ShipM8

https://github.com/oslabs-beta/shipm8

Stack

AWS (IAM)
React Native
Amazon Elastic Kubernetes Service (EKS)
Google Kubernetes Engine (GKE)
Google Cloud Platform (GCP)
Kubernetes
Docker
AsyncStorage
React Hooks
React Native Elements
Redux
Redux-Toolkit
Redux-Thunk
Redux Persist
React Navigation
Jest

Background

With a rise in implementation of micro service architecture and containerization, Kubernetes (a container orchestration system) has become a key player in the management and deployment of containerized applications. Currently, there are no free, open-source, mobile applications for monitoring Kubernetes clusters, until now.

ShipM8 is an open-source React Native application developed for monitoring Kubernetes clusters hosted on AWS Elastic Kubernetes Service (EKS) and Google Kubernetes Engine (GKE). As of now, users can view cluster names and status based on region (AWS) or project (GKE), add specific clusters to keep track of, view pods in a particular namespace, view pertinent details of a selected pod, and delete pods from a cluster.

How it works

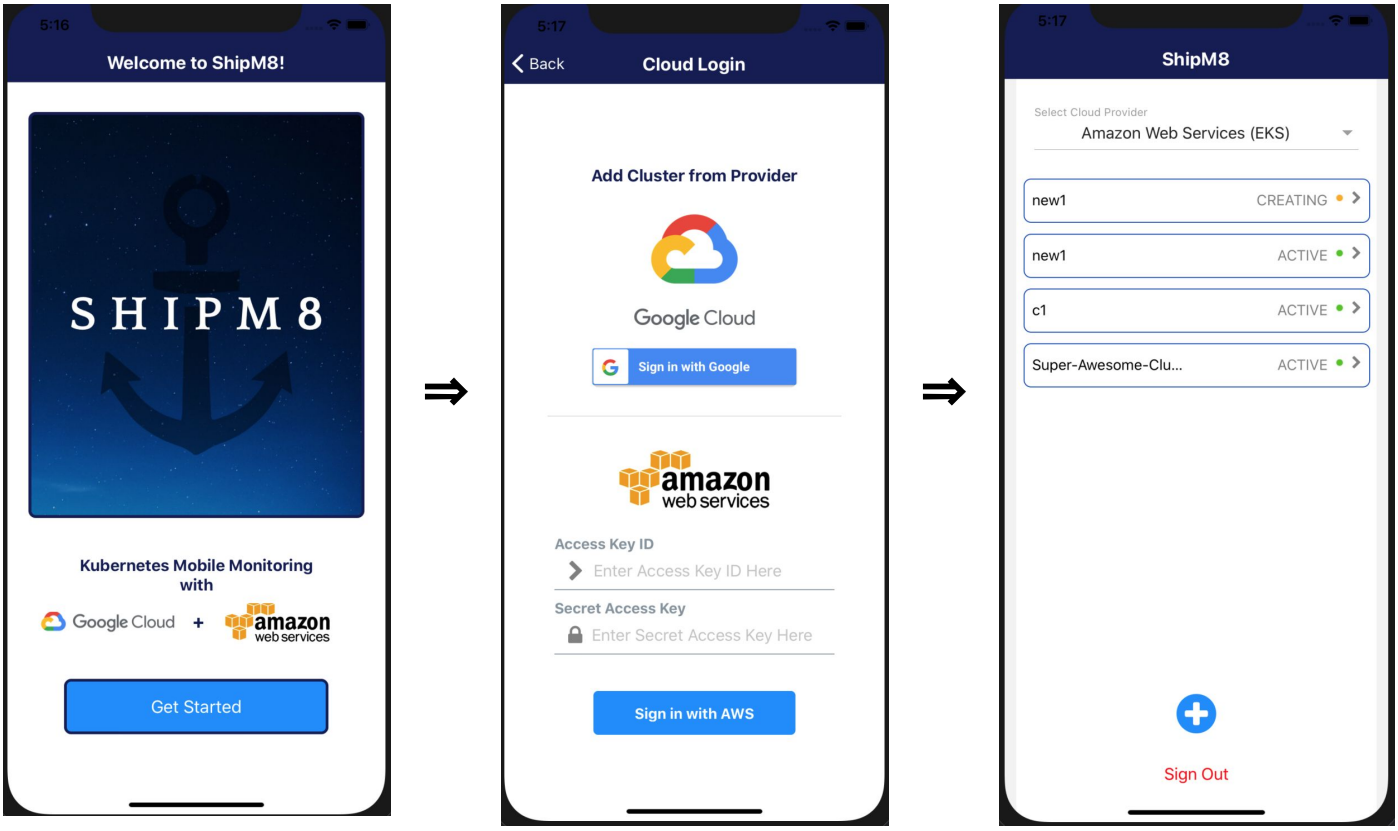
ShipM8 is designed with React Native allowing iOS users to install the app on their phone. ShipM8 connects to AWS using Amazon's Signature V4 and Google Cloud using OAuth 2.0. Clusters can be added for either cloud provider. Cluster information is pulled from the Kubernetes API using Bearer authentication.

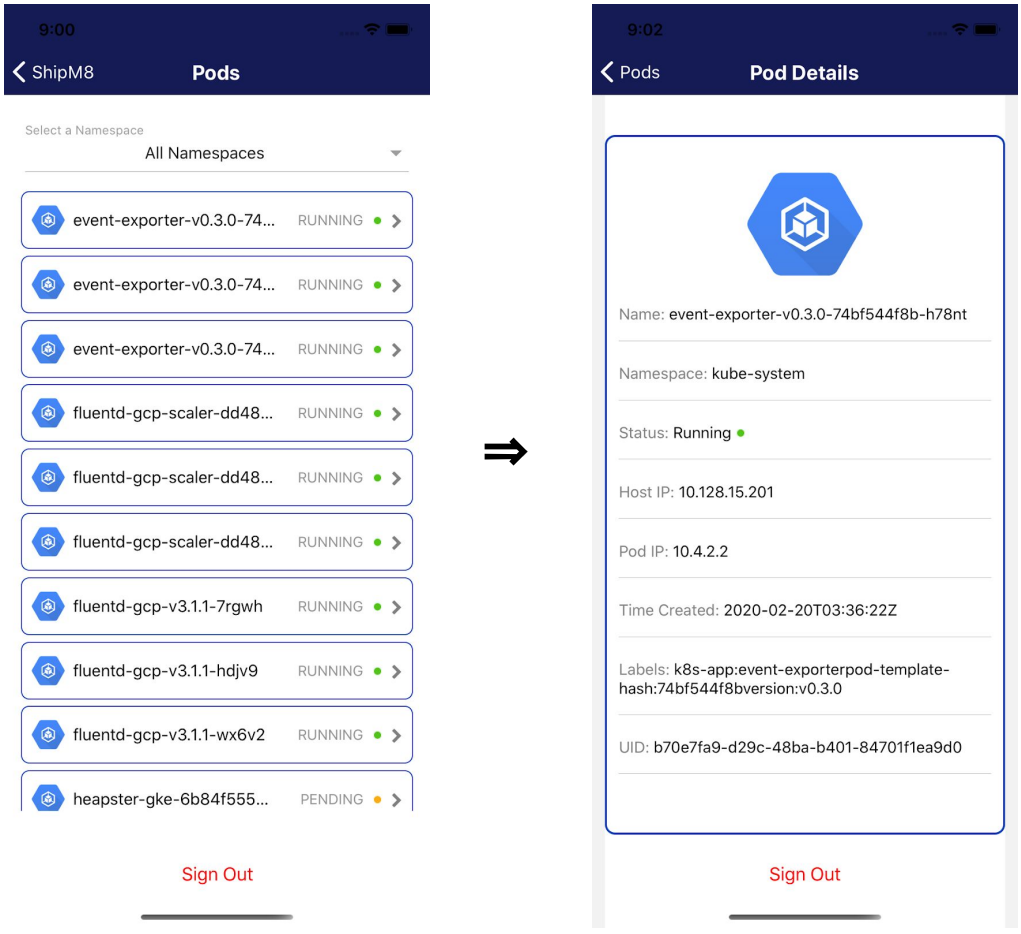
Adding a cluster to the app allows users to view cluster status and pods for that cluster. Clusters can be filtered by cloud provider using a dropdown. Pods are displayed in a list shown with current status. Selecting a pod allows users to view detailed information about that pod. Pods can be deleted from the cluster by swiping left on the pod and tapping delete.

To get started, you will need to have:

1. Kubernetes cluster(s) hosted on either Amazon Web Services, using Elastic Kubernetes Service (EKS), or Google Cloud Platform, using Google Kubernetes Engine (GKE)
2. If hosted on Google, sign in using Google OAuth
3. If hosted on AWS, you will need your Access Key ID and Secret Access Key (these can be found on your AWS account)
4. Once verified, choose your Kubernetes cluster that you would like to monitor by either Project (GKE) or Region (AWS)
5. Once you choose a cluster, you will be redirected to a list of the current clusters you are monitoring.
    a. Clusters can be filtered by cloud provider using the dropdown
    b. Feel free to add additional clusters to your list by tapping the blue plus button towards the bottom of this screen. You will be redirected to previous screen to add additional clusters
6. Once you have selected the specific clusters you would like to monitor, tap a cluster to view pods.
    a. Pods can be filtered by selecting a namespace from the dropdown
7. When you select a pod, you will be redirected to a pod specific details page, showing all pertinent information for that pod.

Screenshots





Technical Challenges

Authentication
Authenticating to AWS EKS clusters from outside of the cluster was probably our biggest challenge. The libraries that provide authentication for this purpose rely on Node core modules that are unavailable in React Native. Due to this, we had to engineer our own solution to gain access to EKS clusters. This involved generating a bearer token from a base64 encoded string of a signed HTTP request to Amazon's STS API, which was then included in all requests made to AWS EKS clusters.

State Management
Persisting state throughout the application was a primary challenge. Because we wanted the user to be able to save clusters and view them later when they close and reopen the application, we needed a way to persist state between app initializations. We solved this by utilizing Redux Persist, which saved our application state to the device's Async Storage, allowing us to rehydrate the state when the user reopens the app. Additionally, we were able to redirect the user to their saved clusters if they had used the app before, or to a splash screen if they had not.

Handling Asynchronicity
Our application relies heavily on making requests to the Kubernetes API, and initially we had trouble storing and displaying the data properly in components. To address this issue, we implemented Redux Thunk middleware to dispatch asynchronous actions to update our application state. Additionally, we used Promise.all to run multiple fetch requests in parallel where we needed the results of all requests to handle certain API functions.

Team Responsibilities

Luke Van Bergen - Responsible for developing client side of application, implementing React Native to create and render reusable components for various screen sizes, and implementing snapshot testing using Jest - Designing of Logo and various layouts of different screens

JJ Friedman - Responsible for developing login page functionality and design while implementing Redux and Google's API middleware.

Taylor Rodrigues - Responsible for implementing authentication (OAuth 2.0 and AWS EKS), designing APIs for cloud services (EKS, GKE), constructing app state structure (Redux Store) with slice reducers, configuring Redux Persist to persist application state between app initializations, and writing Thunks to update state asynchronously via API calls