
heroku-buildpack-static

NOTE: This buildpack is in an experimental OSS project.

This is a buildpack for handling static sites and single page web apps.

For a guide, read the [Getting Started with Single Page Apps on Heroku](#).

WARNING: heroku-buildpack-static is deprecated

This buildpack is deprecated and is no longer being maintained. If you are using this project, you can transition over to NGINX via an NGINX buildpack. Use your project's existing configuration. To find the NGINX configuration generated by the heroku-buildpack-static you can run:

```
1 $ heroku run bash
2 ~ $ bin/config/make-config
3 ~ $ cat config/nginx.conf
```

These commands will output your current NGINX config generated from your `static.json` contents.

- Write these contents to your local repo at `config/nginx.conf.erb`, commit them to git.
- Replace path logic that previously used `mruby` with static logic.
- Configure your app to use the NGINX buildpack via `heroku buildpacks:add heroku-community/nginx`.
- Remove this buildpack via `heroku buildpacks:remove heroku-community/static` (or `heroku buildpacks:remove https://github.com/heroku/heroku-buildpack-static`).

Deprecation PRs

If you have tips or tricks for migrating off of this buildpack and want to add them to the instructions above please send a PR.

Features

- serving static assets
- gzip on by default
- error/access logs support in `heroku logs`
- custom configuration

Deploying

The `static.json` file is required to use this buildpack. This file handles all the configuration described below.

1. Set the app to this buildpack: `$ heroku buildpacks:set heroku-community/static.`
2. Deploy: `$ git push heroku master`

Configuration

You can configure different options for your static application by writing a `static.json` in the root folder of your application.

Root This allows you to specify a different asset root for the directory of your application. For instance, if you're using ember-cli, it naturally builds a `dist/` directory, so you might want to use that instead.

```
1 {  
2   "root": "dist/"  
3 }
```

By default this is set to `public_html/`

Canonical Host This allows you to perform 301 redirects to a specific hostname, which can be useful for redirecting `www` to `non-www` (or vice versa).

```
1 {  
2   "canonical_host": "www.example.com"  
3 }
```

You can use environment variables as well:

```
1 {  
2   "canonical_host": "${HOST}"  
3 }
```

Default Character Set This allows you to specify a character set for your text assets (HTML, Javascript, CSS, and so on). For most apps, this should be the default value of "UTF-8", but you can override it by setting `encoding`:

```
1 {
2   "encoding": "US-ASCII"
3 }
```

Clean URLs For SEO purposes, you can drop the `.html` extension from URLs for say a blog site. This means users could go to `/foo` instead of `/foo.html`.

```
1 {
2   "clean_urls": true
3 }
```

By default this is set to `false`.

Logging You can disable the access log and change the severity level for the error log.

```
1 {
2   "logging": {
3     "access": false,
4     "error": "warn"
5   }
6 }
```

By default `access` is set to `true` and `error` is set to `error`.

The environment variable `STATIC_DEBUG` can be set, to override the `error` log level to `error`.

Custom Routes You can define custom routes that combine to a single file. This allows you to pre-serve routing for a single page web application. The following operators are supported:

- `*` supports a single path segment in the URL. In the configuration below, `/baz.html` would match but `/bar/baz.html` would not.
- `**` supports any length in the URL. In the configuration below, both `/route/foo` would work and `/route/foo/bar/baz`.

```
1 {
2   "routes": {
3     "/*.html": "index.html",
4     "/route/**": "bar/baz.html"
5   }
6 }
```

Browser history and asset files When serving a single page app, it's useful to support wildcard URLs that serves the index.html file, while also continuing to serve JS and CSS files correctly. Route ordering allows you to do both:

```
1 {
2   "routes": {
3     "/assets/*": "/assets/",
4     "/*": "index.html"
5   }
6 }
```

Custom Redirects With custom redirects, you can move pages to new routes but still preserve the old routes for SEO purposes. By default, we return a 301 status code, but you can specify the status code you want.

```
1 {
2   "redirects": {
3     "/old/gone/": {
4       "url": "/",
5       "status": 302
6     }
7   }
8 }
```

Interpolating Env Var Values It's common to want to be able to test the frontend against various backends. The `url` key supports environment variable substitution using `${ENV_VAR_NAME}`. For instance, if there was a staging and production Heroku app for your API, you could setup the config above like the following:

```
1 {
2   "redirects": {
3     "/old/gone/": {
4       "url": "${NEW_SITE_DOMAIN}/new/here/"
5     }
6   }
7 }
```

Then using the config vars, you can point the frontend app to the appropriate backend. To match the original proxy setup:

```
1 $ heroku config:set NEW_SITE_DOMAIN="https://example.herokuapp.com"
```

Custom Error Pages You can replace the default nginx 404 and 500 error pages by defining the path to one in your config.

```
1 {
2   "error_page": "errors/error.html"
3 }
```

HTTPS Only You can redirect all HTTP requests to HTTPS.

```
1 {
2   "https_only": true
3 }
```

Basic Authentication You can enable Basic Authentication so all requests require authentication.

```
1 {
2   "basic_auth": true
3 }
```

This will generate `.htpasswd` using environment variables `BASIC_AUTH_USERNAME` and `BASIC_AUTH_PASSWORD` if they are present. Otherwise it will use a standard `.htpasswd` file present in the `app` directory.

Passwords set via `BASIC_AUTH_PASSWORD` can be generated using OpenSSL or Apache Utils. For instance: `openssl passwd -apr1`.

Proxy Backends For single page web applications like Ember, it's common to back the application with another app that's hosted on Heroku. The down side of separating out these two applications is that now you have to deal with CORS. To get around this (but at the cost of some latency) you can have the static buildpack proxy apps to your backend at a mountpoint. For instance, we can have all the api requests live at `/api/` which actually are just requests to our API server.

```
1 {
2   "proxies": {
3     "/api/": {
4       "origin": "https://hone-ember-todo-rails.herokuapp.com/"
5     }
6   }
7 }
```

Interpolating Env Var Values It's common to want to be able to test the frontend against various backends. The `origin` key supports environment variable substitution using `${ENV_VAR_NAME}`.

For instance, if there was a staging and production Heroku app for your API, you could setup the config above like the following:

```
1 {
2   "proxies": {
3     "/api/": {
4       "origin": "https://${API_APP_NAME}.herokuapp.com/"
5     }
6   }
7 }
```

Then using the config vars, you can point the frontend app to the appropriate backend. To match the original proxy setup:

```
1 $ heroku config:set API_APP_NAME="hone-ember-todo-rails"
```

Custom Headers Using the headers key, you can set custom response headers. It uses the same operators for pathing as Custom Routes.

```
1 {
2   "headers": {
3     "/": {
4       "Cache-Control": "no-store, no-cache"
5     },
6     "/assets/**": {
7       "Cache-Control": "public, max-age=512000"
8     },
9     "/assets/webfonts/*": {
10      "Access-Control-Allow-Origin": "*"
11    }
12  }
13 }
```

For example, to enable CORS for all resources, you just need to enable it for all routes like this:

```
1 {
2   "headers": {
3     "/*": {
4       "Access-Control-Allow-Origin": "*"
5     }
6   }
7 }
```

Precedence When there are header conflicts, the last header definition always wins. The headers do not get appended. For example,

```
1 {
```

```
2   "headers": {
3     "/*": {
4       "X-Foo": "bar",
5       "X-Bar": "baz"
6     },
7     "/foo": {
8       "X-Foo": "foo"
9     }
10  }
11 }
```

when accessing `/foo`, `X-Foo` will have the value `"foo"` and `X-Bar` will not be present.

Route Ordering

- HTTPS redirect
- Root Files
- Clean URLs
- Proxies
- Redirects
- Custom Routes
- 404

Procfile / multiple buildpacks

In case you have multiple buildpacks for the application you can ensure static rendering in [Procfile](#) with `web: bin/boot`.

Testing

For testing we use Docker to replicate Heroku locally. You'll need to have it setup locally. We're also using `rspec` for testing with Ruby. You'll need to have those setup and install those deps:

```
1 $ bundle install
```

To run the test suite just execute:

```
1 $ bundle exec rspec
```

Structure

To add a new test, add another example inside `spec/simple_spec.rb` or create a new file based off of `spec/simple_spec.rb`. All the example apps live in `spec/fixtures`.

When writing a test, `BuildpackBuilder` creates the docker container we need that represents the heroku cedar-14 stack. `AppRunner.new` takes the name of a fixture and mounts it in the container built by `BuildpackBuilder` to run tests against. The `AppRunner` instance provides convenience methods like `get` that just wrap `net/http` for analyzing the response.

Boot2docker

If you are running docker with boot2docker, the buildpack will automatically send tests to the right ip address. You need to forward the docker's port 3000 to the virtual machine's port though.

```
1 VBoxManage modifyvm "boot2docker-vm" --natpf1 "tcp-port3000,tcp
  ,3000,,3000";
```

Releasing new binaries

The steps buildpack maintainers need to perform when releasing new nginx binaries (either for a new stack or `ngx_mruby` version), are:

1. Update the stacks list in `Makefile` and/or the `ngx_mruby` version in `scripts/build_ngx_mruby.sh`.
2. Run `make build` to build all stacks or `make build-heroku-NN` to build just one stack.
3. Ensure the AWS CLI is installed (eg `brew install awscli`).
4. Authenticate with the relevant AWS account (typically by setting the environment variables from PCSK).
5. Run `make sync` (or if using a custom S3 bucket, `S3_BUCKET=... make sync`).
6. Update `bin/compile` to reference the new stacks and/or nginx version URLs.
7. Open a PR with the changes from (1) and (6).