THIS HAS BEEN MOVED OVER TO

https://github.com/spinnaker/community/pull/74

Plugins UI Design Doc

Motivation

Just like the backend in Spinnaker, the frontend needs to be extensible for others to add more functionality. Deck should be able to be extended by well defined extension points. Some examples of extension points could be:

- Custom stages
- Modifying header and footer
- Custom search filters
- Custom routes
- Adding labels or other information to sections in the infrastructure tab

This design doc references the Plugins RFC.

This doc focuses on

- How plugins are implemented in Deck
- Deck ↔ Plugin Interaction

How Plugins are Implemented in Deck

Load Plugins into Deck

In order for plugins to have a UI component, they must be loaded into Deck's runtime. In order to load plugins to Deck's runtime, Deck needs to know two pieces of information:

- What plugins are enabled
- The location of the resources a plugin needs.

Plugin Configuration via Front50

In order for Deck to know which plugins it can load, it must have access to some configuration information, including the name of the plugin and where to download the plugin resource(s). Upon page load, Deck reaches out to Front50 to get the enabled plugin metadata.

Please see the current Plugins RFC for details around Front50 as the source of truth for plugin metadata.

Gathering plugin metadata

> The Deck microservice does not need to download plugin resources; however, the browser does need to know where the plugin resources are located.

Deck provides a function to lookup plugin metadata in Front50, which is exposed via Gate, so the browser can gather that information. In order for resources to be loaded appropriately, Deck must gather the resources prior to the application being bootstrapped.

Loading a Plugin

After Deck queries Front50 for enabled plugins, it gathers the metadata necessary to load the enabled plugins. Deck will then use native module loading to import all required plugins.

Downloading of plugin resources occurs before the app boots (Check out Bootstrapping Deck to learn more . Once those resources are downloaded, plugins are initialized as described below. We

accomplish this by moving the loading of the netflix.spinnaker module to a finally block after initializing the plugins. This ensures that it is always called. This is seen below:

```
loadPlugins()
  .catch(() => {
    // deal with errors
  })
  .finally(() => {
    // load netflix.spinnaker module
  }
```

let loadedPlugins = []

```
export function loadPlugins() {
  const plugins = fetch from Front50;
  return Promise.all(plugins.map(
      // import each plugin and add it to the loadedPlugins
      //   if a plugin fails to load, log it
  ));
}
```

initPlugins

After all of the plugins have been loaded via import, they need to be initialized. It will be ideal to have plugins not need to know about Deck internals to register themselves.

Stop Gap Solution for Registering Plugins:

At the very beginning there will not be well defined extension points for everything. To get around this, plugins may have to register themselves until a well defined extension point is created.

Well Defined Extension Points:

Interfaces need to be created for anything that can be extended.

For custom Stages do something like:

```
export interface IStageRegistry {
  label: string,
  key: string,
  description: string,
  component: any
}
```

Inside of Deck use the information defined in IStageRegistry to register the stage with the PipelineRegistry.

Bootstrapping Deck

Since we are changing how Deck starts up to load plugins, we must make some additional adjustments, such as changing how Deck gets bootstrapped. We have to delay starting AngularJS to load the plugins. In order to delay the starting of AngularJS, we have to manually bootstrap the application. There is another script that gets added as an entrypoint via webpack, called bootstrap-deck.js. We must move the bootstrapping of Angular into its own file, otherwise it will be called during tests. Not doing this causes modules to be initialized when they are not expected to be present, leading to test failures.

```
element(document.documentElement).ready(() => {
  bootstrap(document.documentElement, ['netflix.spinnaker']);
});
```

Known Unknowns

- How do we ensure end users can download plugin resources?
- Do we show a warning if we fail to reach Front50?
    - Spinnaker alert/banner at top of page should inform the user that plugins were not able to be loaded
        - Only show this warning if plugins are enabled
        - Give guidance on how to resolve error or look up more info
- What if we fail to download a plugin resource? And how would we catch that (eg if its in a script tag)?
    - plugin manifests will contain locations where assets are, but another place may serve them (eg secure artifact store), can halyard help with this?

Alternatives to Loading

WebPack

We may be able to modify Webpack to download plugins and compile Deck at deploy time. This eliminates the need for users to reach out to an artifact store and leads to a smoother UX. It would increase deploy times by a lot though, so that is something to consider. Going this path would also mean that Deck would have to be redeployed any time a new plugin has been added.

RequireJS

RequireJS could be used to load plugins. When doing initial prototyping with RequireJS, it was complaining at compile time about not being able to find plugins because they are supposed to be

added at runtime, not compile time.

Deck ↔ Plugin Interaction

Expose Spinnaker Dependencies

When plugin creators create plugins, they should be using versions of dependencies that match what Spinnaker has and they should not have to bundle them with the plugin. Spinnaker needs to make some of it dependencies available to the plugins.

Here is a link to a pull request that implements this: https://github.com/spinnaker/deck/pull/7662 Currently it also exposes @spinnaker/core to the plugins. This way plugins can get a shared set of components to use. The ideal path going forward is to remove the shared UI components to their own library.