Environment Variables

Contents

Luminus aims to facilitate developing 12 factor style applications. The 12 factor approach states that the configuration should be kept separate from the code. The application should not have to be packaged differently for each environment that it's deployed in.

The environment variables are managed by the cprop library. The configuration is represented by a map. The map is constructed by aggregating variables from multiple sources in the environment. The sources include EDN configuration, shell variables, and Java system properties.

EDN Based Configuration

The library will first look for a `config.edn` file on the resource path. This will be used as the base configuration for the application. An external configuration file can be specified using the `conf` Java option at runtime: `-Dconf=prod-config.edn`.

The configuration placed in `config.edn` should consist of a map such as the following:

```
{:port 4000}
```

This configuration will be merged on top of the configuration found on the resource path. The library uses a deep merge strategy, so any nested structures will be merged intelligently.

Build Tool: lein ▾

Topics

Your First Application
REPL Driven Developme
Application Profiles
HTML Templating
Static Assets
ClojureScript
Routing
RESTful Services
Request types
Response types
Websockets
Middleware
Sessions and Cookies
Input Validation
Security
Component Lifecycle
Database Access
Database Migrations
Logging
Internationalization
Testing
Server Tuning
- **Environment Variables**
Deployment
Useful Libraries
Sample Applications
Upgrading
Clojure Resources

Books

Managing Environment Variables

The `System/getProperties` will be merged on top of the configuration found on the resource path and the optional EDN configuration file.

```
java -Ddatabase-url="jdbc:postgresql://localhost/app?user=app_user&pas
```

The variable names are converted into Clojure style keywords. The variables are lowercased and _ characters are used to indicate nesting, while `.` characters are converted to `-` characters.

`-Dport=3000` **->** `{:port 3000}`

`-Dnrepl-port=7000` **->** `{:nrepl-port 7000}`

`-Ddatabase-url="jdbc:h2:./guestbook_dev.db"` **->** `{:database-url "jdbc:h2:./guestbook_dev.db"}`

`-Dio_http.max.connections=10` **->** `{:io {:http-max-connections 10}}`

Any environment variables found in `System/getenv` will be merged last. These variables are parsed using the following strategy:

`PORT=3000` **->** `{:port 3000}`

`NREPL_PORT=7000` **->** `{:nrepl-port 7000}`

`DATABASE_URL="jdbc:h2:./guestbook_dev.db"` **->** `{:database-url "jdbc:h2:./guestbook_dev.db"}`

`IO__HTTP_MAX_CONNECTIONS="{:value 10}"` **->** `{:io {:http-max-connections {:value 10}}}`

Note that the _ is converted to `-`, while __ is used to indicate nesting for shell variables. These conventions can be mixed as seen with `IO__HTTP_MAX_CONNECTIONS`.

See the official documentation for further details.

Default Environment Variables

Luminus projects use the following environment variables by default:

`PORT` - HTTP port that the application will attempt to bind to, defaults to `3000`

`NREPL_PORT` - when set the application will run the nREPL server on the specified port, defaults to `7000` for development

`DATABASE_URL` - the URL for the database connection

`APP_CONTEXT` - used to specify an optional context for the routes in the application

The Config Namespace

The variables are populated in the `env` map that's found in the `<app>.config` namespace that looks as follows:

```
(ns <app>.config
  (:require [cprop.core :refer [load-config]]
            [cprop.source :as source]
            [mount.core :refer [args defstate]]))

(defstate env :start (load-config
                      :merge
                      [(args)
                       (source/from-system-props)
                       (source/from-env)]))
```

The configuration will load the environment variables from the known sources and merge it with the command line arguments populated by clojure.tools.cli in the `<app>.core/start-app` function. The resulting configuration is a map that can be accessed as seen in the example below:

```
(ns <app>.db.core
  (:require [<app>.config :refer [env]]))

(def database-url
  (-> env :database :url))
```

Environment Specific Code

Some code, such as development middleware for showing stacktraces in the browser, is dependent on the mode the application runs in. For example, we'd only want to run the above middleware during development and not show stacktraces to the client in production.

Luminus uses `env/dev/clj` and `env/prod/clj` source paths for this purpose. By default the source path will contain the `<app>.env` namespace that has the environment specific configuration. The `dev` config looks as follows:

```
(ns <project-ns>.env
  (:require [selmer.parser :as parser]
            [clojure.tools.logging :as log]
            [<project-ns>.dev-middleware :refer [wrap-dev]]))

(def defaults
  {:init
   (fn []
     (parser/cache-off!)
     (log/info "\n-=[app started successfully using the development pr
   :middleware wrap-dev}
```

The config references the `<app>.dev-middleware` namespace found in the same source path. Any development specific middleware should be placed there.

Meanwhile, the `prod` config will not

```
(ns <project-ns>.env
  (:require [clojure.tools.logging :as log]))

(def defaults
  {:init
   (fn []
     (log/info "\n-=[app started successfully]=-"))
   :middleware identity})
```

Only the middleware defined in the `<app>.middleware` namespace is run during production.
