# Bundle splitting for plugins


- Enable static modules
- Group your routes
- Manually loading code on-demand
- Sharing code with other themes and plugins
- Route map syntax
- Compatibility
- Verification


By default, every javascript module in a plugin is loaded when Discourse boots. This guide describes the `staticModules` system, which allows code to be automatically loaded on-demand, thereby improving initial boot times.

This system is currently available to plugins. Theme support will come later

# Enable static modules

Add a `frontend` section to your plugin's `about.json`:

```
{
  "frontend": {
    "staticModules": true
  }
}
```

Two things happen:

Your components, helpers, modifiers and lib code stop being loaded automatically. Instead, they are only loaded when referenced by an import in other code.

Your routes, controllers & templates move into a separate bundle which loads on demand.

# Group your routes

With `staticModules` on, all the routes in your plugin will go into one bundle. That is usually what you want.

To split further, name a bundle on a route. The route and everything under it go in that bundle:

```
export default function () {
  this.route("chat", { bundleName: "chat" }, function () {
    this.route("channel", { path: "/c/:title/:id" });
  });

  this.route("chat-settings", { bundleName: "chat-settings" });
}
```

Here `chat` and `chat.channel` load together, and `chat-settings` loads separately.

If a map mounts on another route and all of its routes belong in the same bundle, name it once on the map instead:

```
export default {
  resource: "admin.adminPlugins.show",
  bundleName: "workflows-admin",

  map() {
    this.route("workflows", { path: "workflows" });
    this.route("workflows-variables", { path: "variables" });
  },
};
```

# Manually loading code on-demand

Bundling is automatic for routes only. Anything imported from an initializer or service is loaded eagerly.

When something is large and rarely used (e.g. a modal that opens on a click), you should import it dynamically instead.

```
export default async function showNewMessageModal(modal) {
  const { default: ChatModalNewMessage } =
    await import("../components/chat/modal/new-message");

  modal.show(ChatModalNewMessage);
}
```

The path must be a literal, so that the build can find it. To pull in a group of modules with one import, re-export them from a single module and import that.

When lazy-loading components, it normally makes sense to use the `DAsyncContent` wrapper from `ui-kit`. For example:

```
export default class LazyLoader extends Component {
  get complexComponent() {
    return import("./complex-component");
  }

  <template>
    <DAsyncContent @asyncData={{this.complexComponent}}>
      <:content as |componentModule|>
        <componentModule.default />
      </:content>
    </DAsyncContent>
  </template>
}
```

`DAsyncContent` registers its own test waiter. If your plugin uses qunit tests and you call `import()` yourself, wrap the promise in `waitForPromise` so that tests wait for it to load:

```
import { waitForPromise } from "@ember/test-waiters";

await waitForPromise(import("./my-module"));
```

# Sharing code with other themes and plugins

Enabling `staticModules` puts a boundary around your plugin. If another plugin or theme needs to import one of your modules, list it under `exports` in `about.json`:

```
{
  "frontend": {
    "staticModules": true,
    "exports": {
      "discourse/components/chat-channel": "discourse/components/chat-channel",
      "discourse/models/chat-channel": "discourse/models/chat-channel"
    }
  }
}
```

The key is the name other plugins import by, and the value is the module behind it. Keeping the two apart means you can move a module without breaking anyone, and you can export the same module under more than one name. Other plugins reach these as `discourse/plugins/<your-plugin>/<key>`.

Adding a module here will cause it to be eagerly loaded when the application boots, and therefore carries a performance cost. Consider providing an asynchronous-friendly entrypoint for any shared components/libs, so that most code is only loaded on the first use.

# Route map syntax

Route maps are statically analyzed during the build, so it's important for them to use simple syntax. Route maps must only include `this.route` calls, with literal string names and options.

This means no conditionals, no loops, and no values read from settings, services, or imports. If the build cannot parse your map, your plugin will fail to compile.

# Compatibility

When using staticModules, you should avoid using these legacy features:

- `templateName` / `controllerName` overrides in routes
- Injecting controllers or routes via `@controller` or `.lookup()`
- Loading components via `.lookup()`
- `requirejs.entries`, `require()`, `define()`
- Loading your plugin's modules from other plugins (except those listed in `exports`)

# Verification

Generated bundles are listed in `app/assets/generated/<your-plugin>/manifest.json` after a build. Discourse will preload the right bundle when a user lands directly on your page.

---

□□□ #2
□ □□□ □□□ 2026-10-31 13:28:48 UTC
□ □□□ □□□ 2026-10-31 13:28:49 UTC