The Practical Guide to Building LiveChat Apps: Deploy your app. Netlify, Firebase, and Heroku

An overview of hosting services

Hi! This is Oliwia from the LiveChat Developer Program, and right now you're watching the Practical Guide to Building LiveChat Apps. In this episode, we'll teach you step-by-step how to easily deploy your production-ready application to an external hosting service.

So for your application to be available publicly, you need to host it somewhere for users to access. Usually, it's a service of sorts. So, we selected three from the most popular hosting services — Netlify, Firebase, and Heroku. In this video, you'll see how are they different from each other and get a better understanding of the solution you might want to choose for your own app.

So for each of the services we'll deploy our application to, we selected the easiest route for the app deployment. So, knowing that, you'll be able to determine whether you want to take up on a bigger challenge, or rather stick with the simplest solution.

If you wish to know even more about possible hosting options, be sure to have a look at our article that breaks down many different hosting solutions on the market. We categorized them by possible apps you're likely to build, so it's very easy to navigate and test yourself.

Hosting on Netlify (easiest)

So without further ado, let's start with the solution we think is the most simple one — Netlify. Naturally, you need to register your account prior to deploying your app there.

So, right now, you can see that we're in the Netlify's main team view. Before we deploy our application, we need to create a production build of the app in our repository file directory. We'll use a simple `npm run build` command. With that, we'll have a production-ready build that we can deploy directly to Netlify.

Currently, we don't have any sites created in our Netlify team account, so let's create the first one for our app. For that, we'll use the simple drag-and-drop option, and we'll drop our build folder from the repository's directory to Netlify deploy field.

Alright! The app has been deployed, and we got the URL to access it directly. So, for our app to work in the LiveChat Agent App, we'll need to modify our Authorization Block in the LiveChat Developer Console first. And what we need to do here is to add the application's URL to our app's redirect URIs whitelist. We need to do this as without this step, our app will not be authorized to receive an access token on the newly deployed URL. And, the last thing we want to do is to update our Agent App Widgets block to include the new app's URL. As you may remember, by far, our app's been working directly from our local environment, a localhost. So now, after it's been deployed to a public hosting, we can access it externally. So we can freely switch out the localhost from the widget source URL to our new app's URL from Netlify.

Alright! And once that's done, we can check in the LiveChat Agent App if everything went smoothly. Yep! Looks like everything works perfectly fine, our app's authorized.

Okay! That would be all for Netlify — super easy, right? We find Netlify a brilliant solution for app deployment. And we also use this service internally, so we can recommend this as a great option to host your app.

Hosting on Firebase

Alright! Firebase is the next solution we'll teach you how to use. It requires a bit more steps, but it is still a very simple and accessible hosting to deploy your application. So before you start, make sure you register an account with Firebase. In the browser's address bar, you'll see the URL to the Firebase's console.

So to deploy our app, in this console, we want to create a new project and name it. Right now, we're asked if we want to use Google Analytics — personally, we don't need that for our application, so we'll opt out from this. And now, we need to wait a moment, so Firebase creates the project for us. Alright! Once that's done, we can see the overview of our created project.

For this project, we'll be using the Firebase CLI. You'll find information on how to install this CLI on your desktop in the Firebase's documentation. It's initially either installing this with a simple curl, or you can do this by npm. We'll skip this step, as we have already installed this CLI locally on our machine.

Alright, so in a terminal opened in the repository's file directory, similar to Netlify, we will again create a production build using the `npm run build` command.

So now, we'll be using the Firebase CLI. And we'll input the `firebase login` command to authorize our terminal first. Now the CLI will ask us if we want to share reporting data. And now, we want to authorize our application via our Google account, and authorize with Firebase. After all that's done, we see the success information.

Next, we want to initialize our Firebase project in the repository we've already opened in the terminal, using the `firebase init` command. Now, we will have to go through some Firebase configuration. So first, we're asked what we want to use the Firebase tool for — we're interested in the hosting solution, so we'll select this option navigating with arrows, and confirm this by pressing enter.

Next, we'll have to select if we want to use an existing project or create a new one for our app. We've already prepared a project, so we'll select the "existing project" option. And, select our Tag Master project.

Then, we want to specify a public directory for our project. The public option is the default one, but note in the case of a React application like ours, the public directory would be "build". Next, we're asked whether we want to configure our app as a single-page app, which in our case would be "no". We're also asked if we want to set up automatic deploys with GitHub. This means that in case of automatic changes, the new Firebase builds would be created. And we're not interested in that, so we'll press "no". Next, we can separately select if we want to rewrite 404 and index files. We don't want to rewrite either of these, so we'll select "no" to those questions as well. Alright! And after all those questions, the Firebase initialization is now complete.

So since we created a production build, authorized the terminal, and initialized Firebase, we can now deploy our build to the Firebase project by typing the `firebase deploy` command.

Okay! So that would be all that we'd need to do to deploy our application through the Firebase CLI. And now, our application is available under the URL returned through the terminal.

So for our application to work through this URL, we'll need to perform the same adjustments as we did in the case of Netlify — add this URL to the Redirect URIs whitelist and switch our localhost widget source with the new URL to the app. So let's do that.

And as you can see here in the LiveChat Agent App, the app works fine as well. Now it's fetched from the external URL deployed to Firebase. Great!

Hosting on Heroku

Okay! Now we can move to the last hosting service in this episode, which is Heroku. The process of deploying the app here is pretty similar to Firebase. And, we'll also be using a dedicated CLI for that.

Alright, so now we're logged in to our Heroku account, which you'll also need to register in. So here, we'll create a new app, give it a unique name, and select the Europe region, as that's where we're located.

In the Heroku's dev center, you'll find an installation instruction on how to install the Heroku CLI we mentioned at the beginning. We already have it installed on our machine as well, so we'll skip this step, but note that it is mandatory for you if you don't yet have this on your computer.

Now, we can open our terminal, and authenticate it for Heroku to access using the `heroku login` command. Then, we can log in to Heroku in our browser. As soon as we're authenticated, we can initialize the repository with a `git init` command. Once that's done, we will connect the CLI with our repository. So we'll type a command saying: `heroku git:remote -a`, and the name of our app we created in Heroku.

Now, the CLI is connected with our repository! So, unlike other deployments we did today, for Heroku, we'll need to create some adjustments in our application's code. So let's go to our code editor and open the package.json file.

The first thing we need to change is to remove the `HTTPS true` flag. We do that as this will be processed on Heroku's side now, and if we'll leave it, Heroku will return us with errors. Next, we'll need to determine the engines we're working on, and that would be the node and npm versions. We need to determine that so our build builds correctly in Heroku. Next up, we'll need to prepare a relevant commit and push the changes we've just made.

And as soon as that's done, we'll deploy our application by typing the `git push heroku master` command. Now, all of the files from our repository will be pushed to the Heroku's server, which will create a production build for this application.

Alright! Once that's done, we see a URL for our deployed application in Heroku dashboard, as well as in the terminal. And, same as before, we'll need to adjust our authorization in the LiveChat Developer Console, so this URL receives an access token without issues.
So we'll add this URL to the Redirect URI whitelist and replace this as the Widget source URL as well. And, as soon as we have that, we can go to the Agent App to see if our app works fine.

Looks like everything's A-OK, so that's brilliant! Another successful deployment behind us.

Summary

Alright, so that would be all for this episode!
We're hoping that by now, you have a better understanding of app deployment, and maybe even have an idea of where you should deploy your very own application. Remember to have a look at the app hosting article if you want to get an even more in-depth overview of what options are available for you.

In the next episode, we'll teach you all you need to know about the LiveChat app review process and how to make your app the best it can be.

So, see you soon!