**<https://www.architect.io/blog/2022-08-16/react-environment-variables-developers-guide/>**

**React environment variables: A developer’s guide**

A hands-on guide for managing environment variables in your React app, including best practices and common mistakes.

[Lee Brandt](https://www.architect.io/blog/author/lee-brandt/) • Aug 16 •

When working on the front-end portion of an application, you may need to interact with data from a back-end server at several locations within the codebase. This is often necessary for functions like updating a user’s shopping cart or providing access to premium features.

Usually, access to this data can only be approved using lengthy alphanumeric API credentials. Entering the same string each time you need to access the server is redundant and time-consuming. Plus, it risks exposing the security-critical key to malicious third parties.

One solution to this monotonous and potentially compromising practice is using environmental variables (EVs).

In this article, I’ll explain EVs, how you can use them in your applications, and catalog some best practices that front-end developers should implement when handling them.

**What are environment variables?**

In software development, an environment refers to the settings and conditions in which a piece of code is executed. It generally corresponds to a phase in the software development lifecycle — like development, testing, or production.

Engineering teams usually adopt different tools, platforms, or software configurations for each of these environments. For example:

* The database used in the development/testing stage will be different from that used in production to prevent interference with production data.
* The hostname for each domain will also vary (for example, production.example.com or dev.example.com) to maintain the separation of concerns.

EVs let you store globally scoped values to the environment your code is running in, making them available throughout the codebase. They enable you to:

* Decouple configurations from your code and limit the need to modify and re-deploy an application when configuration data changes.
* Set different configurations for different environments. For example, you could enable debugging logging and disable caching when in development — then enable caching and disable debug logging in production.
* Enable your application to be deployed in any environment without code changes

**How environment variables work in React**

In React, EVs are written as key and value pairs to be defined in the shell before the process that runs the server/application has started.

For example:

REACT\_APP\_YOUR\_VARIABLE\_NAME\_HERE = somevalue

React enforces EVs to be prefixed with the word **REACT\_APP\_** to enable the React engine to identify them as custom EVs. Any variable without the prefix is ignored during bundling to prevent you from accidentally [exposing a private key on a machine that could have the same name](https://github.com/facebook/create-react-app/issues/865#issuecomment-252199527).

React then loads the variables into **process.env** — a global object injected by Node.js at runtime that represents the state of the environment in which your app is running. Since environment variables have global scope in this environment, you can access them through it.

For example, you can access an environment variable named **REACT\_APP\_MY\_API\_KEY** in your code as **process.env.REACT\_APP\_MY\_API\_KEY**.

Note the standard naming convention for EVs can consist solely of uppercase letters, digits, and underscores (\_) but can’t begin with a digit.

**How to set up environment variables**

Depending on which bundler (for example, Webpack or Parcel) or framework (like React, Gatsby, or Nextjs) you use to manage your front end, the process for setting up environment variables may differ slightly.

Here are some approaches to set up environment variables in your React application.

**Using a single .env file**

The most common approach to define EVs in a React application is to store them in a plain text file with a .env extension located at the root of your project — that is, within the same directory as the package.json file.

For example:

//within a .env file

REACT\_APP\_YOUR\_API\_KEY = abxyz

If you use [Create React App](https://create-react-app.dev/) (CRA) to set up your React application, it uses the [**dotenv**](https://www.npmjs.com/package/dotenv) library to read environment variables found in any .env file and loads them into the `process.env` object. This ensures that each variable can be accessed using `process.env` as read-only values within the codebase.

Node.js also offers a built-in environment variable called **NODE\_ENV** that represents our application’s environment. In React, its value changes based on the script that’s running. Running **npm start** changes the environment value to “development.” Running **npm test**changes it to “test,” and running **npm run build**changes it to “production.”

This variable is special as you can use it to access different environment configurations. For example, suppose you have different databases for your production and development builds (to prevent interference). In that case, you can use **NODE\_ENV** to access certain variables based on the environment. Here’s an example:

function fetchDatabaseResponse () {

if (process.env.NODE\_ENV == 'production') {

return fetch(`${process.env.REACT\_APP\_PRODUCTION\_DATABASE\_URL}/users`)

} else {

return fetch(`${process.env.REACT\_APP\_DEVELOPMENT\_DATABASE\_URL}/users`)

}

}

**Using multiple .env files**

As your application grows, you may adopt different databases, servers, and third-party APIs for different environments to prevent interference. Rather than storing them all in a single .env file and using the **NODE\_ENV** environment variable to include them in your code, you can create different .env files for each of these environments. The main .env file usually contains all common/shared environment variables while other .env files with different suffixes (for example, .env.development, .env.production, .env.staging) contain variables for other environments.

For example, you could have the credentials to the development database defined in a .env.development file:

REACT\_APP\_API\_KEY=your-development-api-key

REACT\_APP\_DATABASE\_URL=https://your-development-database.com

REACT\_APP\_PROJECT\_ID=your-development-project-id

You can store the credentials in the production database in a .env.production file:

REACT\_APP\_API\_KEY=your-production-api-key

REACT\_APP\_DATABASE\_URL=https://your-production-database.com

REACT\_APP\_PROJECT\_ID=your-production-project-id

At runtime, depending on the current environment your app is running in, **dotenv** loads the correct environment variables from the corresponding .env file and replaces the reference to each environment variable name within the codebase with its current value.

Additionally, you might want to use environment variables for environments other than development, test, and production — like staging or debugging. You need the [**env-cmd**](https://www.npmjs.com/package/env-cmd) library to achieve this.

In the command line, within the root directory of your React application, run the command below:

npm install env-cmd

This installs the **env-cmd** library to help in using/executing a selected .env file.

Now, modify the script section in the **package.json** file to use the **env-cmd** command for the staging environment.

"scripts": {

"start:staging": "env-cmd -f .env.staging react-scripts start",

...

}

Then, running the command: **npm run start:staging** forces React to use the environment variables from the .env.staging file.

**Declaring temporary environment variables in the shell**

With CRA, you can declare environment variables within the shell on your machine. However, the specific method of declaring them varies among operating systems and command lines.

For the command prompt (cmd.exe) on a Windows machine, run the command:

set "REACT\_APP\_NOT\_SECRET\_CODE=abcdef" && npm start

For Microsoft PowerShell, use this command:

($env:REACT\_APP\_NOT\_SECRET\_CODE = "abcdef") -and (npm start)

Finally, for a Linux or macOS machine, use this command:

REACT\_APP\_NOT\_SECRET\_CODE=abcdef npm start

Notice that in each of the commands, **npm start** is called alongside the EV declaration. This is because environment variables defined in the shell are temporary and only last as long as that shell session lasts.

**Best practices for using environment variables**

To use environmental variables on the front end effectively, it’s important to remember a few best practices.

**Map environment variables to readable names**

You reference environment variables using the syntax **process.env.VARIABLE\_NAME**. If you have a lot of environment variables that will be accessed in multiple places across the codebase, repeatedly writing out that syntax is tedious at best. At worst, it leads to mistakes.

Therefore, try to use descriptive, short, and readable names to define them.

**Avoid adding strings or space characters**

By default, the values assigned to environment variables are enclosed in quotes when compiled at build time.

Consider this variable snippet:

REACT\_APP\_API\_KEY=AHEHEHR

When the code compiles, this will result in a string, “AHEHEHR.”

So, assigning a string to the **REACT\_APP\_API\_KEY** variable will result in the value appearing in double-quotes: “”AHEHEHR.”” This is a poor practice.

**How Architect offers a better solution**

One significant rule when using environment variables is to never commit your .env files with sensitive information (like your API key) to Git or upload to a public location like GitHub as this information can be abused or misused.

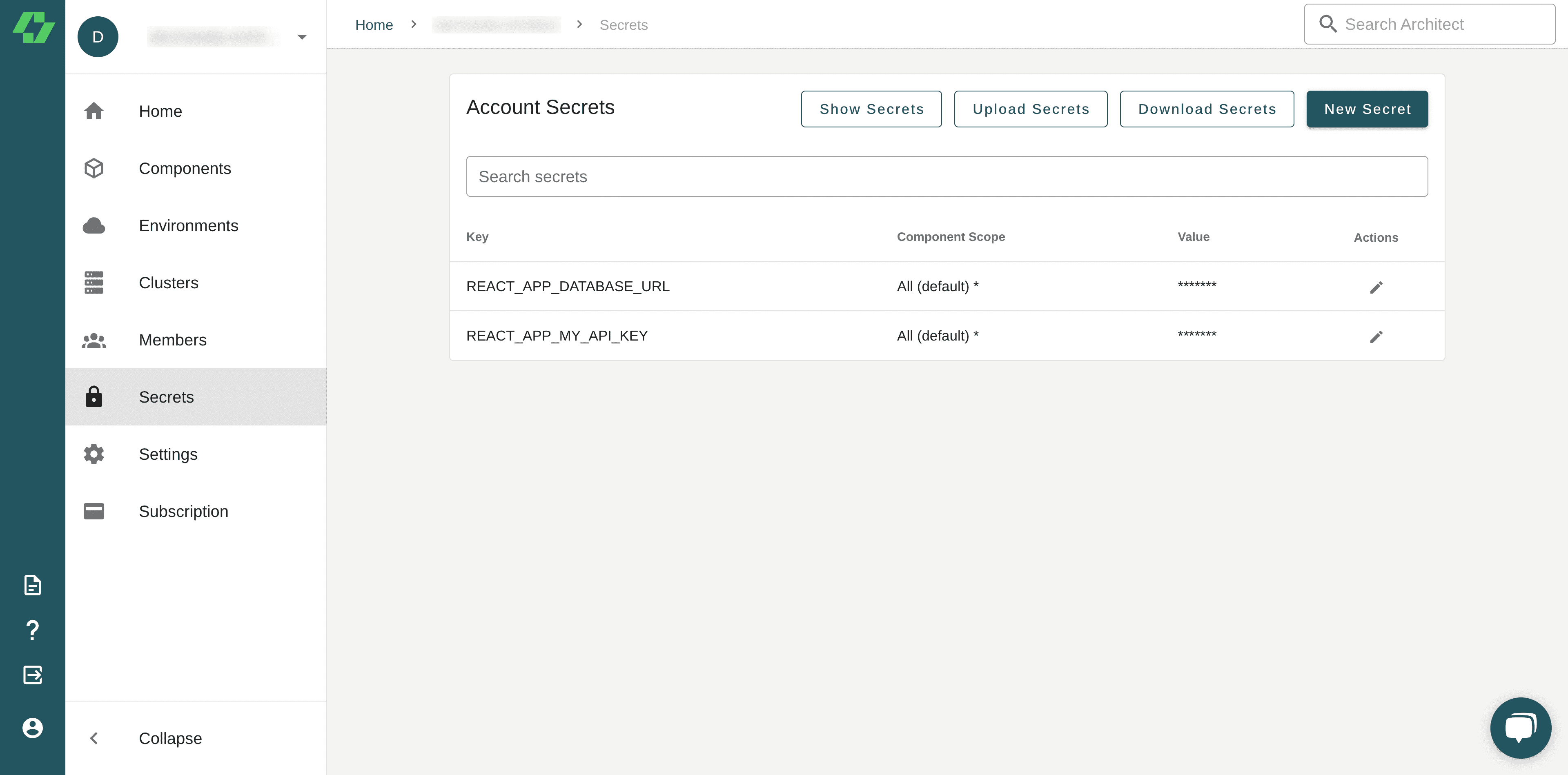
AWS keys are a typical example of this. Hackers create bots for the sole purpose of scanning GitHub repositories for AWS keys. Once hackers discover an AWS key, hackers can use that key to access resources and host resource-demanding applications, like Bitcoin miners, at your expense.

If you’ve previously committed such files, you must remove them from the commit history. Luckily, GitHub has a guide on [how to purge a file from the commit history](https://help.github.com/articles/removing-sensitive-data-from-a-repository/)].

So, how do you use these environment variables in your applications if the .env files aren’t included in your code? The answer is to use a platform that offers secret management — like [Architect](http://architect.io).

Architect is an all-in-one CI/CD platform for your cloud-native applications that lets you define and manage [secrets](https://docs.architect.io/components/secrets) like environment variables for different environments in your applications.

You can deploy to the Architect platform using a [command line](https://docs.architect.io/deployments/secrets#from-the-command-line), [config file](https://docs.architect.io/deployments/secrets#using-a-config-file), or your Architect dashboard. This is ideal for production-grade credentials — such as API keys — that you don’t want to save in your repository. Architect eliminates the need for .env files.



**Environment variables make your apps easier to configure**

Environment variables separate infrequently changing data from code, thus making apps easier to configure. To ensure you’re using them to their fullest, it’s essential to be aware of best security practices.

For example, avoid storing or managing sensitive information through .env files because they are stored in plain text and vulnerable to exposure to unauthorized users. Consider using a secret management service like [Architect](http://architet.io).

Check out our [getting started](https://docs.architect.io/getting-started/) guide to learn more about Architect and how it works, and our documentation on [secret management](https://docs.architect.io/deployments/secrets) to learn how you can integrate secrets into your applications. Try it out for yourself!