You should also set up a fallback value to use in case of an error, and make sure to handle the fallback case appropriately in your application.

Here's how:

.NET AI SDK | Go AI SDK | Node.js AI SDK | Python AI SDK | Ruby AI SDK

```
var fallbackConfig = LdAiConfig.New()
  .SetEnabled(false)
  .Build()

var tracker = aiClient.Config(
  "ai-config-key-123abc",
  context,
  fallbackConfig,
  new Dictionary<string, object> {
    { "example_variable", "puppy" }
  }
);

// Based on the example AI Config variation shown in step 2,
// tracker.Config.Messages[0].Content will be:
```

Call your generative AI model, track metrics

Next, make a call to your generative AI model and pass in the result of the `config` function.

Use one of the `track[Model]Metrics` or `TrackRequest` functions to record metrics from your AI model generation. This function takes a completion from your AI model generation. Remember that you need to call `config` each time you generate content from your AI model.

LaunchDarkly provides specific functions for completions for several common AI model families, and an option to record this information yourself.

Here's how to use a provider-specific `track[Model]Metrics` function to call a supported provider or framework, such as OpenAI, Amazon Bedrock, or LangChain, and record metrics from your AI model generation:

Node.js AI SDK (TypeScript), OpenAI model | Node.js AI SDK (TypeScript), Bedrock model | Python AI SDK, OpenAI model | Python AI SDK, Bedrock model | Python AI SDK, LangChain model | Python AI SDK, LangChain helper functions | Ruby AI SDK, OpenAI model | Ruby AI SDK, Bedrock model | Ruby AI SDK, Bedrock helper function

```
1   const { tracker } = aiConfig;
2
3   if (aiConfig.enabled) {
4     // Pass in the result of the OpenAI operation.
5     // When you call the OpenAI operation, use details from aiConfig.
6     // For instance, you can pass aiConfig.messages
7     // and aiConfig.model to your specific OpenAI operation.
8     //
9     // CAUTION: If the call inside of trackOpenAIMetrics throws an except
10    // the SDK will re-throw that exception
11
12    const completion = await tracker.trackOpenAIMetrics(async () =>
13      client.chat.completions.create({
```

```
      maxTokens: (aiConfig.model?.parameters?.maxTokens as number) ?? 4
    }),
  );

} else {

  // Application path to take when the aiConfig is disabled

}

// Call config() again each time you want to call the OpenAI operation

const aiConfig = aiClient.config(
  'ai-config-key-123abc',
  context,
  fallbackConfig,
  { 'example_variable': 'elephant' },
);

const { tracker } = aiConfig;
const completion = await tracker.trackOpenAIMetrics(...)
```

Here's how to use the general `TrackRequest` function to call any AI model provider and record metrics from your AI model generation:

.NET AI SDK, any model | Go AI SDK, any model

```
if (tracker.Config.Enabled == true) {

  var response = tracker.TrackRequest(Task.Run(() =>
    {
      // Make request to a provider, which automatically tracks metrics
      // When sending the request to a provider, use details from track
      // For instance, you can pass tracker.Config.Model and tracker.Co
      // Optionally, return response metadata, for example to do your o
      //
      // CAUTION: If the call inside of Task.Run() throws an exception,
      // the SDK will re-throw that exception.
```

```
15          Usage = new Usage { Total = 1, Input = 1, Output = 1 }, /* Toke
16          Metrics = new Metrics { LatencyMs = 100 } /* Metrics data */
17        };
18      }
19    ));
20
21  } else {
22
23    // Application path to take when the tracker.Config is disabled
24
25  }
26
27  // Call Config() again each time you want to call the generative AI ope
28  var tracker = aiClient.Config(
29    "ai-config-key-123abc",
30    context,
31    fallbackConfig,
32    new Dictionary<string, object> {
33      { "example_variable", "elephant" }
34    }
35  );
36
37  var response = tracker.TrackRequest(...)
```

Whether you use `track[Model]Metrics` or `TrackRequest`, the SDK automatically flushes these pending analytics events to LaunchDarkly at regular intervals. If you have a short-lived application, such as a script or unit test, you may need to explicitly request that the underlying LaunchDarkly client deliver any pending analytics events to LaunchDarkly, using `flush()` or `close()`.

Here's how:

.NET AI SDK | Go AI SDK | Node.js AI SDK | Python AI SDK | Ruby AI SDK

```
1 baseClient.Flush();
```

You can also track metrics yourself

If you are using another provider, or want to record additional metrics, each AI SDK also includes individual `track*` methods to record duration, token usage, generation success, generation error, time to first token, output satisfaction, and more. To learn more, read AI metrics.

LaunchDarkly AI SDK sample applications

For a complete example application, you can review some of our sample applications:

- Node.js AI SDK, using OpenAI
- Node.js AI SDK, using Bedrock
- Python AI SDK, using OpenAI
- Python AI SDK, using Bedrock
- Python AI SDK, using LangChain
- Ruby AI SDK, using Bedrock
- Ruby AI SDK, using OpenAI

Use Gemini prompt caching with AI Configs

To use Gemini's explicit prompt caching with AI Configs, we recommend structuring your variation with two system messages:

1. Use the first system message (`messages[0]`) for static content that should be cached.
2. Use the second system message (`messages[1]`) for dynamic content that changes per request, such as user input or context variables.

In your application, call `aiClient.config()` to evaluate the AI Config. Extract `messages[0]` and pass it to Gemini's `system_instruction` field or use it when creating a cached prompt using the Gemini SDK. Use `messages[1]` as your dynamic prompt content.

To detect when the cached prompt should be refreshed, access `tracker.config.version`. This value changes when LaunchDarkly serves a different

Step 5, in LaunchDarkly: Monitor the AI Config

Select the **Monitoring** tab for your AI Config. As end users use your application, LaunchDarkly monitors the performance of your AI Configs. Metrics are updated approximately every minute.

Learn more about AI Configs

The following sections provide answers to common questions about working with AI Configs.

Integration with AI providers

In the AI Configs product, LaunchDarkly does not handle the integration to the AI provider. The LaunchDarkly AI SDKs provide your application with model configuration details for providers and frameworks such as OpenAI, Amazon Bedrock, and LangChain, including customized messages and model parameters such as temperature and tokens. It is your application's responsibility to pass this information to the AI provider or framework.

The LaunchDarkly AI SDKs provide methods to help you track how your AI model generation is performing, and in some cases, these methods take the completion from common AI providers as a parameter. However, it is still your application's responsibility to call the AI provider. To learn more, read Tracking AI metrics.

Privacy and personally identifiable information (PII)

LaunchDarkly does not send any of the information you provide to any models, and does not use any of the information to fine tune any models.

You should follow your own organization's policies regarding if or when it may be acceptable to send end-user data either to LaunchDarkly or to an AI provider. To learn

Availability of new models

When you create a new AI Config variation, you can select a model from the provided list. LaunchDarkly updates this list regularly. To request a new model, click the **Give feedback** option and let us know what models you'd like to have included.

You can also add your own model at any time. To learn how, read Create AI model configurations.

Previous Create AI Configs Next