Manifest overview

ON THIS PAGE

The manifest contains three required top-level properties: `app` , `modules` , and `permissions` , and number of optional properties. For example:

```
app:
  id: "ari:cloud:ecosystem::app/406d303d-0393-4ec4-ad7c-1435be94583a"
  licensing:
    enabled: true
```

Property	Required	Description
`app`	Yes	Identifying information, licensing details, and app storage (EAP). See App to learn more.
`permissions`	Yes	A list of the permissions required by the app. See Permissions to learn more.
`modules`	Yes*	A list of the modules used by the app. Cannot be empty. See Modules to learn more. *Not required when `connectModules` is present.

`connectModules`	This is a backwards compatibility feature to support incremental migration from Connect to Forge. Names and fields map 1:1 to Connect descriptor modules. `connectModules` will be populated automatically by `@atlassian/connect-to-forge` during descriptor conversion. Adding Connect Modules to a new Forge app is not supported.
`endpoint`	A list of remote endpoints referenced by remote resolver invocations. See Endpoint to learn more.
`providers`	Authentication providers used by the app. See Providers to learn more.
`remotes`	A list of remote services required by the app (along with additional options for declaring egress details for data residency). See Remotes to learn more.
`resources`	A list of the resources used by the app. See Resources to learn more.
`environment`	A list of *environment variables* to be parsed by the Forge CLI for entire or partial field values. After specifying a variable in `environment`, you can declare it in any field by enclosing it in `${` and `}`. See Environment to learn more.
`translations`	A list of translation resources supported by the app, including the fallback configurations for when a desired translation is not available. See Translations to learn more.

The Forge platform enforces a maximum file size limit of 200 KB for the `manifest.yml` file. If your manifest exceeds this size, deployment will fail with a validation error.

To avoid this, ensure that your manifest only includes the modules and configuration necessary for your app. If you encounter this limit, consider removing unused modules, splitting your app into smaller components, or refactoring your configuration. This limit helps maintain platform performance and reliability.

App

The `app` dictionary contains properties about your Forge app. Some of these are populated as part of the `forge create` command (for example, `id`).

`id`	Yes	Forge CLI supplies this identifier when you create or register an app for the first time.
`connect`		Details specific to Adopting Forge from Connect. This is required if the manifest has `connectModules`. See Connect to learn more.
`licensing`	No	The app's licensing state. To enable licensing for your app, add the `enabled` field attribute and set its value to `true`. See licensing to learn more.
`package`	No	Settings relating to packaging the Forge application. See Packaging to learn more.
`runtime`	Yes	Settings relating to the Forge runtime. See Runtime to learn more.
`storage`	No	A list of *custom entities* and their respective *indexes*. Custom entities are user-defined data structures for storing app data. Forge's storage API lets you query data stored in these structures using a wide array of query conditions. These query conditions make it possible to build advanced, complex queries to suit your app's operations. See Custom entities to learn more.

Runtime

The `runtime` property lets you configure the Forge runtime using the following settings:

Setting	Required	Type	Description
`name`	Yes	`string`	Lets you specify the Forge runtime environment version on which to deploy your app. This field supports the following values: • `nodejs24.x` : (recommended) specifies the Node.js 24 runtime version. This version runs on a standard Node.js 24 ↗ environment. • `nodejs22.x` : specifies the Node.js 22 runtime version. This version runs on a standard Node.js 22 ↗ environment. • `nodejs20.x` : specifies the Node.js 20 runtime version. This version runs on a standard Node.js 20 ↗ environment. See Runtime for more information about these runtime versions.
`architecture`	No	`string`	Specify the architecture used to deploy your app. Supported values are: `arm64` and `x86_64`. Default

			functions powered by `arm64` architecture are designed to deliver up to 19 percent better performance at 20 percent lower cost. Note that `arm64` architecture is not available on Atlassian Government Cloud.
`memoryMB`	No	`number`	The default amount of memory available to all the functions at runtime. Increasing the function memory also increases its CPU allocation. The default value is 512 MB. The value can be between 128 MB and 1,024 MB. You can override the memory available for individual functions by setting at the function module definition.

⚠ The legacy Javascript sandbox runtime is now deprecated. This means that Forge can only create new apps on the latest runtime version. In addition:

- From October 29, 2024: Forge apps that haven't been updated since January 1, 2023 will no longer function.
- From February 28, 2025: *All* Forge apps still running on the legacy runtime version will no longer function.

If your app is running on the Javascript sandbox runtime, we strongly advise that you upgrade to the latest runtime.

Packaging

The `package` property lets you configure how the application's source code is packaged during deployment.

Setting	Type	Description
`extraFiles`	`string[]`	Extra files to copy to the deployed application. These can include application data, configuration files or additional programs the application might want to read or launch. Each item in this list can point to a single file or a glob pattern ↗ . When the Forge function runs, the files matching the specified patterns are available in the application directory.
`bundler` (EAP)	`string`	⚠ This is an experimental Early Access Program (EAP) feature, offered to selected users for testing and feedback purposes. EAP features are unsupported, not usable in production environments, and subject to change without notice. To start testing this feature, register your app here ↗ . By default, Forge uses Webpack to bundle your application code together with its dependencies. To compile your app code with TypeScript instead, specify `typescript` as your `bundler` . When you do:

your application's dependencies, you can upgrade the TypeScript version separately from the Forge CLI.

- All `dependencies` of the application will be packaged together with the application code, including all their data files. This allows your Forge application to use, for example, modules written in WebAssembly, or ones including native binaries.
- Including all dependencies might result in a larger package size (compared to the default Webpack bundling). If the package size exceeds the limits, the deployment will fail.
- The app's `devDependencies` will not be packaged. In addition, `@forge/react` and `@forge/bridge` packages will never be packaged, as they are only used for UI Kit (and therefore not used by back-end Forge functions).

Reading packaged files

You can read packaged files using the fs module ↗ .

```
import { readFileSync } from 'node:fs';

const data = readFileSync('./file.txt', 'utf8');
```

Executing packaged binaries

If you intend to add extra executables to your application and call them from functions, make sure they are compatible with the Forge runtime environment. You might want to use statically linked executables if possible, or include the required libraries together with the executable.

You can execute packaged binaries using the child_process module ↗ .

```
import { execFileSync } from 'node:child_process';

const stdout = execFileSync('./binary', ['--args'], {
  stdio: 'pipe',
  encoding: 'utf8',
});
```

⚠ While you can read files and execute binaries packaged with your app, you must not persist customer data to disk or allow long-running child processes to retain it. If you aren't sure whether a process will cache the data, don't allow it to persist beyond a single function invocation.

Refer to Expanded developer responsibilities for more information.

Connect

Connect apps that have adopted Forge modules can include Connect modules and a Connect key.

`key`	Yes	`key` is environment specific. See how to manage environments when using Forge from your Connect app. Note: The production environment of the app must match the Atlassian Marketplace key.
`remote`		The key of the `remotes` entry that holds the Connect app baseUrl. This is required if the manifest has `connectModules` .

Example

```
remotes:
  - key: connect-app-server
    baseUrl: https://hello-world-app.example.com
app:
  connect:
    key: hello-world
    remote: connect-app-server
```