Node Parcel Merchant Pro Teams Pricing Documentation Community

npm Sign Up Sign In

Search packages Search

Join us to discuss the challenges, solutions and best practices for in-house JavaScript code sharing. Tuesday, 12/17 at 10am PT/1pm ET. Sign up here »

cfenv

1.2.2 • Public • Published 9 months ago

Readme

Explore BETA

3 Dependencies

136 Dependents

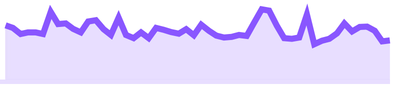
10 Versions

Install

```
> npm i cfenv
```

Weekly Downloads

177,718



Version

1.2.2

License

Apache-2.0

Unpacked Size

78.8 kB

Total Files

20

Issues

10

Pull Requests

1

Homepage

github.com/cloudfoundry-community/node-cfenv

Repository

github.com/cloudfoundry-community/node-cfenv

Last publish

9 months ago

Collaborators

>_ Try on RunKit

Report a vulnerability

cfenv - easy access to your Cloud Foundry application environment

The cfenv package provides functions to parse Cloud Foundry-provided environment variables. Provides easy access to your port, http binding host name/ip address, URL of the application, etc. Also provides useful default values when you're running locally.

The package determines if you are running "locally" versus running as a Cloud Foundry app, based on whether the `VCAP_APPLICATION` environment variable is set. If not set, the functions run in "local" mode, otherwise they run in "cloud" mode.

quick start

```
var cfenv = require("cfenv")

var appEnv = cfenv.getAppEnv()
...
// start the server on the given port and binding host, and print
// url to server when it starts
```

```
server.listen(appEnv.port, appEnv.bind, function() {
    console.log("server starting on " + appEnv.url)
})
```

This code snippet above will get the port, binding host, and full URL to your HTTP server and use them to bind the server and print the server URL when starting.

running in Cloud Foundry vs locally

This package makes use of a number of environment variables that are set when your application is running in Cloud Foundry. These include:

- `VCAP_SERVICES`
- `VCAP_APPLICATION`
- `PORT`

If these aren't set, the `getAppEnv()` API will still return useful values, as appropriate. This means you can use this package in your program and it will provide useful values when you're running in Cloud Foundry AND when you're running locally. You can supply local versions of `VCAP_SERVICES` and/or `VCAP_APPLICATION` by using the `vcap` and `vcapFile` options described below.

api

The `cfenv` package exports the following function:

`getAppEnv(options)`

Get the core bits of Cloud Foundry data as an object.

The `options` parameter is optional, and can contain the following properties:

- `name` - name of the application

  This value is used as the default `name` property of the returned object, and as the name passed to the ports package `getPort()` function to get a default port.

  If not specified, the name will looked for in the following places:

  - the `name` property of the `VCAP_APPLICATION` environment variable
  - the `name` property from the `manifest.yml` file in the current directory
  - the `name` property from the `package.json` file in the current directory
- `protocol` - protocol used in the generated URLs

  This value is to override the default protocol used when generating the URLs in the returned object. It should be the same format as node's url `protocol` property. That is, it should end with a `:` character.

- `vcap` - provide values for the `VCAP_APPLICATION` and `VCAP_SERVICES` environment variable, when running locally. The object can have properties `application` and/or `services` , whose values are the same as the values serialized in the respective environment variables.

  Note that the `url` and `urls` properties of the returned object are not based on the vcap `application` object, when running locally.

  This option property is ignored if not running locally.

- `vcapFile` - provide the same function as the `vcap` option, but instead of setting the option value to an object, you set it to the name of a JSON file, which will be parsed and used as the `vcap` option value is used, as described above.

  When both `vcap` and `vcapFile` options are provided, the values in `vcap` are ignored.

  This option property is ignored if not running locally.

return value

The `getAppEnv()` function returns an object with the following properties:

- `app` : object version of `VCAP_APPLICATION` env var
- `services` : object version of `VCAP_SERVICES` env var
- `name` : name of the application
- `port` : HTTP port
- `bind` : hostname/ip address for binding
- `urls` : URLs used to access the servers
- `url` : first URL in `urls`
- `isLocal` : false if a valid `VCAP_APPLICATION` env var was found, true otherwise

The returned object also has the following methods available:

- `appEnv.getServices()`
- `appEnv.getService(spec)`
- `appEnv.getServiceURL(spec, replacements)`
- `appEnv.getServiceCreds(spec)`

If no value can be determined for `port` , and the `name` property on the `options` parameter is not set and cannot be determined, a port of 3000 will be used.

If no value can be determined for `port` , and the `name` property on the `options` parameter is set or can otherwise be determined, that name will be passed to the ports package `getPort()` function to get a default port.

The protocol used for the URLs will be `http:` if the app is running locally, and `https:` otherwise; you can force a particular protocol by using the `protocol` property on the `options` parameter.

When running in Cloud Foundry, the `url` and `urls` values will have `localhost` as their hostname, if the actual hostnames cannot be determined.

AppEnv methods

The following methods are also available on the object returned by `cfenv.getAppEnv()` .

`appEnv.getServices()`

Return all services, in an object keyed by service name.

Note that this is different than the `services` property returned from `getAppEnv()` .

For example, assume VCAP_SERVICES was set to the following:

```
{
    "user-provided": [
        {
            "name": "cf-env-test",
            "label": "user-provided",
            "tags": [],
            "credentials": {
                "database": "database",
                "password": "passw0rd",
                "url": "https://example.com/",
                "username": "userid"
            },
            "syslog_drain_url": "http://example.com/syslog"
        }
    ]
}
```

In this case, `appEnv.services` would be set to that same object, but `appEnv.getServices()` would return

```
{
    "cf-env-test": {
        "name": "cf-env-test",
        "label": "user-provided",
        "tags": [],
        "credentials": {
            "database": "database",
            "password": "passw0rd",
            "url": "https://example.com/",
            "username": "userid"
        },
        "syslog_drain_url": "http://example.com/syslog"
    }
}
```

`appEnv.getService(spec)`

Return a service object by name.

The `spec` parameter should be a regular expression, or a string which is the exact name of the service. For a regular expression, the first service name which matches the regular expression will be returned.

Returns the service object from VCAP_SERVICES or null if not found.

`appEnv.getServiceURL(spec, replacements)`

Returns a service URL by name.

The `spec` parameter should be a regular expression, or a string which is the exact name of the service. For a regular expression, the first service name which matches the regular expression will be returned.

The `replacements` parameter is an object with the properties used in **node's url function `url.format()`** .

Returns a URL generated from VCAP_SERVICES or null if not found.

To generate the URL, processing first starts with a `url` property in the service credentials. You can override the `url` property in the service credentials (if no such property exists), with a `replacements` property of `url` , and a value which is the name of the property in the service credentials whose value contains the base URL.

That url is parsed with **node's url function `url.parse()`** to get a set of initial url properties. These properties are then overridden by entries in `replacements` , using the following operation, for a given replacement `key` and `value` .

```
url[key] = service.credentials[value]
```

The URL `auth` replacement is a bit special, in that it's value should be a two-element array of [userid, password], where those values are keys in the service.credentials

For example, assume VCAP_SERVICES was set to the following:

```
{
    "user-provided": [
        {
            "name": "cf-env-test",
            "label": "user-provided",
            "tags": [],
            "credentials": {
                "database": "database",
                "password": "passw0rd",
                "url": "https://example.com/",
                "username": "userid"
            },
            "syslog_drain_url": "http://example.com/syslog"
        }
    ]
}
```

Assume you run the following code:

```
url = appEnv.getServiceURL("cf-env-test", {
    pathname: "database",
    auth:     ["username", "password"]
})
```

The `url` result will be `https://userid:passw0rd@example.com/database`

Note that there **MUST** be a `url` property in the credentials, or replacement for it, in the service, or the call will return `null`. Also, because the `url` is parsed first with `url.parse()`, there will be a `host` property in the result, so you won't be able to use the `hostname` and `port` values directly. You can **ONLY** set the resultant hostname and port with the `host` property.

Note that `url.parse()` and the later `url.format()` calls to construct the result, will not produce pleasing results for "unusual" URLs, especially those which do not use `http:` or `https:` protocols. The `url` `parse()` and `format()` methods will not be used though, if you have no replacement values, or the only replacement property is `url`, and so are safe to use in that case.

Since the `appEnv.getServiceURL()` method operates against the `appEnv.services` property, you can fudge this object if that makes your life easier.

`appEnv.getServiceCreds(spec)`

Returns the `credentials` object of a service by name.

The `spec` parameter is the same as that used by the `appEnv.getServiceURL()` method. If there is no service that matches the `spec` parameter, this method will return `null`.

If there is a service that matches the `spec` parameter, the value of it's `credentials` property will be returned. If for some reason, there is no `credentials` property on the service, an empty object - `{}` - will be returned.

testing with Cloud Foundry

You can push this project as a Cloud Foundry project to try it out.

First, create a service name `cf-env-test` with the following command:

```
cf cups cf-env-test -p "url, username, password, database"
```

You will be prompted for these values; enter something reasonable like:

```
url>      http://example.com
username> userid
password> passw0rd
database> the-db
```

Next, push the app with `cf push`.

When you visit the site, you'll see the output of various cfenv calls.

changes

1.2.2 - 2019/03/26

- handle ports package race condition by returning port 3000 - pr #41

1.2.1 - 2019/03/25

- upgrade js-yaml to avoid vulnerability - pr #39

1.2.0 - 2019/02/21

- use vcapFile port value if available - pr #36
- upgrade underscore from 1.8.x to 1.9.x - pr #37
- use `random-route:` instead of `${random-word}` in sample manifest - pr #38

1.1.0 - 2018/04/18

- add the `vcapFile` option - issue #31

1.0.4 - 2017/01/13

- fix to getServiceURL() with non-http URLs - issue #21

1.0.3 - 2014/10/02

- fixes for compatibility with Diego - issue #11

1.0.2 - 2014/09/29

- delete a lingering `npm-debug.log` left behind
- add `npm-debug.log to .gitignore`

1.0.1 - 2014/09/29

- remove node_modules from .cfignore - issue #8
- updated package dependencies
- changed README.md to correct sample service to cf-env-test
- files in lib/ recompiled due to coffee-script update

1.0.0 - 2014/09/03

- initial 1.0.0 release

contributing

See the **CONTRIBUTING.md** doc for more information on contributing to this project.

license

Apache License, Version 2.0

http://www.apache.org/licenses/LICENSE-2.0.html

Keywords

none

Help

Documentation

Community

Resources

Advisories

Status

Contact

About

Company

Blog

Careers

Webinars