European Journal of Advances in Engineering and Technology, 2023, 10(9):27-32



Research Article

ISSN: 2394 - 658X

Environment Variable Management in React with Azure Static Web App

Bhargav Bachina

ABSTRACT

In the evolving landscape of web development, the deployment and construction of frontend applications have seen significant advancements, particularly with the advent of serverless architectures and containerized solutions. Among these innovations, Azure Static Web Apps service, a recent offering from Microsoft Azure, stands out for its streamlined approach to hosting and managing static web applications, currently available in preview mode. This technical paper delves into the practical aspects of utilizing Azure Static Web Apps service for deploying React applications, with a special focus on the separation of configuration from code. This separation enhances maintainability and allows for dynamic environment-specific adjustments without the need for multiple codebases builds. We outline the essential prerequisites for employing this service, alongside a step-by-step guide on setting up an example project.

Key words: JavaScript, React, Web Development, Cloud Computing, Azure

INTRODUCTION

In the current era, the methodologies for constructing and deploying frontend applications have diversified significantly, incorporating innovative technologies like serverless frameworks and containerization. Among these modern deployment strategies, the Azure Static Web Apps service emerges as a notable solution. This service, a recent addition to the Microsoft Azure suite, is specifically tailored for static web applications and is currently available in preview mode, highlighting its cutting-edge nature.

The practice of decoupling configuration details from the application code has become a standard approach in software development. This strategy enhances the application's flexibility and maintainability by enabling runtime configuration and eliminating the necessity for separate code builds for different deployment environments. Such an approach proves particularly beneficial in dynamic and scalable deployment scenarios.

In this detailed exploration, we will delve into the application of environment variables within the context of the Azure Static Web Apps service, particularly focusing on React-based applications. The discussion will encompass the advantages of this separation in terms of ease of maintenance and operational efficiency. By understanding how to effectively leverage environment variables during the development and deployment phases of Azure Static Web Apps, developers can significantly streamline their workflows and adapt more readily to varying environmental requirements.

- Prerequisites
- Example Project
- How To Configure In Local Development
- How To Configure in Other Environments
- Summary
- Conclusion

PREREQUISITES

You need to know a lot of things as prerequisites if you want to understand the Azure Static web apps service. First, you need to create two accounts: a Github account to store the source code and Microsoft Account to deploy that code using Azure static web app service. Let's create these accounts by following the below links. You can start both for free.

- Github Account (https://github.com/)
- Microsoft Azure Account (https://azure.microsoft.com/en-us/)

Since we are building a React application you need to be familiar with React CLI, nodejs, and typescript. First, you need to install NodeJS, create-react-app, and then you can create an application with React CLI.

- NodeJS (https://nodejs.org/en/download/)
- create-react-app (https://create-react-app.dev/)
- Typescript (https://www.typescriptlang.org/)
- React Bootstrap (https://react-bootstrap.github.io/)
- VSCode (https://code.visualstudio.com/)

All the API code that you develop for the application will be converted to Azure functions at the time of deployment. You need to be familiar with the following. When you make API calls from your app you need to proxy those calls to API you need a Live Server extension for that.

- Azure Functions (https://azure.microsoft.com/en-us/services/functions/)
- Azure Functions extension for Visual Studio Code (https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions)
- Live Server extension for Visual Studio Code (https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer)

EXAMPLE PROJECT

Here is an example of a simple todo application that creates, retrieves, edits, and deletes tasks. In a normal way, we actually run the API on the NodeJS server, and you can use MongoDB to save all these tasks.

https://miro.medium.com/v2/resize:fit:720/0*T8idJ6q3UWWFD2cU.gif

When it comes to Azure Static web apps you have to run the API with Azure Functions. We need to go through a series of steps to set it up. Let's dive into those steps in the following sections.

In the above example, we want to configure the *header color* and *text color* as environment variables so that we can configure these at runtime or change it without touching the code.

You can clone the repository and run it on your local machine. You can access the application on localhost and on port 3000.

// clonegit clone https://github.com/bbachi/react-azure-static-web-app.git

// Run the apped todo-appnpm installnpm start

// Run the APIcd apinpm installnpm start

HOW TO CONFIGURE IN LOCAL DEVELOPMENT

First, we need to understand how we can get the settings from the API before we load the React app. You need to understand the React Hooks for that, especially, **useEffect**() with an empty array.

If you look at the below example, we are fetching the data in the react hook. If we pass an empty array to the hook, it is called once after the component is mounted.

https://gist.github.com/bbachi/4185e7af857505803d544309c0399bc1#file-app-js

A.API

Let's see how we can pass environment variables in local development. When you create an Azure Function NodeJS project with Azure Core Tools, the file called *local.settings.json* is created in which you can put variables. For example, let's define some variables for testing.

https://gist.github.com/bbachi/8d2ea9610d8cba6fa9cd36824bb8c960#file-local-settings-json

We have defined variables in the settings JSON. Let's create an API to fetch these settings and also, we will see how to read these in the code.

https://gist.github.com/bbachi/78f2f33d5394636541b091efc0cff2d0#file-function-json

We have defined a function and exposed these settings on route /settings. Notice that we are reading environment variables with process.env. Let's run the API and test the API as below.



Figure 1: API Running

Let's hit that URL in the browser and see that in action.



Figure 2: API working

B. React

We have created an API that reads the environment variables and exposes that as a GET REST API. We will know how we should read this in the React application. We need to get these settings before we load the app since the header is the first one to load in the browser. We must use the react hook useEffect as mentioned above. Let's define the function in the service which calls the settings API and get those details. Here is the service file. https://gist.github.com/bbachi/dd0f7443ac71522dfd61b38c48b75daa#file-todoservice-js

Once you define these you need to import this in the App.js as below. Notice that we are calling the API through App Service. We are defining the state variable for the settings and calling the API in the useEffect() lifecycle hook as below. Since we are passing the empty array, it is called only once and we are setting the state variable *appSettings* and pass that to the Header component.

https://gist.github.com/bbachi/c474a1b321cdaf23013150b70b2fcccc#file-app-js

Here is the Header Component that takes the props and loads only when these are available. https://gist.github.com/bbachi/3840fb73801b832f77f482538903778a#file-header-js

We are using the Style property to dynamically apply these styles to the Header after reading and setting these variables in the Component file. You can see those styles applied all the way from the local settings file.

ToDo L	.ist				
Task			Assignee		
Create a Ta	ok.		Assignee		
Status:					
To Be Don					
Submit					
Tasks					
Tasks Taskid	Task Name	Assignee	Status		
Scient Tasks Taskid	Task Name task1	Assignee assignee1000	Status completed	ter	Delete
Schmet Tasks Task Id 1	Task Name task1 task2	Assignee assignee1000 assignee1001	Status Completed completed	East East	Delete

Figure 3: Styles applied to the Header

HOW TO CONFIGURE IN OTHER ENVIRONMENTS

Let's push the code and see how we can configure the same thing in Azure. You can see the deployment started after you push changes to GitHub.

○ Code ① Issues (§) □] Pul	requests O Actions Projects (1) Will	O Security 1	⊻ Insights 1(3) Se		
added settings Azure Static	: Web Apps CI/CD #8				
) Summary	Triggered via push 2 minutes age	Status	Total duration		
	bbachi pushed <> 33x8131 suster)	in progress	-		
Build and Deploy Job					
Close Pull Request Job	azure-static-web-apps-gray-tree-04c57000f.yml				
	Build and Deploy Job to 336				
	Close Pull Request Job Co.				

Figure 4: Deployment in Progress

Once the deployment is completed you can see the status in the Azure portal as below.

P Search (Cmd+/)] < O Refvesh ≘ ⊃	dette:			
Duriview	Queins environments as	a suffernationally created when a real service	is descented, and are non-solid lists	make too over the rad	I nervent is married.
Access control (AM)					
Ø Tags	Production				
lattings	There is one production	environment per app. Check its status	and last update time here.		
Configuration	Branch	Last Update Time	Status		Browse
Custom domains	master	March 15, 2021, 12:1	19.12 AM CDT O Ready		Browne CS
// Functions			CONTRACTOR CONTRACTOR		
Environments	Staging				
* Role management	Open pull requests again	inst the linked repository to create a sta	ging environment.		
🗄 Locks	Title	Branch	Last Update Time	Status	Browne

Figure 5: Deployment Status

You can load the application now in the browser without any environment variables defined. You can notice there is no background color and header text color.

Task Com Skitter To Task Task	re a Task k le Done S		Assignee Assignee		•
Cim Statu To / Taski Taski	ne a Task k Is Done K S		Assignee Assignee		v
Circ Statue To I Taski Taski	re a Task Ie Done Int		Assignee		×
Statur To I Taski Taski	s le Done Nit				×
Task Task	le Done				٣
Tasł ^{Taski}	ne . IS				
Task	is .				
Task	s				
Task					
	d Task Name	Assignee	Status		
1	task1	assignee1000	completed	Ede	Delete
2	task2	assignee1001	completed	Ede	Delete
3	task3	assignee1002	completed	1.44	(interested)
Elements Cornelle	Bourses Network Parlamence	Mamory Application Security U	ightouse Augury AngularJS	tot	a concernant
😵 Q. 🗆 Pereve la	Diadre cathe Ontre + 1				
0.84	data URLs [] XHR JS CSS imp N	hedia Farit Doc WS Manheit Other	This blocked cookies [] Blocked Requests	1	

Figure 6: Application without Configuration

Let's define these configuration variables by clicking on the configuration on the left blade.

TodoAppReact	Configuration -								
P Search (Cmd+/)	• Eine X bunt Or	etech							
Chenview Access control (AM) Tags	Application settings Application settings applicable display them in plain text in you	to your site are encry r browser by using t	pted at rest and to he controls befow.	anamitted over	r an encrypted ch	unnel. You	can choose	e to	
Settions Configuration Costom domains	Environment Production	V Advanced edit	8 Debre						
W Punctions	V Filter application setting								
Role management	Name T	Valu	*						
🛆 Locks				No applicat	ion setting to digita	v			
Automation									
Export template									

Figure 7: Configuration

We have defined these variables below.

TodoAppReact	Configuration					
Search (Cmd+/)	🛛 « 🔛 Save 🗙 Discard 🕻) Refresh				
Overview	Application settings					
Access control (IAM)	Application seconds					
Tags	Application settings applicat display them in plain text in	Application settings applicable to your site are encrypted at rest and transmitted over an encrypted channel. You can choor display them in plain text in your browser by using the controls below. Learn more				
Settings						
Configuration	Environment Production	Environment Production V				
Custom domains	+ Add Show value	Advanced edit 📄 Delete				
(A) Functions	Filter application setting	ng				
Environments	Name 1	Value				
7 Role management						
A Locks		Hoden value, cuick to show value				
	HEADER_TITLE_COLOR	Hidden value. Click to show value				
Automation						
👃 Tasks (preview)						
(B) E-most termelate						

Figure 8: Environment Variables

You can refresh the page after we define these variables. You can see those styles applied to the header.

ry-tree-04c\$7000f.azurestaticapps.net			\$ 0 9 6 0 0 Y					
	Azure	Static Web /	Apps With	React Demo				
ToDo L _{Task}	ist		Assignee					
Create a Ta	sk		Assignee					
Status:								
To Be Done	É.				*			
Tasks								
Task Id	Task Name	Assignee	Status					
1	task1	assignee1000	completed	Edit	Delete			
2	task2	assignee1001	completed	Edit	Delete			
3	task3	assignee1002	completed	Edit	Delete			
4	task4	assignee1000	completed	Edit	Delete			

Figure 9: Styles applied to the header

You can see the API call working and fetching those values.

Status:	1
Elements Console S	Sources Network Performance Memory Application Security Lighthouse Augury AngularJS
O	Disable cache Online + 1 ± ±
Filter Hide	data URLs 🔝 XHR JS CSS Img Media Font Doc WS Manifest Other 🗌 Has blocked cookies 🗌 Blocked Req
100 ms 200 ms	300 ms 400 ms 500 ms 600 ms 1000 ms 1000 ms
Name grap-thee-OdeS70001 azvmetaticapps. 2.11829350 chunk.css main:d301cHed chunk.css 2.900 tal87 chunk.js main:d507248 chunk.js 50dos 50dos 64510ps 64510ps 64510ps	<pre>x Headers Periode Response Metador Timing 1 { TenderCaler"; "Lightble", TenderTitleCaler"; "red" } </pre>
manifest.json	

Figure 10: Settings Call

SUMMARY

- Building with Azure Static Web apps service is one of them and it is recently released by Microsoft Azure and it's in the preview mode.
- We usually separate some configuration stuff from the code so that we can configure that at runtime, or it becomes easier to maintain because you don't have to build the code for every environment separately.
- You need to understand how Lifecycle Hooks works in React since we need to get these settings before you load the app in the browser.
- When you create an Azure Function NodeJS project with Azure Core Tools, the file called local.settings.json is created in which you can put variables.
- You must configure environment variables in the configuration section under settings in the Azure Static Web Apps Service.

CONCLUSION

In conclusion, this paper has detailed the process and advantages of utilizing the Azure Static Web Apps service, a recent innovation by Microsoft Azure currently in preview mode, for building and deploying web applications. We highlighted the importance of decoupling configuration details from the application code, which not only simplifies maintenance but also facilitates the customization of runtime environments without necessitating multiple builds for different settings. Furthermore, we emphasized the critical understanding of React Lifecycle Hooks to effectively manage application settings before browser loading. The integration of Azure Function NodeJS projects with Azure Core Tools, which generates a local.settings.json file for variable storage, was discussed to illustrate practical implementation.

Lastly, the paper outlined the necessary steps to properly configure environment variables within the Azure Static Web Apps Service settings, underscoring the significance of environment management in modern web development. By adopting these practices, developers can enhance the flexibility, maintainability, and scalability of their React applications, leveraging the full capabilities of Azure's cloud infrastructure for efficient web app deployment.

REFERENCES

- [1]. React Official Documentation https://react.dev/
- [2]. Azure Documentation https://learn.microsoft.com/en-us/docs/
- [3]. Azure Static Web Apps https://azure.microsoft.com/en-us/products/app-service/static