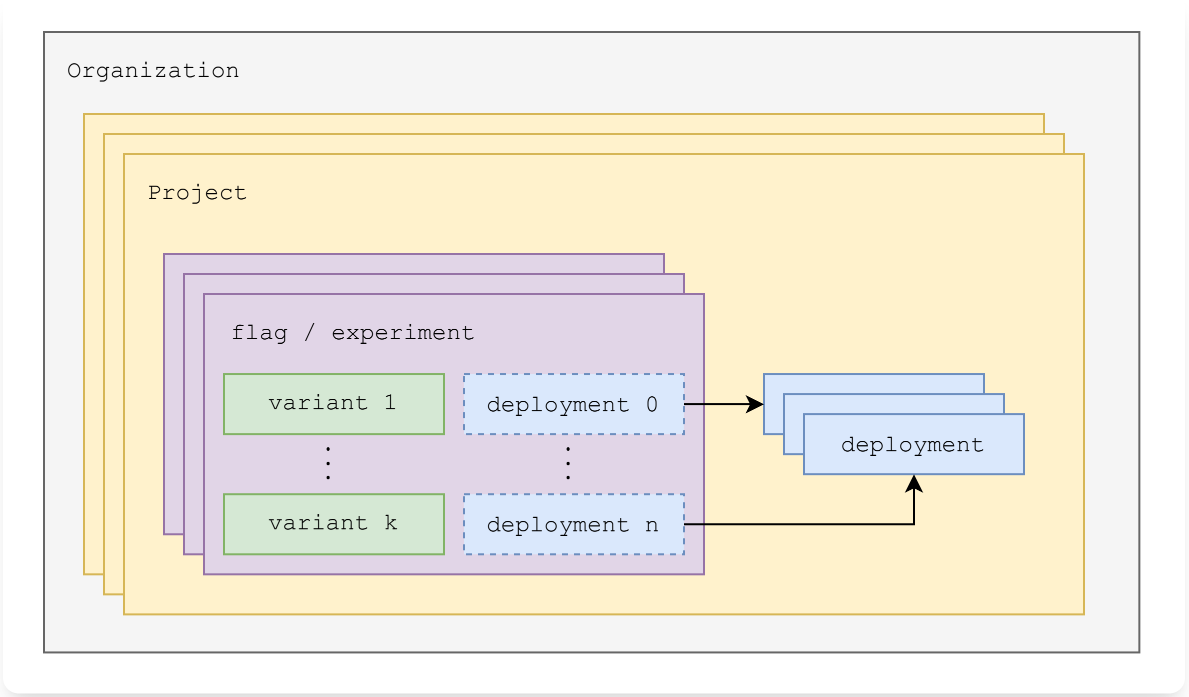
Data model

Copy page

At the top level in Amplitude is your **organization**. Within an organization, Amplitude Experiment follows the **project** structure defined by Amplitude Analytics. In short, all Experiment data must be associated with an Amplitude Analytics project.

Flags, experiments, and deployments all live within an Amplitude project.



Projects

Experiment uses the same projects which are required for Amplitude Analytics. As a best practice, create a project per product and per environment. Because flags, experiments, When developing a new feature with an experiment, you can create the experiment in the dev environment project to develop and test that the implementation is correct, then copy the experiment into the prod project to run the experiment in prod.

In Amplitude Experiment, a deployment serves a group of flags or experiments for use in an application. Each project has a deployment using the project API key as the deployment key, available by default. On creation, experiment deployments have an associated randomly generated **deployment key** which Experiment uses to identify the deployment and authorize requests to the evaluation servers.

Client vs. server deployments

Deployments are either client or server deployments. Use client-side deployments to initialize client-side SDKs, and server-side deployments to initialize server-side SDKs or authorize requests to the Evaluation API.

Deployments

Deployments belong to Amplitude Analytics projects, and a project can have multiple deployments. Amplitude recommends that you name deployments after the platform (client-side) or service (server-side) to which Experiment serves variants (for example: `android`, `ios`, `web`). The default project API key deployment is useful for getting started, but using explicit deployments for each platform or service is the best practice for larger organizations or teams that may share the same Amplitude project across multiple platforms for the same application.

Add deployments to Flags and Experiments in the same project. When Experiment's evaluation servers receive a request to fetch variants for a user, Experiment uses the deployment key to look up all associated flags and experiments for evaluation.

Flags and experiments

experiment can be locally evaluated and may limit the targeting capabilities for the flag if set to local.

Feature flags and experiments share the same underlying data model, and you can migrate from one to the other retroactively. The most visible difference comes in the product interface: experiments guide you through an experiment lifecycle and give you the ability to define success metrics and perform analysis. Flags are more bare-bones, and don't include special planning and analysis sections.

Flags

Used for standard feature flagging without user analysis. When created, comes with a default variant, `on`.

Flag use cases

- Rolling out a feature to a subset of users (for example, beta customers).
- Different experience for a behavioral cohort (for example, power users).

Experiments

Used for feature experimentation on users. When created, comes with two default variants, `control` and `treatment`.

Experiment use cases

- Run an A/B test for a new feature in your application.
- Experiment on multiple recommendation algorithms on your server.

Variants

`Value`	**Required**	A string which identifies the variant in the instrumentation. The value string is checked for equality when a variant is accessed from the SDK or Evaluation REST API. Format must be lowercase, kebab-case, or snake_case.
`Payload`	Optional	Dynamic JSON payload for sending arbitrary data down with the variant. For example, you could send down a hex code to change the color of a component in your application.
`Name`	Optional	Name for the variant. This is like `Value`, but doesn't have formatting limitations, and you can change it without breaking the instrumentation in your code base.
`Description`	Optional	A more detailed description of the variant. You can use this to describe what the user experiences when viewing the variable experience in more detail.

SDK use

Only the `Value` and `Payload` are available when accessing a variant from an SDK or the Evaluation REST API.

Users

Experiment users map to a user within Amplitude Analytics. Alongside flag configurations, users are an input to evaluation. Flag and experiment targeting rules can make use of user properties.

Pass users to evaluation via `fetch` requests for remote evaluation, or directly to the `evaluate` function for local evaluation.

Warning

You must include either a user ID or device ID in the user object for evaluation to succeed. For example, remote evaluation returns a 400 error if both the User ID and Device ID are null, empty, or missing.

		Amplitude ID on enrichment before remote evaluation where the Amplitude ID is the default bucketing key.
`device_id`	`string`	The Device ID is the secondary identifier for the user. This is usually randomly generated by an analytics SDK on the client side or set in a cookie on the server side. The Device ID is also used when resolving the Amplitude ID on enrichment before remote evaluation where the Amplitude ID is the default bucketing key.
`user_properties`	`object`	Optional object of custom properties used when evaluating the user during local or remote evaluation.
`groups`	`object`	Beta. Optional object that lists groups associated with this user. Format is an object where the key is the group type, and the value is an array of group value strings (for example, `{"org name":["Amplitude"]}`)
`group_properties`	`object`	Beta. Optional object listing group properties associated with this user. Format is an nested object where the key is the group type, and the value is an object where the key is a the group value, and the value is an object of properties (for example, `{"org name":{"Amplitude":{"plan":"enterprise"}}}`)

Beta

If your organization has purchased the Accounts add-on you may perform bucketing and analysis on groups rather than users. Reach out to your representative to gain access to this beta feature.

Groups must be included in the user sent with the fetch request (recommended), or identified with the user via a group identify call from the Group Identify API or with `setGroup()` from an analytics SDK.

All Experiment SDKs support groups, with minimum versions described in the following table:

SDK	Minimum version
Android	1.9.0
iOS	1.10.0
React Native	1.1.0

Go	1.7.0
Python	1.3.0
JVM	1.3.0
Node	1.5.0
PHP	1.0.0

Full user definition

Full user definition >

Was this page helpful? ☆ ☆ ☆ ☆ ☆
June 3rd, 2024