How to Modernize Your App and Help Your Grandmother

One day in class, my college writing professor plead for no more “dead grandmother” stories. Her point was that lots of kids had experienced loss, but none had a unique angle on the story that would make it interesting to read. I did not have a dead grandmother story at the time but now I do and its also a “How To” that I think Dr. Welt, might like.

When my grandmother passed she left behind a lot of US savings bonds that identified her grandchildren as payable-on-death (POD) beneficiaries. Because I am the youngest and, clearly, the cutest of my siblings, I ended up with most of those bonds. Since then, I have done an extensive survey (of four people) and found that many people from that generation received a portion of their paycheck in the form of US bonds. Fortunately, I found the Savings Bond Wizard application from the US Treasury which I used to organize those bonds and save them in SBW files.

Need to Modernize the App

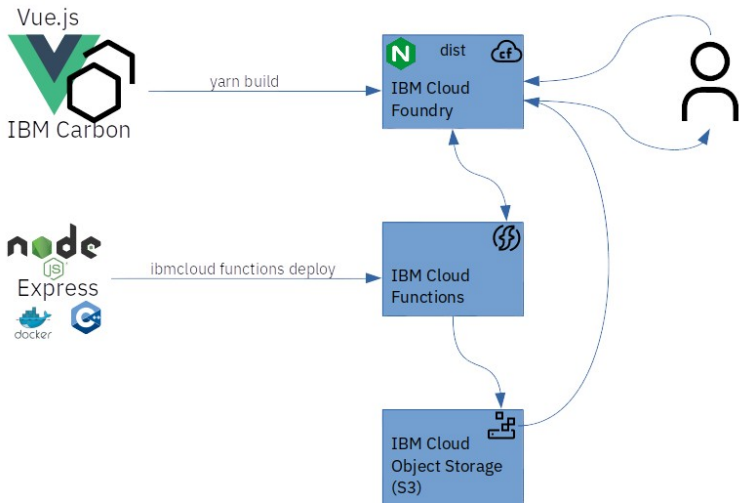
However, the Savings Bond Wizard app was discontinued in 2008. And, when my mother-in-law recently asked me to sort through some bond data saved in SBW files I couldn’t open the files. As with most things, there is an open-source solution; a GNOME program called GBonds. However, that didn’t work because of conflicting packages requirements. I did manage to get it running in a Docker image and it did successfully import the SBW files but GBonds does not offer a way to export the data. Thus, I had an opportunity to modernize an app.

OK it’s not a monolithic, enterprise WebSphere app but the use case is the same. It also begets the same questions: Do I “lift and shift” by trying to resolve the package issues I ran into with GBonds? Do I “lift and shift” by using the Docker image I created and run that in the cloud with some sort of scheme for exporting the X11 interface? Or do I use a phased strategy https://www.ibm.com/cloud/application-modernization:

- **Simplify**: containerize the application to reduce costs and simplify operations
- **Extend**: use APIs for existing applications that are difficult to enable for cloud
- **Decompose**: use microservices to break down monolithic applications into deployable components
- **Refactor**: add new microservices to innovate incrementally

The Technical Solution

In this case I **simplified** the app by making it headless and reducing it to just a CSV export. I did that by wrapping the GBonds import code with a little C++ glue to make it available as a CLI. With that function **decomposed,** I wrapped the CLI in an Node.js/Express microservice and deployed that to the cloud as an IBM Function/OpenWhisk action. And finally I wrote a minimal new cloud-based UI with Vue.js and deployed that as a Cloud Foundry app.



All the code is on GitHub here: https://github.com/davidnixon/sbw2csv and the app is deployed here: https://sbw2csv.us-east.mybluemix.net/#/