Group 7:
Jada Dolby
Jasmine Masopeh
Sabrina Polynice
Raul Gonzalez Cespedes

InnerFlock Deployment Plan:

Taking in factors such as scalability, flexibility, ease of use and familiarity, our team decided that using Render would be the best course of action to deploy our application. Render provides a streamlined approach to hosting, scaling, and managing web applications in a distributed environment, making it easy to deploy and maintain our application as it grows. Furthermore, a good majority of our team has experience deploying applications on Render and would lead to a smoother deployment process.

Renders web service platform provides a lot of scalability functionalities such as manual or automatic control of server instances, persistent disks, instant rollbacks and more. Its infrastructure eliminates the need for manual server configuration, which frees time and resources to focus on building and deploying our application without having to worry too much about infrastructure management. Render has many basic application templates such as Express, Rocket, Laravel, and most importantly for our application – Django. Lastly, Render has seamless Git integration, allowing for automatic deployments upon code changes which facilitates continuous integration and deployment workflows (CI/CD).

To begin the deployment process, first a Render account must be established, in which we then select the web service option from its list of possible deployment services. We then connect the Render service with the Git repository containing our project. After which, we must set the necessary prerequisites on our code base. These prerequisites include creating file 'requirements.txt' to contain all the projects dependencies, installing gunicorn, and adding the web service host name (given in the RENDER_EXTERNAL_HOSTNAME environment variable) to the list of allowed hosts in Django.

It is at this point that we are in the Render dashboard where we are free to adjust things like deployment root, add our command to install requirements.txt, add our gunicorn command to run the project, and establish any environment variables that we might need to add. After which, we set automatic deployment, so that upon any code changes pushed to the connected Git repository will prompt deployments.

Once everything has been set, Render starts deployment in three stages. First is build, Render will detect any changes made and start the building process. This building process entails pulling code from the repository, installing dependencies, compiling code and all other build steps specified in project configuration.

After the build process, Render creates a container image that encapsulates the application and its dependencies. This container is lightweight and contains things such as the runtime environment, libraries and application code that is all needed to run the application.

Finally, Render takes to container image and deploys it to its infrastructure. Renders deployment process starts by spinning up at least one instance of the application in a containerized environment. Render will then manage the monitoring and scaling of the application automatically.

Renders architecture aligns with the principles of distributed programming, enabling control over scalability through distributed systems. By Leveraging Renders managed infrastructure, we are able to deploy our application across multiple nodes, benefiting from shared load balance, fault tolerance and horizontal scalability. Furthermore, in context of our project, Renders ability to automate scaling services pairs well with the nature of user-to-user messaging applications. As the user count fluctuates, the amount of server instances being reserved will change in accordance with the demands of the application. With an application such as ours where there may be moments of high and low traffic, this feature allows us to streamline our resource use and make sure we are only using as much as we need at any given moment. Establishing automatic deployments and having a CI/CD workflow is also crucial for this type of application, as any delay in fixing any issue that may arise in such a dynamic and interactive system could be detrimental.