Inside Koop CLI

HAOLIANG YU

HUB TECH TALK

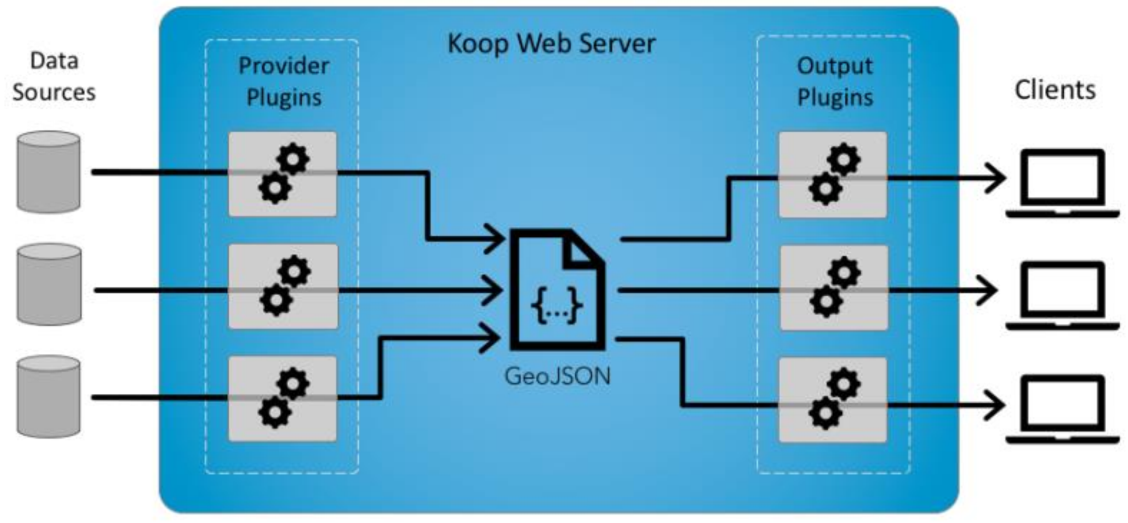
06/25/2019

Content

- What's Koop?
- Koop Plugin
- Koop CLI
- Experiments

What's Koop?

- A configurable Node.js web-server for on-the-fly transformation of geospatial data.
- Use plugins to read data from sources and transform it into a different format



Provider Core Output

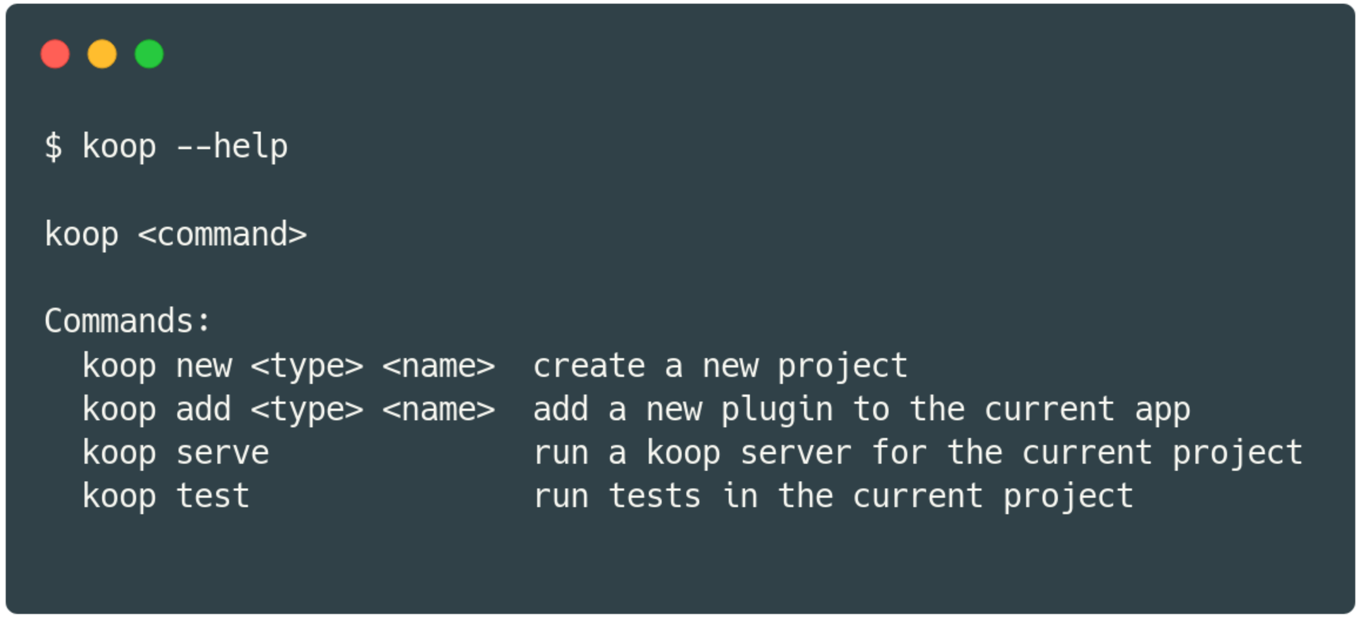
If we have provider A and output B, we can use Koop to do XYZ.

A good Koop plugin

- Follow specifications
- Simple to setup and develop
- Configurable and reusable
- Cross-platform
- Testable (with CI)
- Published to NPM

@koopjs/cli

- A Node.js CLI tool for developers who are developing Koop applications and plugins
- Main goals:
  - Automate trivial coding tasks
  - Promote good practices



```
$ koop --help

koop <command>

Commands:
  koop new <type> <name>  create a new project
  koop add <type> <name>  add a new plugin to the current app
  koop serve              run a koop server for the current project
  koop test               run tests in the current project
```

Before the CLI…

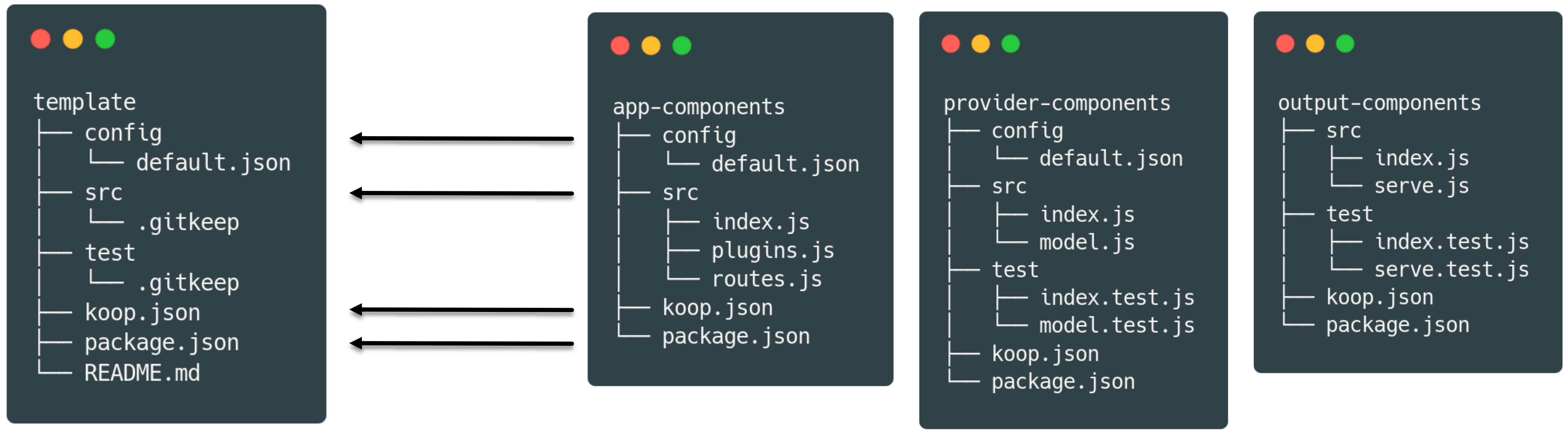
Creating a Koop plugin means

- Building it from scratch
  - Add everything for a package
  - Add my code
  - Add my tests
- From koop-provider-sample
  - Git clone
  - Remove unwanted files
  - Remove unwanted code
  - Add my code
  - Add my tests

$\longrightarrow$ Add my code and tests

What it does is copy-and-paste

$ koop new app my-app



```
template
├── config
│   └── default.json
├── src
│   └── .gitkeep
├── test
│   └── .gitkeep
├── koop.json
├── package.json
└── README.md
```

A new project template

```
app-components
├── config
│   └── default.json
├── src
│   ├── index.js
│   ├── plugins.js
│   └── routes.js
├── koop.json
└── package.json
```

App components

```
provider-components
├── config
│   └── default.json
├── src
│   ├── index.js
│   └── model.js
├── test
│   ├── index.test.js
│   └── model.test.js
├── koop.json
└── package.json
```

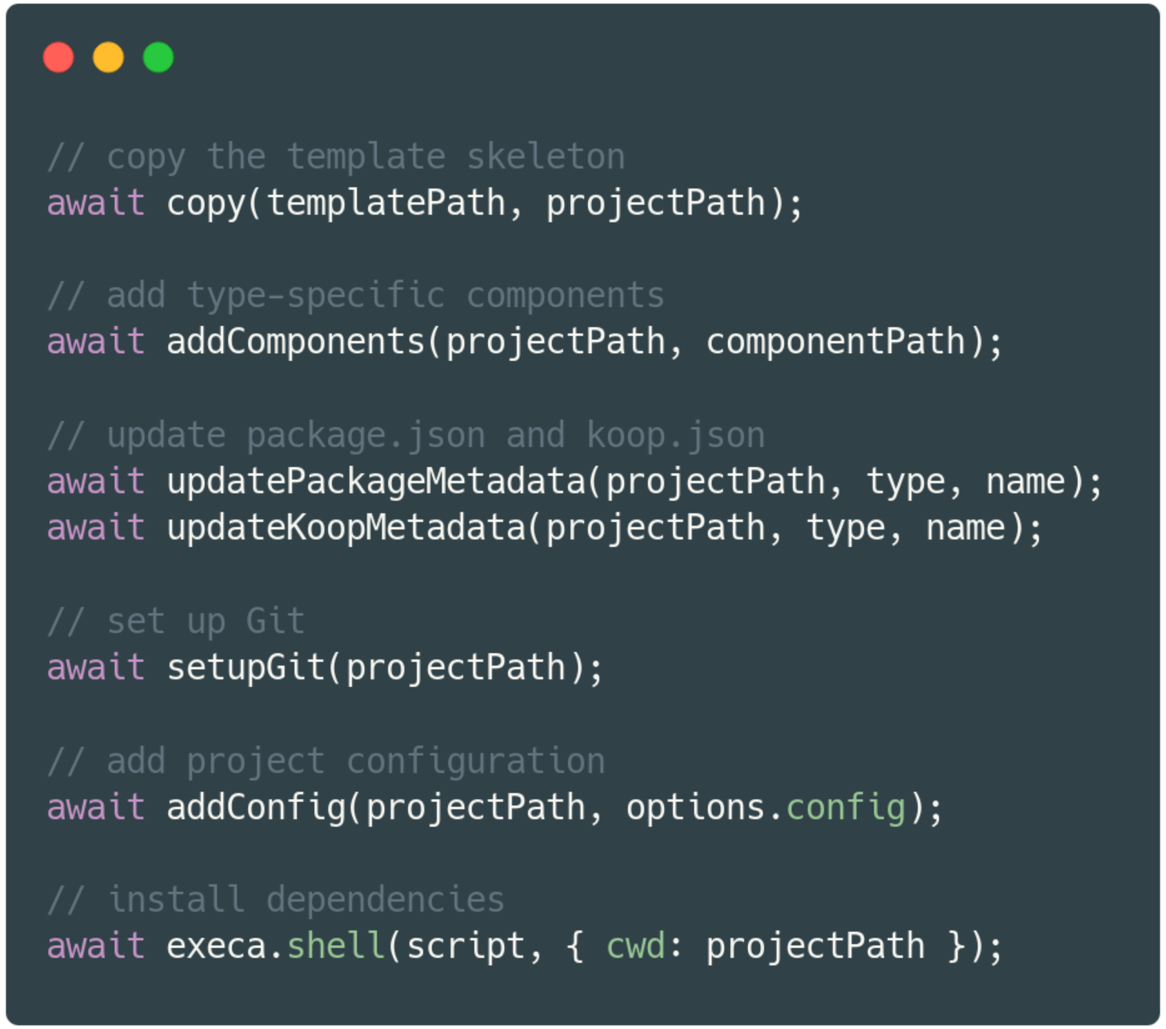
Provider components

```
output-components
├── src
│   ├── index.js
│   └── serve.js
├── test
│   ├── index.test.js
│   └── serve.test.js
├── koop.json
└── package.json
```

Output components

A project template with good practices

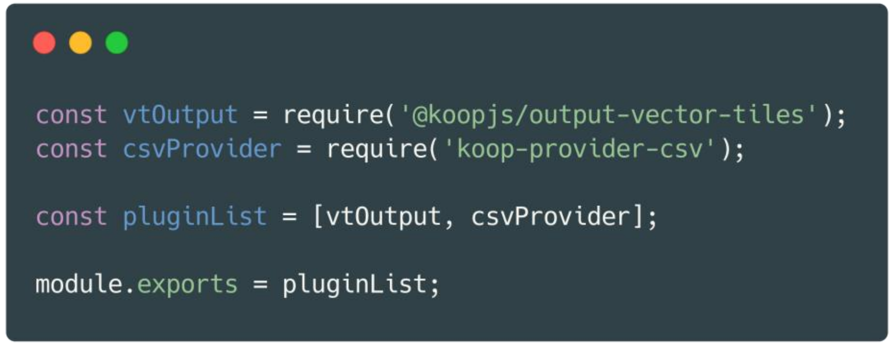
- Same for all types of app and plugins
  - Project structure
  - Configuration method
- Each source file is associated with a test file
  - Tested based on the specification
- Koop project metadata



```javascript
// copy the template skeleton
await copy(templatePath, projectPath);

// add type-specific components
await addComponents(projectPath, componentPath);

// update package.json and koop.json
await updatePackageMetadata(projectPath, type, name);
await updateKoopMetadata(projectPath, type, name);

// set up Git
await setupGit(projectPath);

// add project configuration
await addConfig(projectPath, options.config);

// install dependencies
await execa.shell(script, { cwd: projectPath });
```

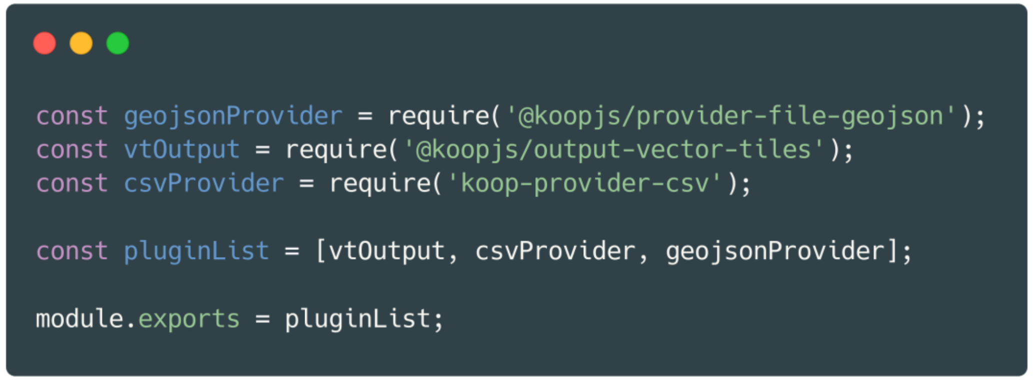
What it does is copy-and-paste-and-edit

$ koop new provider @koopjs/provider-file-geojson



```
const vtOutput = require('@koopjs/output-vector-tiles');
const csvProvider = require('koop-provider-csv');

const pluginList = [vtOutput, csvProvider];

module.exports = pluginList;
```

$\longrightarrow$

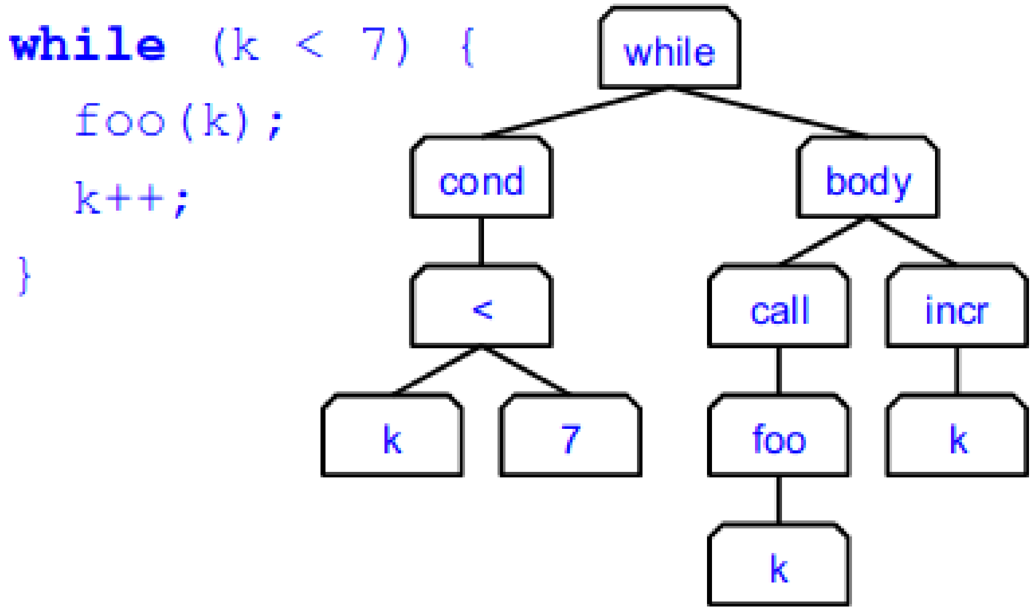


```
const geojsonProvider = require('@koopjs/provider-file-geojson');
const vtOutput = require('@koopjs/output-vector-tiles');
const csvProvider = require('koop-provider-csv');

const pluginList = [vtOutput, csvProvider, geojsonProvider];

module.exports = pluginList;
```

- Adding a plugin in an existing app's plugin list:
  - Load the provider
  - Appending to the plugin list

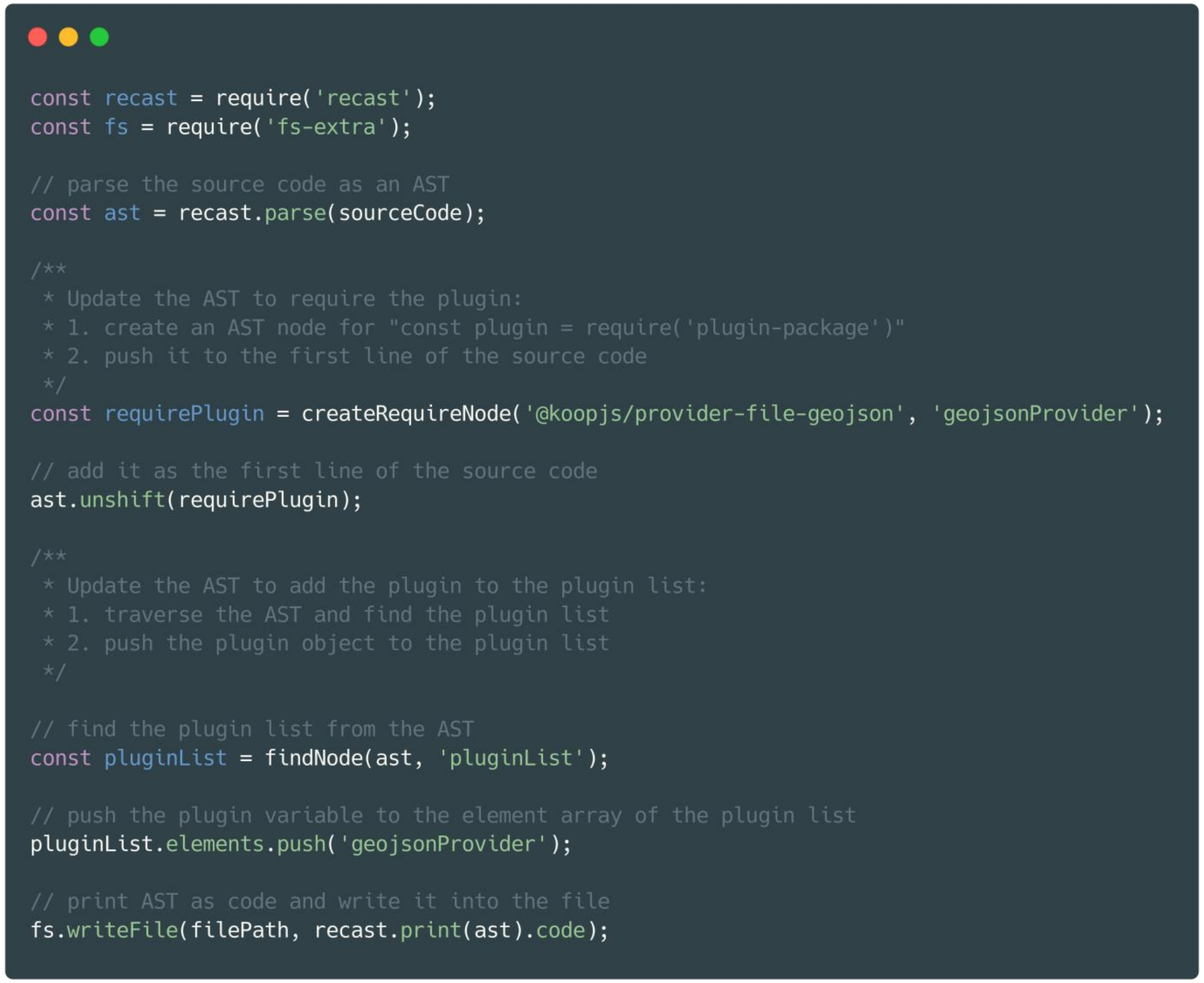
Editing Source Code

- String manipulation is not an option
- Use Abstract Syntax Tree (AST)
  - Parse the source code as an AST
  - Traverse the tree and look for the target node
  - Update node values
  - Print the AST as source code



```
while (k < 7) {
   foo(k);
   k++;
}
```

AST Example (Source)



```
const recast = require('recast');
const fs = require('fs-extra');

// parse the source code as an AST
const ast = recast.parse(sourceCode);

/**
 * Update the AST to require the plugin:
 * 1. create an AST node for "const plugin = require('plugin-package')"
 * 2. push it to the first line of the source code
 */
const requirePlugin = createRequireNode('@koopjs/provider-file-geojson', 'geojsonProvider');

// add it as the first line of the source code
ast.unshift(requirePlugin);

/**
 * Update the AST to add the plugin to the plugin list:
 * 1. traverse the AST and find the plugin list
 * 2. push the plugin object to the plugin list
 */

// find the plugin list from the AST
const pluginList = findNode(ast, 'pluginList');

// push the plugin variable to the element array of the plugin list
pluginList.elements.push('geojsonProvider');

// print AST as code and write it into the file
fs.writeFile(filePath, recast.print(ast).code);
```

What it does is copy-and-paste-and-edit **ON WINDOWS**

- Cross-platform is not an option, but a necessity.
  - Many developers are using Linux/MacOS/Windows.
- Pay attention to:
  - Dependencies (must be cross-platform)
  - Newline (use os.EOL and adapt to user input)
  - Temp folder (use os.tempdir())
  - File path (use path)
  - ENV (use cross-env)
- Tested with Windows (thanks Travis CI)

Testing is testing

```
$ koop test
```

It just runs the "npm test" command for the project.

Nothing special ¯_(ツ)_/¯

Running a dev server

```
$ koop serve –-port 3000
```

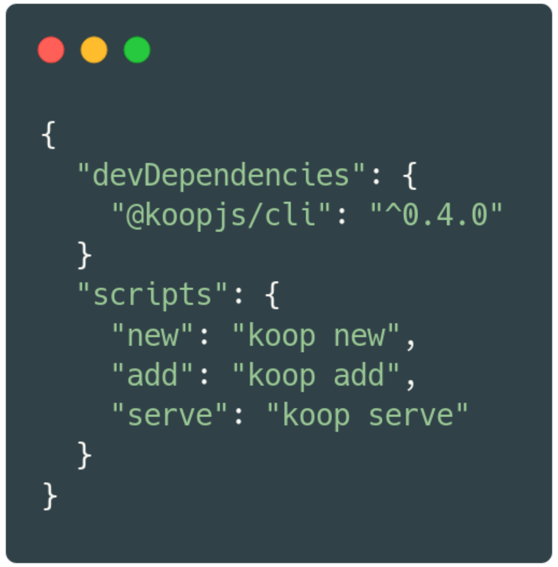
Each type of project has different needs for the dev server:

- App: just run the index.js
- Provider: needs additional output
- Output: needs additional provider

The command eases the pain by providing a default dev server for every project type.

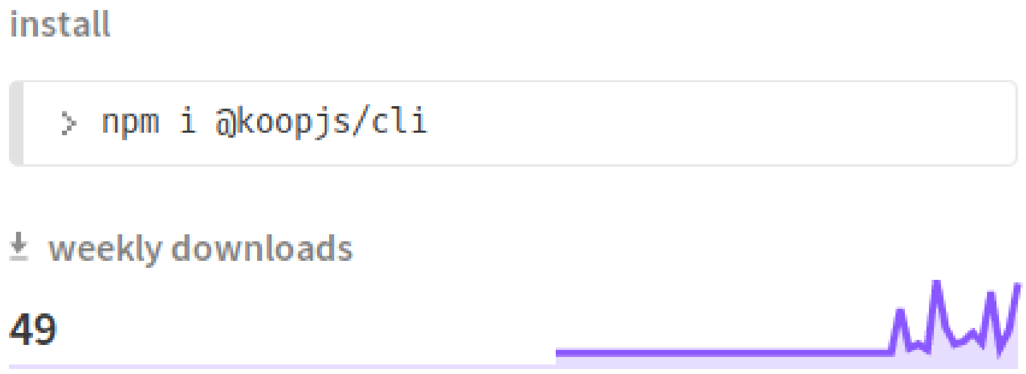
Run as a local dependency

Not everyone is happy with installing a global dependency.



```
{
  "devDependencies": {
    "@koopjs/cli": "^0.4.0"
  }
  "scripts": {
    "new": "koop new",
    "add": "koop add",
    "serve": "koop serve"
  }
}
```

Package.json



install

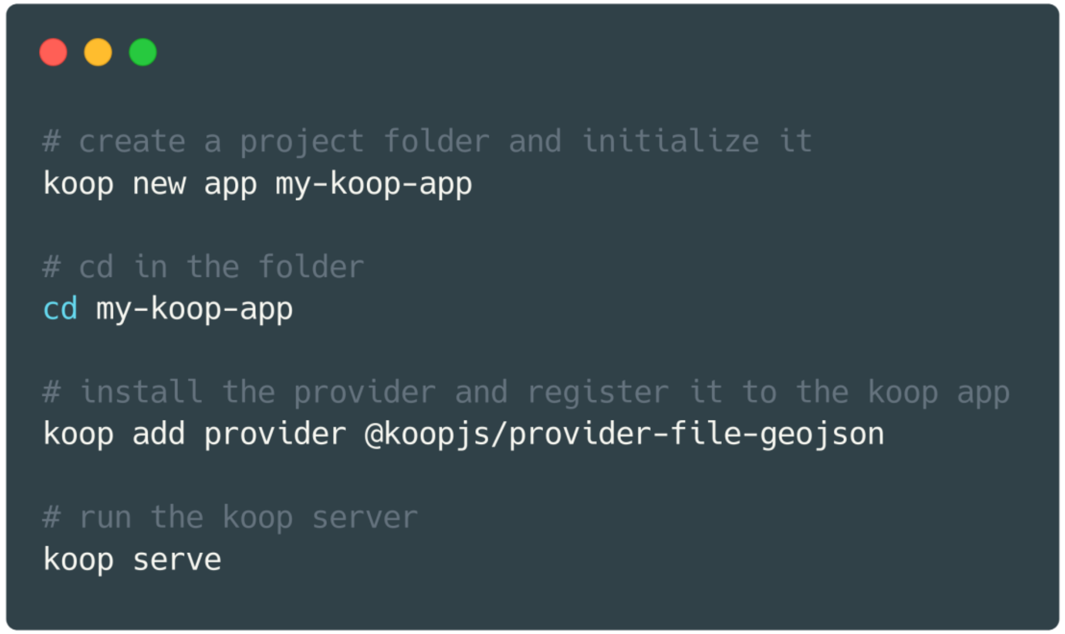
```
> npm i @koopjs/cli
```

weekly downloads

49

A good way to track community activity

Build a Koop app with CLI



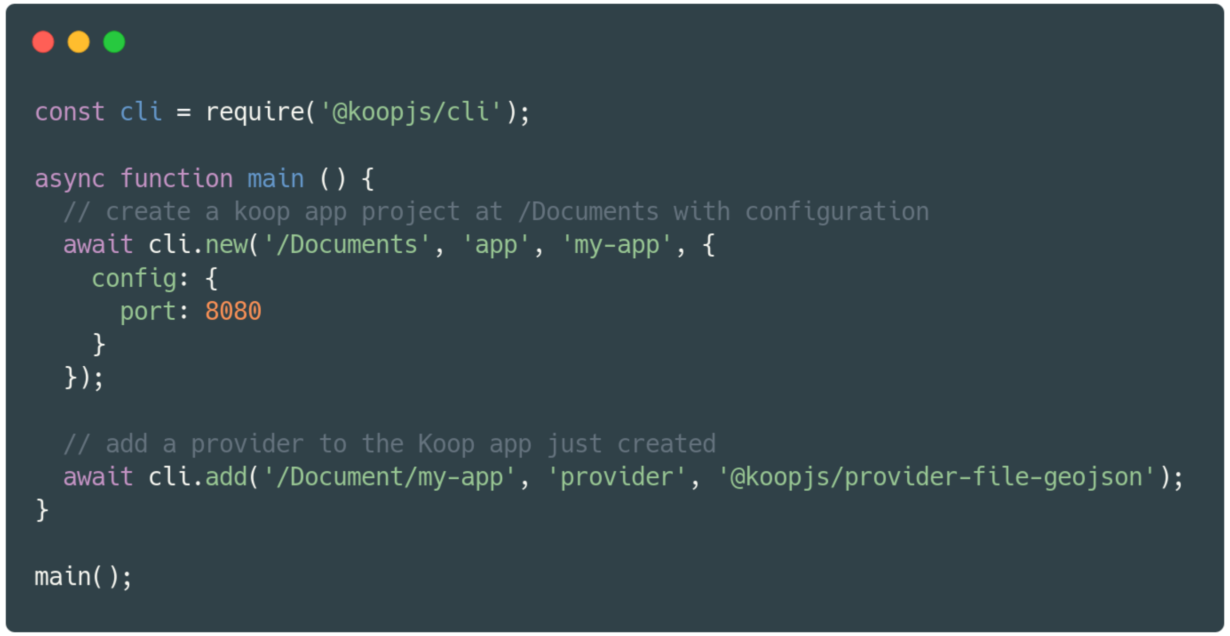
```
# create a project folder and initialize it
koop new app my-koop-app

# cd in the folder
cd my-koop-app

# install the provider and register it to the koop app
koop add provider @koopjs/provider-file-geojson

# run the koop server
koop serve
```

…and it is a cool one

- Follow specifications
  - Project template guarantees it
- Simple to setup and develop
  - Automation is in place
- Configurable and reusable
  - It depends, but the configuration file is added
- Cross-platform
  - Code from the CLI is tested at multiple platforms
- Testable
  - Tests are added automatically

Beyond CLI

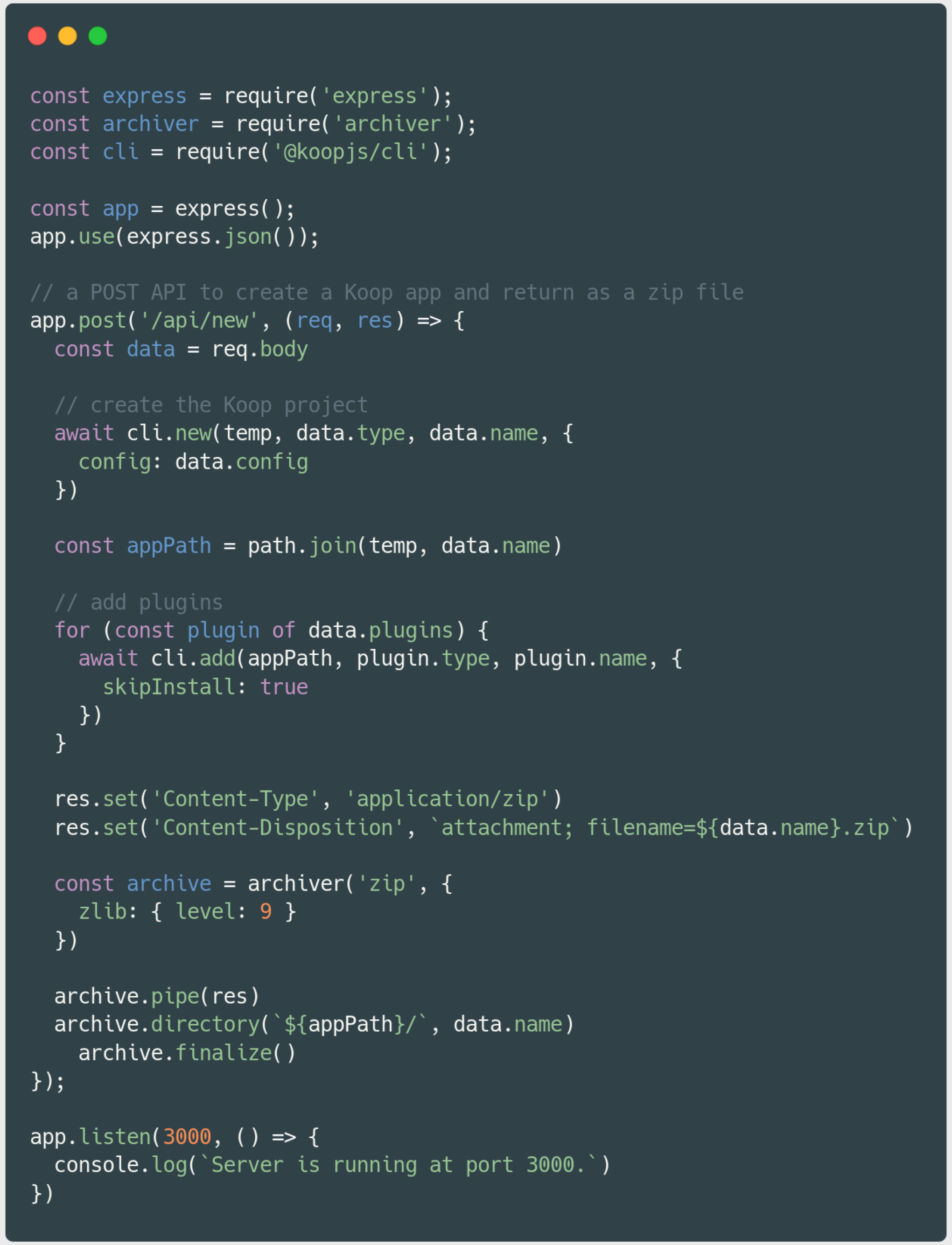
The "new" and "add" commands are exposed as Node.js functions:



```
const cli = require('@koopjs/cli');

async function main () {
  // create a koop app project at /Documents with configuration
  await cli.new('/Documents', 'app', 'my-app', {
    config: {
      port: 8080
    }
  });

  // add a provider to the Koop app just created
  await cli.add('/Document/my-app', 'provider', '@koopjs/provider-file-geojson');
}

main();
```

Get a Koop project from an API

- Wrap everything in an API
- Try https://create-koop-app.herokuapp.com/api/new/app/my-app
- Visit [create-koop-app repo]()



```javascript
const express = require('express');
const archiver = require('archiver');
const cli = require('@koopjs/cli');

const app = express();
app.use(express.json());

// a POST API to create a Koop app and return as a zip file
app.post('/api/new', (req, res) => {
  const data = req.body

  // create the Koop project
  await cli.new(temp, data.type, data.name, {
    config: data.config
  })

  const appPath = path.join(temp, data.name)

  // add plugins
  for (const plugin of data.plugins) {
    await cli.add(appPath, plugin.type, plugin.name, {
      skipInstall: true
    })
  }

  res.set('Content-Type', 'application/zip')
  res.set('Content-Disposition', `attachment; filename=${data.name}.zip`)

  const archive = archiver('zip', {
    zlib: { level: 9 }
  })

  archive.pipe(res)
  archive.directory(`${appPath}/`, data.name)
    archive.finalize()
});

app.listen(3000, () => {
  console.log(`Server is running at port 3000.`)
})
```

Let's Koop it.

Some important packages

- yargs, CLI framework
- recast, JavaScript AST parser and printer
- fs-extra, more powerful and easier file manipulation
- execa, cross-platform process executor
- klaw-sync, walk through directories
- cross-env, cross-platform ENV
- mocha/chai, testing