Deploying Ratpack Applications to Google Cloud AppEngine

Daniel Hyun

2016 July 04

Table of Contents

1. Intro — tl;dr

Google Cloud App Engine has new "Flexible Platform" in beta.

The new platform allows more "flexible" deployments, meaning you can use Java 8 and ship webapps with their own networking abilities.

Because Ratpack is just a set of Java libraries, it's very easy to deploy a Ratpack application to the new Flexible Platform.

Because Ratpack applications are simple Java SE applications, this post can also be applied to general Java 8 Applications being deployed to GAE.

This post is also available in pdf format

2. gcloud

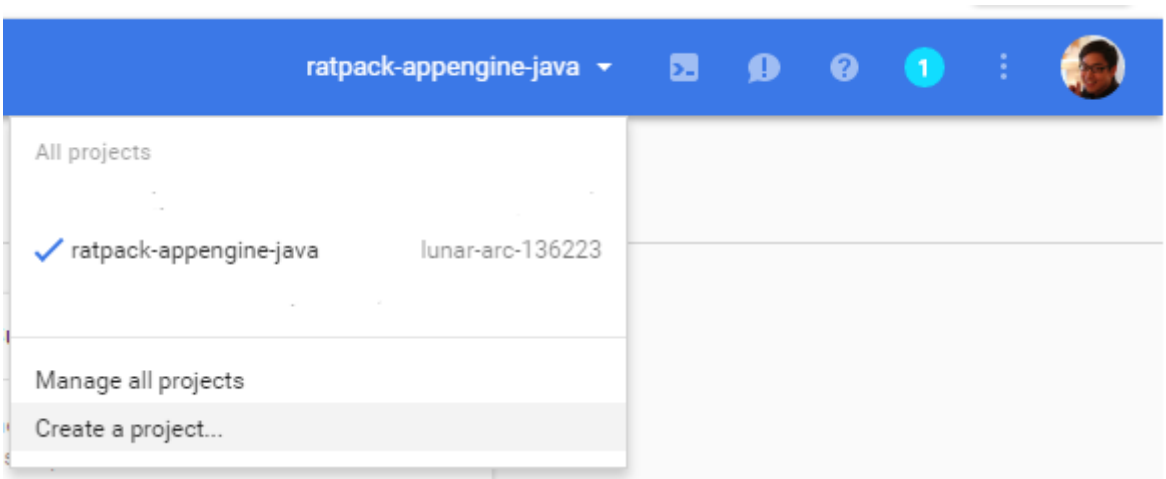
The Google App Engine docs for Java use Maven in their examples. There's no need to use Maven or the Maven plugin to manage your GAE apps. Instead I recommend you use the `gcloud` CLI tool. You can download the tool here.

The CLI tool is a set of Python scripts with shell scripts that know how to execute your commands.

Guillaume Laforge — (Groovy project lead and Google Cloud platform advocate) — pointed me to the official Gradle plugin for GAE. I have not used it but reportedly Guillaume and Vladimir Orany use it for managing Gaelyk applications.

3. Creating a new project

At the time of writing there is no easy way to create new GAE apps from the CLI tool. You need to create new project from web console (CLI access only available via the Alpha API — which requires whitelisted access).



After creating your project invoke via `glcoud`:

Interactive command line exchange that associates your machine with a GAE project

```
gcloud init
```

This command will make your configuration the same across all terminals in space and time. You can reconfigure the settings but you cannot have two terminals open with settings for two different GAE projects.

4. Preparing your Ratpack application

We will use a sample CORS enabled REST application in Ratpack written in both Java and Groovy. There are well detailed guides that explain how the Java and Groovy versions were written.

The Ratpack Gradle plugin applies the `application` plugin which also applies the `distribution` plugin. The distribution plugin provides the `distTar` and `distZip` tasks that will package your entire application as a tar or zip file.

```
$ ./gradlew clean distTar -i
```

We can use this because the base Docker image has `env` and `bash` so allows us to execute script. The start scripts generated by the application plugin make use of both `env` and `bash` so it's worth checking that your target environment support these.

Here is the `app.yaml` file that we will use for our application:

app.yaml

```
runtime: custom ①
vm: true ①

manual_scaling: ②
  instances: 1
```

① Directives to use the new beta flexible platform offering

② Start the app on one instance, [autoscale options available]

If you're curious about what GAE will look for in your `app.yaml` file feel free to check app yaml docs. In addition to the `app.yaml` file you'll also need a `Dockerfile` to instruct the "custom vm" how to invoke your application.

Dockerfile for the Java version

```
FROM gcr.io/google_appengine/openjdk8

ADD build/distributions/ratpack-appengine-java.tar /

①
ENV JAVA_OPTS='-Dratpack.port=8080 -Djava.security.egd=file:/dev/./urandom'

②
ENTRYPOINT ["/ratpack-appengine-java/bin/ratpack-appengine-java"]
```

① By default traffic will be served out of port `8080` so we instruct Ratpack to bind to the port accordingly.

② We set the entrypoint to the start script generated from the Gradle application plugin.

Here is the `Dockerfile` for the Groovy version of the application, note the only change is the name of the artifact and start script.

Dockerfile for the Groovy version

```
FROM gcr.io/google_appengine/openjdk8

ADD build/distributions/ratpack-appengine-groovy.tar /

ENV JAVA_OPTS='-Dratpack.port=8080 -Djava.security.egd=file:/dev/./urandom'

ENTRYPOINT ["/ratpack-appengine-groovy/bin/ratpack-appengine-groovy"]
```

Thanks to John Engelman — of the Gradle Shadow plugin — for the pro tip, add tar automatically decompresses the tar file

5. Deploying to Google Cloud App Engine

Once you have `gcloud` installed and the application prepared via `app.yaml` and `Dockerfile`, you can proceed with deployment by invoking:

```
gcloud preview app deploy
```

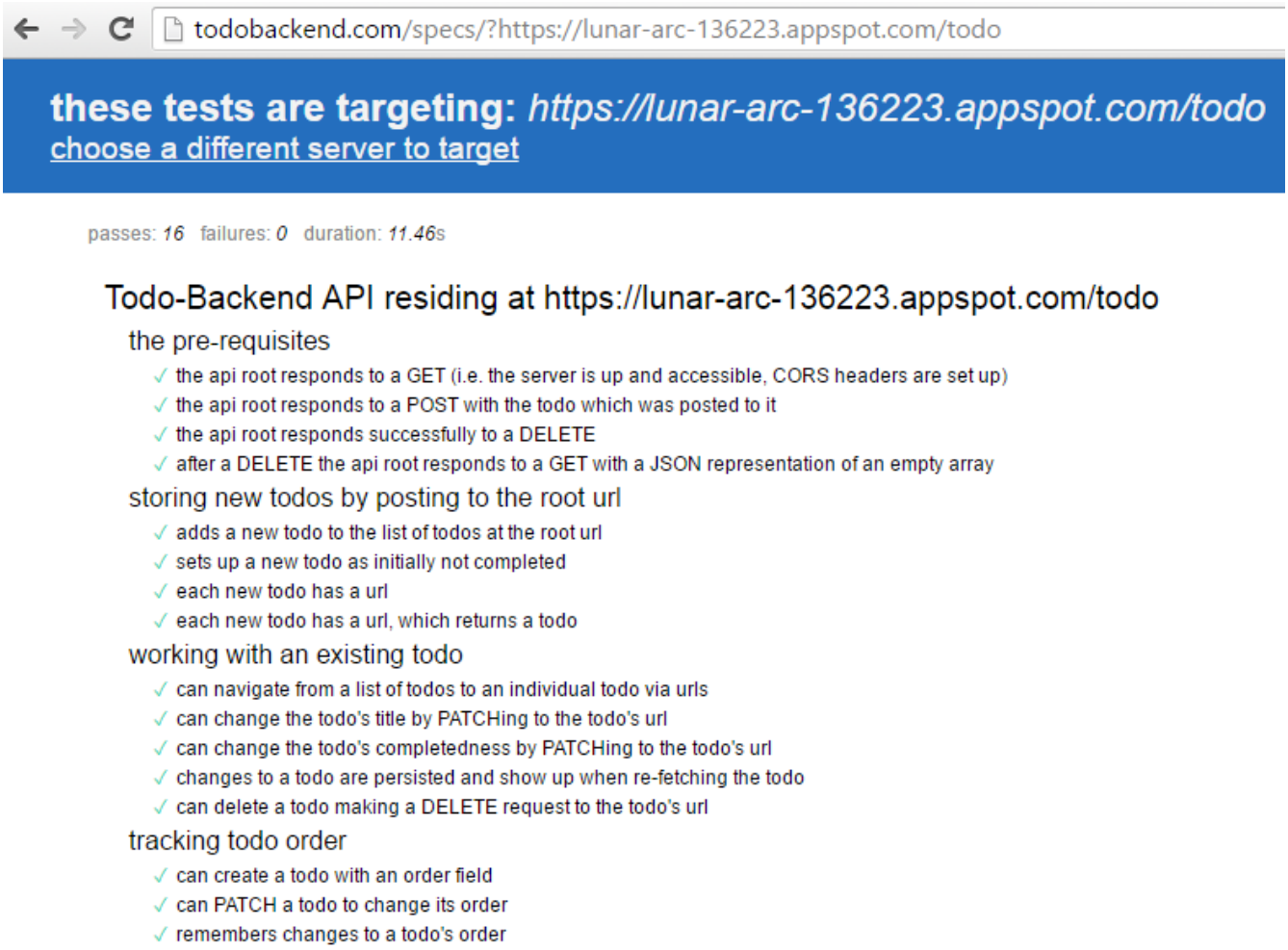
Once the application is deployed you'll see the following message in your stdout:

Successful deployment

```
You can read logs from the command line by running:
  $ gcloud app logs read
To view your application in the web browser run:
  $ gcloud app browse
```

6. Giving it a spin

Because our application is built to the TodoBackend spec we can check to ensure that our application passes the spec.



7. Notes

GAE will create 2 instances of your application and load balance between them. Because we were using an in-memory DB (h2) we needed to manually reduce the number of instances to 1 to ensure a consistent experience. Had we used an external experience to persist Todos we could easily use `n` number of instances to handle incoming requests.

Be warned — using `gcloud` will kill your network's performance. While using the `gcloud` CLI tool to deploy my applications I was not able to use my network access for anything else.