Getting Started

This document describes how to setup your development environment.

Architecture Overview

Kubernetes Dashboard project consists of two main components. They are called here the frontend and the backend. The frontend is a single page web application that runs in a browser. It fetches all its business data from the backend using standard HTTP methods. The backend implements UI-business logic and fetches raw data from the various Kubernetes APIs.

Preparation

Make sure the following software is installed and added to the `$PATH` variable: * Docker (1.10+) * go (1.5+) * nodejs (5.1.1+) * npm (3+) * java (7+) * gulp (3.9+)

You can follow detailed steps on how to install these requirements.

Clone the repository and install the dependencies:

```
$ npm install
```

Run a Kubernetes Cluster

For development it is recommended to run a local Kubernetes cluster. For your convenience, a task is provided that checks out the latest stable version, and runs it inside a Docker container. First, a Docker setting is required to be adapted. Execute the following script, but only once:

```
$ sudo ./build/setup-docker.sh
```

Then, open a separate tab in your terminal and run the following command:

```
$ gulp local-up-cluster
```

This will build and start a lightweight local cluster, consisting of a master and a single node. All processes run locally, in Docker container. The local cluster should behave like a real cluster, however, plugins like heapster are not installed. To shut it down, type the following command that kills all running Docker containers:

```
$ docker kill $(docker ps -aq)
```

From time to time you might want to use to a real Kubernetes cluster (e.g. GCE, Vagrant) instead of the local one. The most convenient way is to create a proxy. Run the following command instead of the gulp task from above:

```
$ kubectl proxy --port=8080
```

kubectl will handle authentication with Kubernetes and create an API proxy with the address `localhost:8080`. Therefore, no changes in the configuration are required.

Serving Dashboard for Development

It is easy to compile and run Dashboard. Open a new tab in your terminal and type:

```
$ gulp serve
```

Open a browser and access the UI under `localhost:9090`. A lot of things happened underneath. Let's scratch on the surface a bit.

Compilation: * Stylesheets are implemented with SASS and compiled to CSS with libsass * JavaScript is implemented in ES6. It is compiled with Babel for development and the Google-Closure-Compiler for production. * Go is used for the implementation of the backend. The source code gets compiled into the single binary 'dashboard'

Execution: * Frontend is served by BrowserSync. It enables features like live reloading when HTML/CSS/JS change and even synchronize scrolls, clicks and form inputs across multiple devices. * Backend is served by the 'dashboard' binary.

File watchers listen for source code changes (CSS, JS, GO) and automatically recompile. All changes are instantly reflected, e.g. by automatic process restarts or browser refreshes. The build artifacts are created in a hidden folder (`.tmp`).

After successful execution of `gulp local-up-cluster` and `gulp serve`, the following processes should be running (respective ports are given in parentheses):

BrowserSync (9090) —> Dashboard backend (9091) —> Kubernetes API server (8080)

Building Dashboard for Production

The Dashboard project can be built for production by using the following task: `$ gulp build` The code is compiled, compressed and debug support removed. The artifacts can be found in the `dist` folder.

In order to serve Dashboard from the `dist` folder, use the following task:

```
$ gulp serve:prod
```

Open a browser and access the UI under `localhost:9090`. The following processes should be running (respective ports are given in parentheses):

Dashboard backend (9090) —> Kubernetes API server (8080)

In order to package everything into a ready-to-run Docker image, use the following task:

```
$ gulp docker-image:canary
```

You might notice that the Docker image is very small and requires only a few MB. Only Dashboard assets are added to a scratch image. This is possible, because the `dashboard` binary has no external dependencies. Awesome!

Run the Tests

Unit tests should be executed after every source code change. The following task makes this a breeze by automatically executing the unit tests after every save action.

```
$ gulp test:watch
```

The full test suite includes static code analysis, unit tests and integration tests. It can be executed with:

```
$ gulp check
```

Building Dashboard Inside a Container

It's possible to run `gulp` and all the dependencies inside a development container. To do this, just replace `gulp [some arg]` commands with `build/run-gulp-in-docker.sh [some arg]`. If you do this, the only dependency is `docker`, and required commands such as `npm install` will be run automatically.

Contribute

Wish to contribute? Great, start here.