Using Grafana In A Web Application

**Introduction**

Grafana is primarily a way of visualizing time series data, i.e. a set of paired data (x, y) where x measured in time and y is measured in some other unit, such as meters, users or doodads. With Grafana, you create dashboards through the UI which can then in turn be accessed through calls to a Grafana server and embedded in web applications.

When running, Grafana sets up a server and exposes an API that allows you to make post and get requests to obtain the dashboards you create within the Grafana UI. You can also manually create dashboards through API calls. The API documentation can be found at:

[https://grafana.com/docs/reference/http\_api/#users](https://grafana.com/docs/reference/http_api/%23users)

There is also a library of pre-made dashboard provided by the Grafana community available at:

<https://grafana.com/grafana/dashboards>

These dashboards can be configured to most data sources and embedded in your web application as you please. In general, Grafana has excellent documentation available at:

<https://grafana.com/docs/>

that will walk you through the whole setup and configuration process. However, if you want the bare minimum to get Grafana up and running and put to work in your application, this document will hopefully provide that. By the end, you should have a local Grafana server serving up a custom dashboard that will display in a simple html component you can embed in a larger web application.

Note: The github for this frontend,

<https://github.ibm.com/grant-moore/grafana-demo>

is to be used in conjunction with the backend repo at,

<https://github.ibm.com/grant-moore/grafana-auth-server.git>

**Grafana Setup**

1. First things first. Download the Grafana binaries. Here are the links.

WINDOWS:

Download zip file containing binaries from:

<https://grafana.com/grafana/download?platform=windows>

LINUX:

Follow the instructions found on:

<https://grafana.com/grafana/download?platform=linux>

2. Assuming you’re on Windows, before extracting file, right click zip file and click *Properties*. Make sure the 'unblock' checkbox is marked. Then, extract the folder to where ever you want Grafana to run from.

3. Go to the /conf/ folder in the extracted folder. Copy and past the 'sample.ini' file and rename   
'custom.ini'. This file will configure you server.

All of the properties are commented out by default. You will need to uncomment the following to get your Grafana server up and running,

3A. In the Security heading,

*admin\_user = root*

*admin\_password = root*

3B. In the Server heading,

*http\_port = 8088*

We will edit a few other properties in this file a little later, so keep it handy.

**Setup Local Database**

4. We’ll need a data source for Grafana. Grafana is preconfigured to work with a large variety of databases. For the purposes of this tutorial, we will use a Postgres SQL Database. Download Postgres from the following link,

<https://www.postgresql.org/download/>

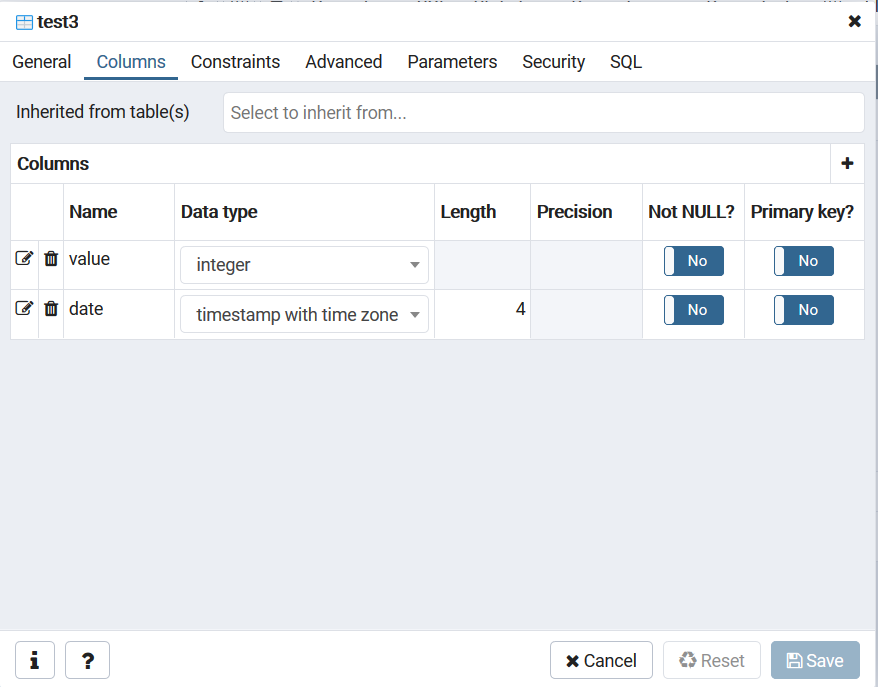
Create a database on your local computer. You can do so manually, but pgAdmin provides a UI and is much easier. pgAdmin is one of the options you can install along with the basic Postgres installation. If you already have Postgres, but you don't have a pgAdmin, download and install it from here,

<https://www.pgadmin.org/download/>

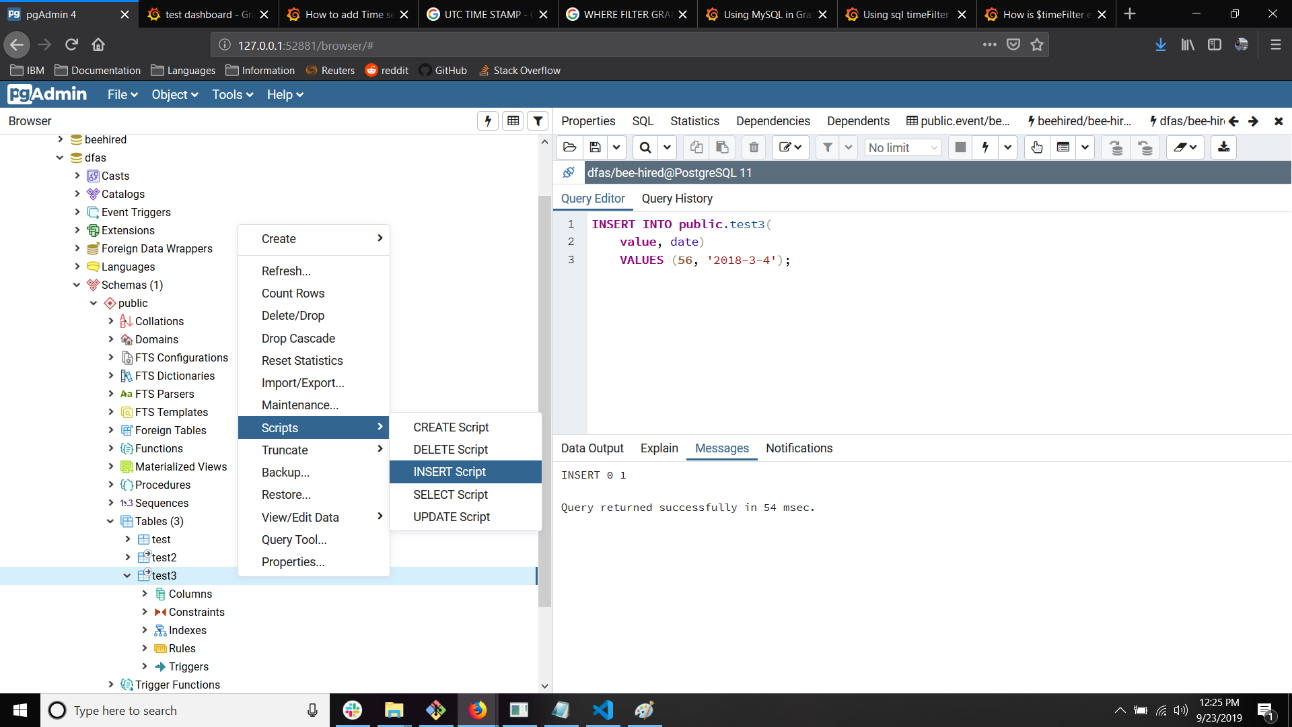
Once you have pgAdmin, right click your localhost server (create a server on your localhost if one isn't set up by default) and click *Create > Database*. Title the database whatever you want; I called mine 'grafana-demo'.

This database will act as the data source for Grafana, which we will configure in the next section. Postgres should set up on port 5432 by default. We will need to tell Grafana whatever port Postgres is running on, so take note.

5. Create a table with two columns. Name one of the columns '*value*' (otherwise you will need to specify the column to graph in Grafana; by default, Grafana looks for the '*value*' column, plus a column that contains time-formatted information, or a ‘timestamp’). Name the other column '*date*' or '*time*' or something like that. Make sure the columns have the data types seen in the following picture (or at the very least, the ‘*date*’ column should have the same data type; otherwise you will need to configure a macro to format the incoming time into something Grafana understands).



6. Insert some test data by right clicking the table and creating an INSERT script, shown below,

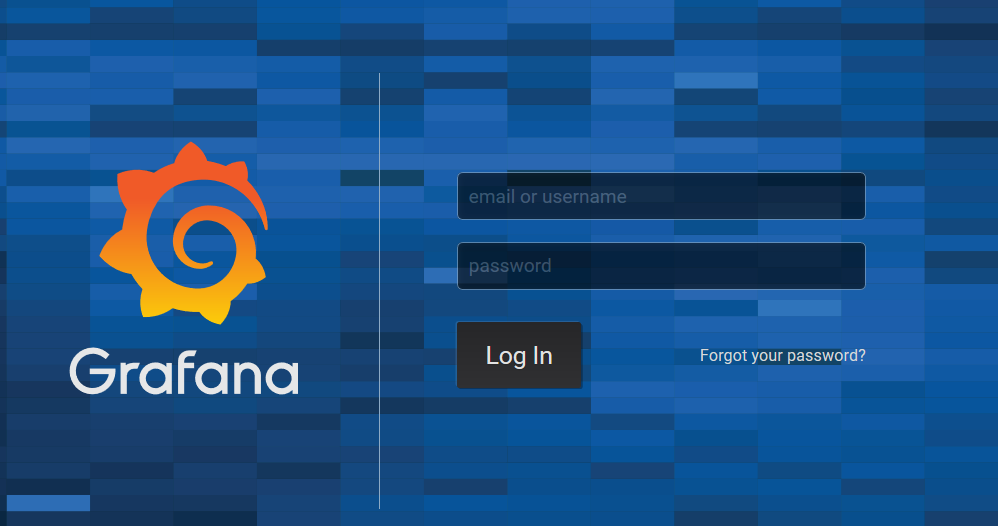


We will create a visualization for this data in Grafana after we connect the database in the following steps.

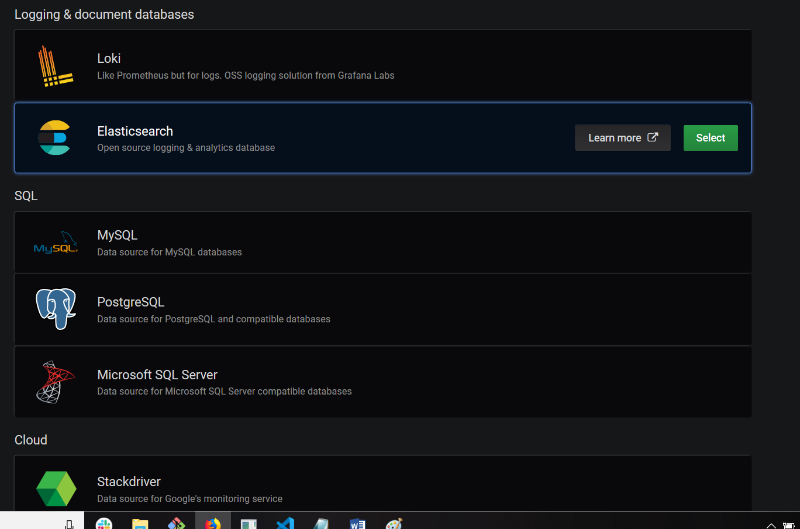
**Start Grafana Server and Create A Dashboard**

7. Go to the Grafana /bin/ folder in the extracted zip file and double-click the *'grafana-server.exe'* or navigate to the folder in a shell and call the executable. This will start the Grafana server, assuming there are no other processes running on the port you specified in Step 3B.

8. Once the Grafana server is up and running, go to <http://localhost:8088> (or whatever port you specified in Step 3B) in your browser. You will see the Grafana login page. Enter the *admin\_user* and *admin\_password* from Step 3 into the username and password input fields,

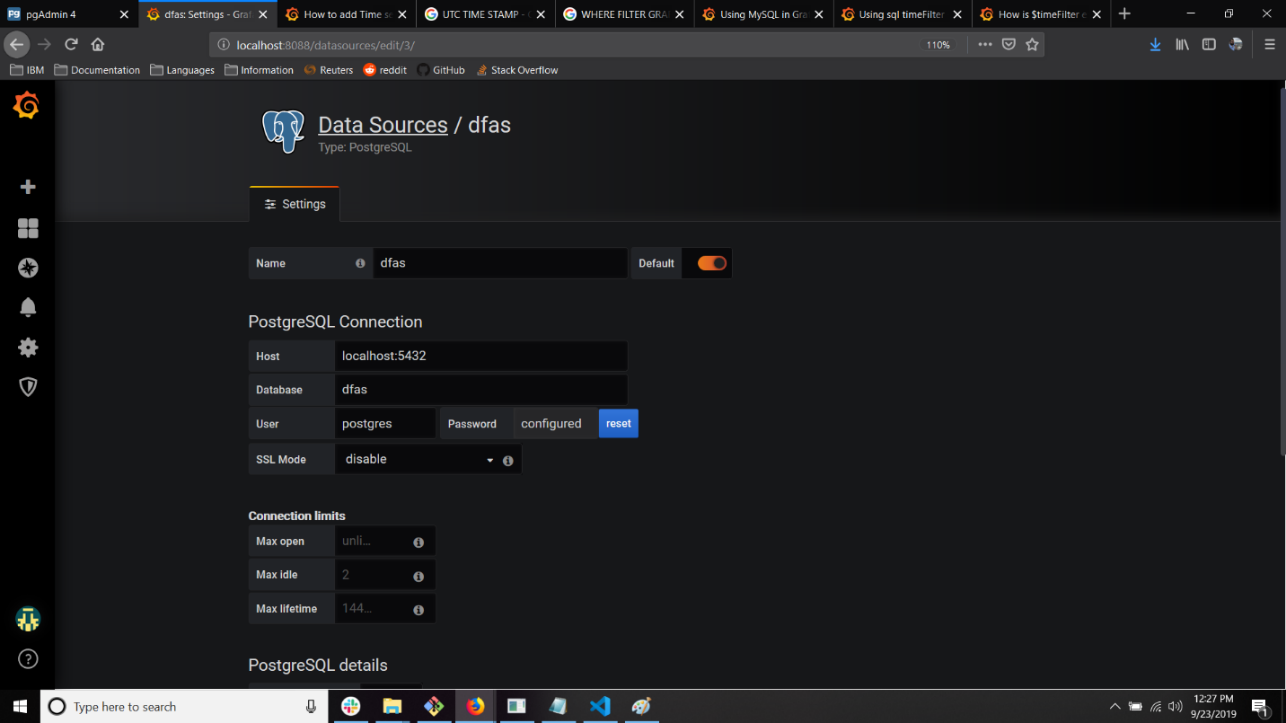


8. You will then be taken to the main menu. From the menu, click the Hamburger Icon and select ‘*Data Sources*’ from the Configuration submenu. Click ‘*Add Data Source*’. You will see a list of common data sources. There should be an entry for Postgres SQL,



If you don’t see a Postgres data source, try searching for it. If you still can’t find one, you may have to install a plugin. There is a link on the bottom of the page to search for available plugins.

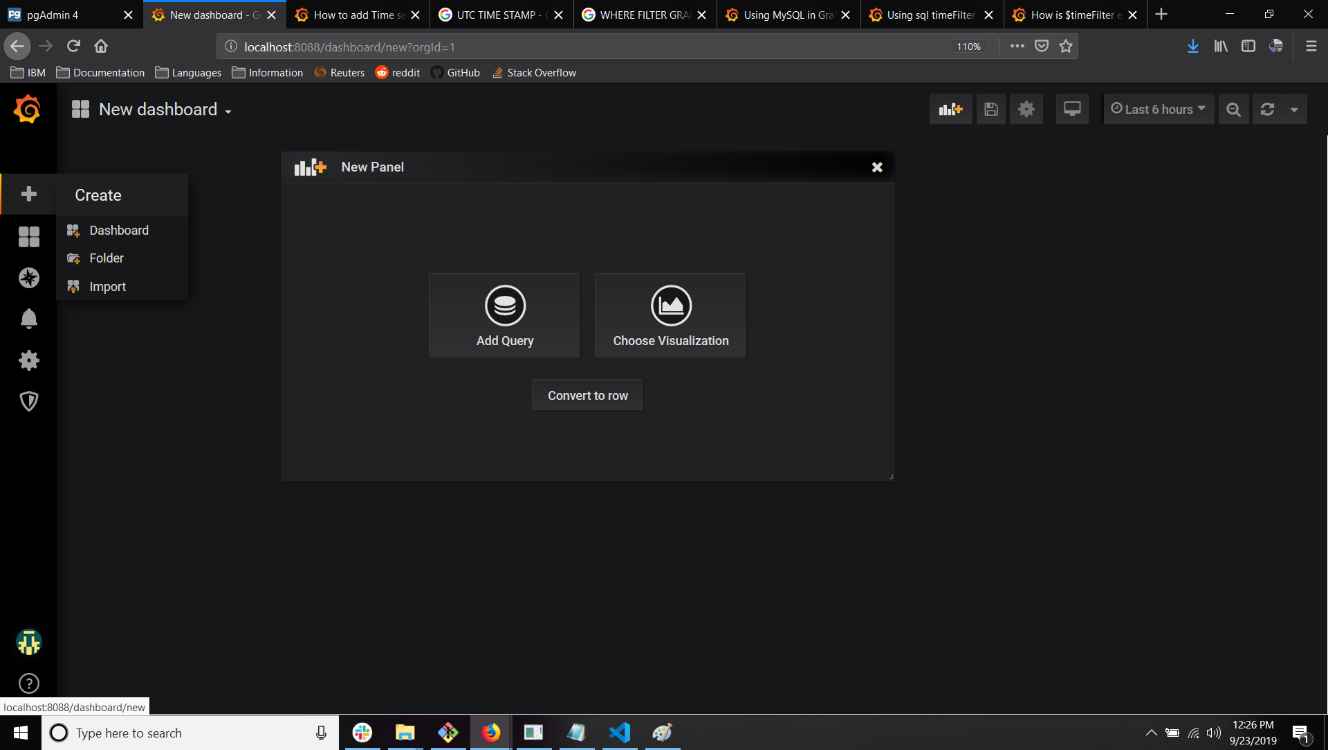
9. In the Postgres SQL Connection Menu, enter the information about the host, database name, username and password. Disable SSL. Here is what you should see,



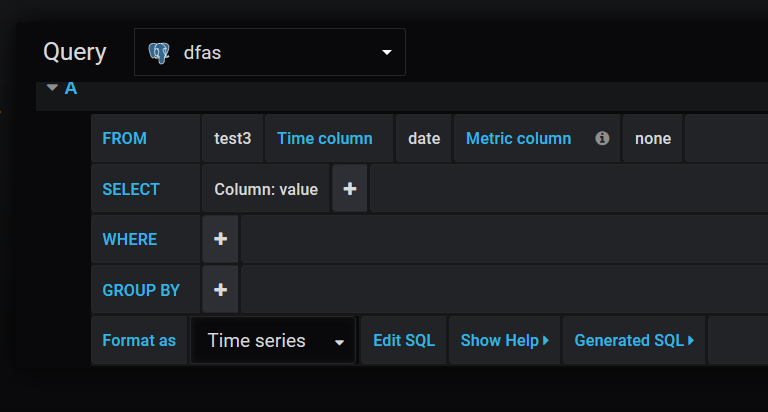
If you have everything configured properly, scroll to the bottom and click *'Save & Test'*. If everything goes well, you will see a green success message and the data source will now be available from the ‘*Data Sources’* menu.

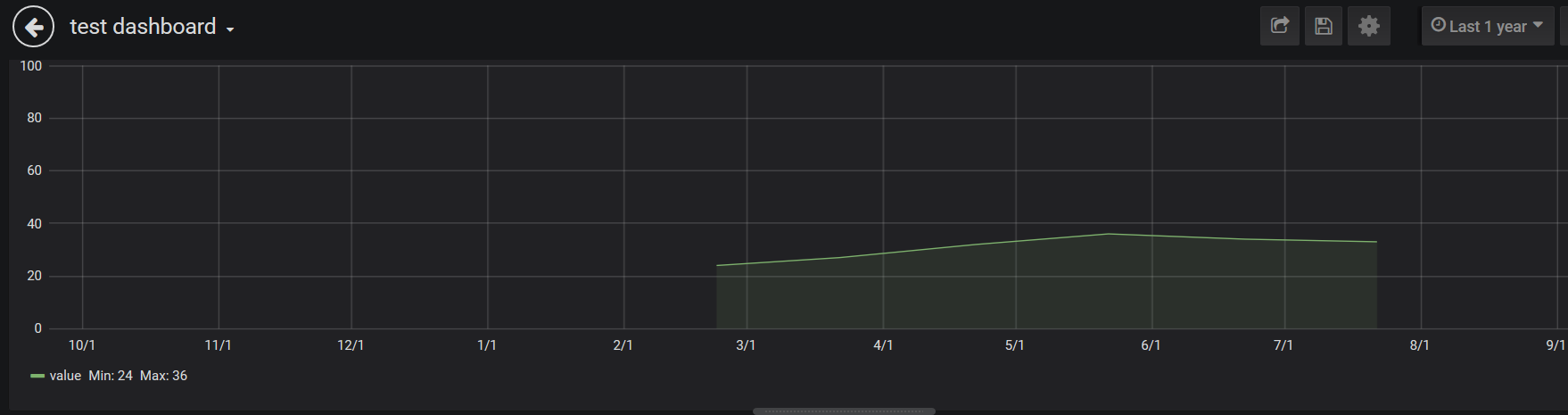
NOTE: The Host has the grayed out text localhost:5432 by default, but that isn't actually typed into the box. It can be a source of frustration wondering why it isn't connecting if you haven’t manually typed in the Host.

10. After the data source is added, you are ready to create a dashboard. Click the ‘*Add Dashboard’* button and ‘*Add A New Panel’*, shown below,



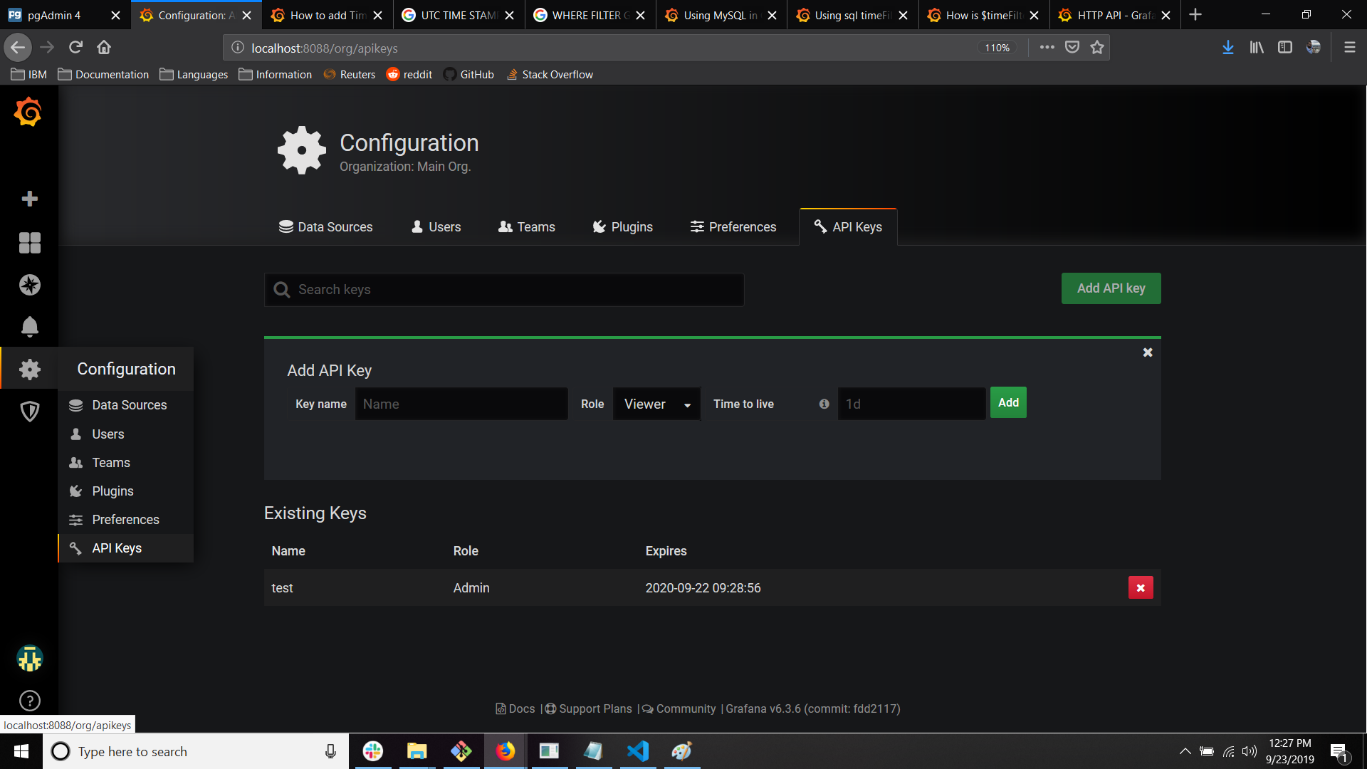
11. Click the Panel title and select ‘*Edit*’. This will bring up the query builder. As mentioned earlier, the query builder should automatically populate the table in the query by looking for tables with a ‘*value*’ column and a timestamped column. If no tables match this format, you can manually create a query using the macros provided by Grafana. For now, enter in the following query (changing your table and field names where appropriate) and the graph should automatically populate.





Glorious, isn’t it?

12. Once the Dashboard is created, save it. To access the Dashboard with a HTTP get request, you need to create an API key to append to all the requests so Grafana can authenticate you. The API Key can be created by clicking the ‘*API Keys’* item from the ‘*Configuration’* submenu,

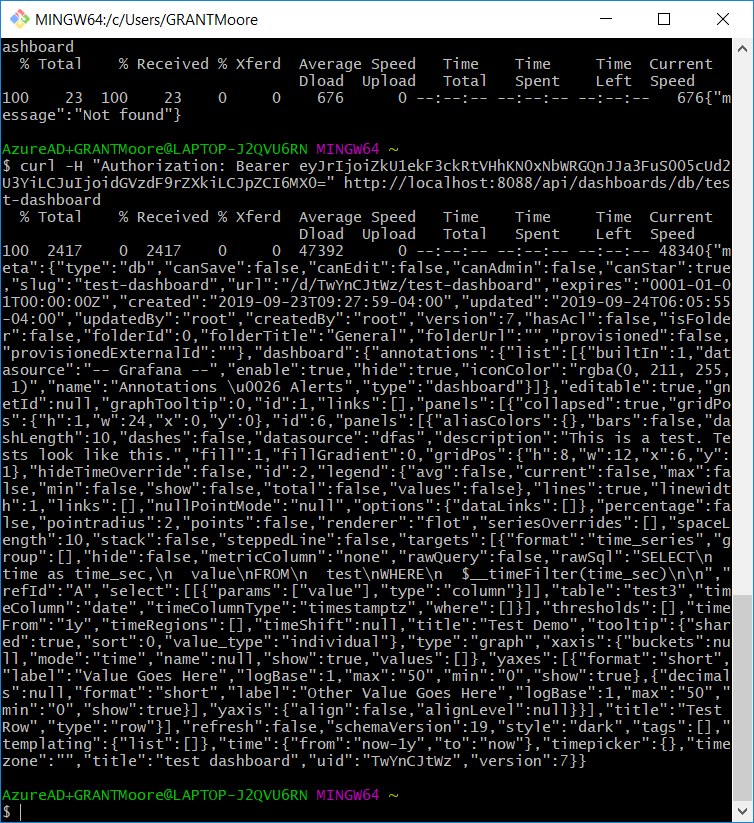


Enter the key name, the role (roles can be edited in the dashboard settings to allow whatever permissions you want), and the lifetime of the key. If you do not enter a lifetime, it will be valid forever. Once created, the dashboard can be accessed by the API exposed by the Grafana server if requests have the key appended to their ‘*Authorization*’ header. See

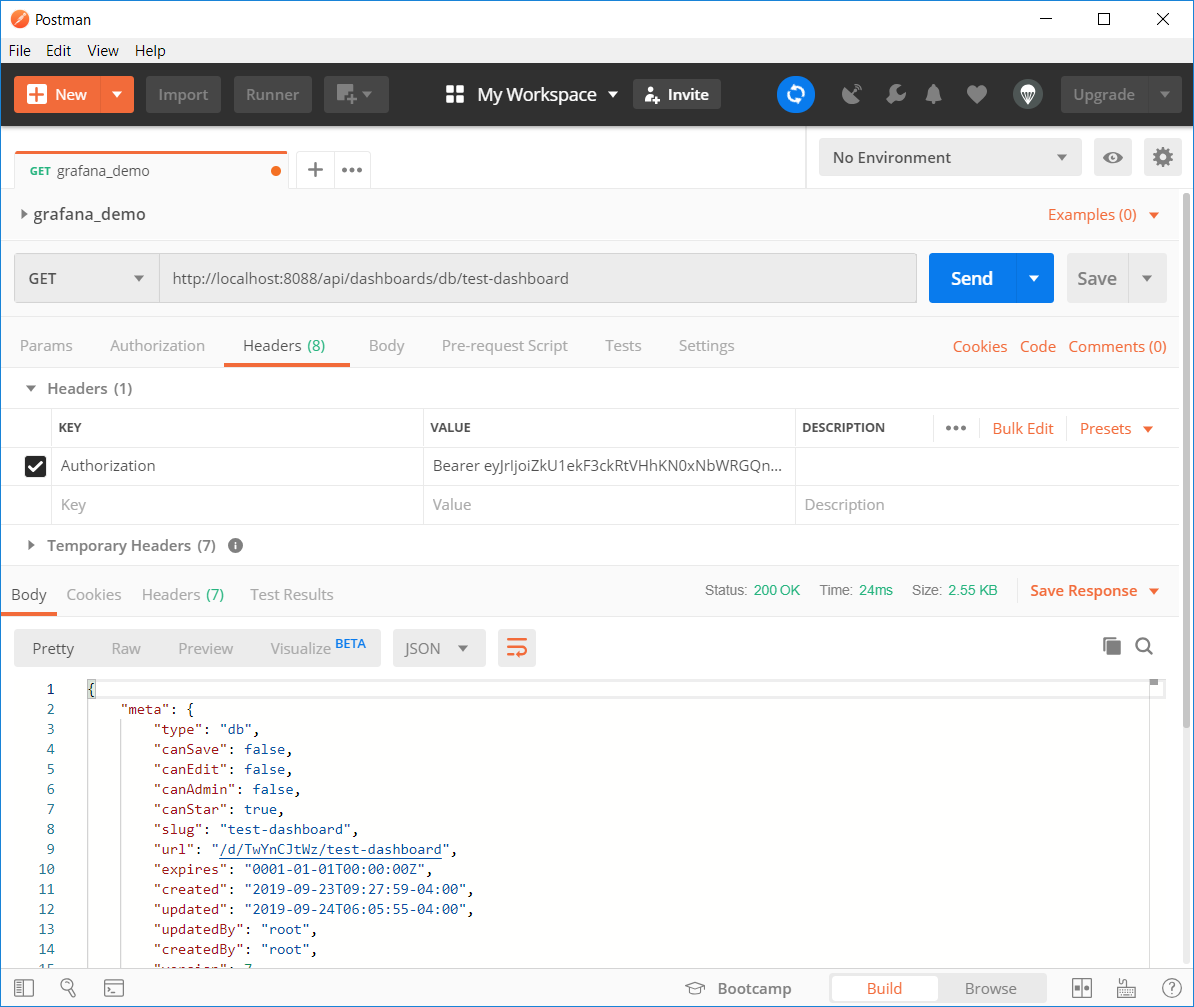
<https://grafana.com/docs/reference/http_api/>

for more details about the API. We will use the ‘GET’ documentation in the next section to create an html component that displays the dashboard we just created.

13. With the Grafana server running in the background, verify you can GET the dashboard using either **curl** in bash,



Or by using *Postman* to form a request with the API key in the ‘*Authorization’* header,



NOTE: The API key must be prefixed with the ‘*Bearer ‘* string! Refer to the Grafana documentation for more information: <https://grafana.com/docs/>

We now have a Grafana server capable of serving up the dashboard we created and authenticating the dashboard requests to verify they come from reliable sources. It would nice if everything were good to go at this point, but unfortunately as it stands, requests sent directly to Grafana will produce the infamous CORS error. There is no option to allow CORS requests in Grafana, so per the documentation, we will set up a reverse proxy to forward requests to Grafana.

**Reverse Proxy Setup**

14. The idea is to provide an iframe html element with the API endpoint from the Grafana server that renders the panel. In order to do this properly, Grafana’s documentation recommends setting up a proxy to forward requests with the API key appended to the ‘*Authorization*’ header. The setup that follows uses Node.js Express to create a proxy and the indicated server flow. The full code can be found on the author’s personal IBM Github at,

<https://github.ibm.com/grant-moore>

15. To create a server, *npm init* in a new folder, go through the Node package.json wizard and then install the following Node modules,

*npm install –save express  
npm install –save body-parser  
npm install –save cors  
npm install –save express-http-proxy*

16. Create an *app.js* file in the same folder and import the modules you just installed.

const express = require('express')

const path = require('path');

const bodyParser = require('body-parser')

const cors = require('cors')

const helper = require('./helper.js')

const proxy = require('express-http-proxy');

17. Make sure Express is instantiated and configured with the following middleware,

// EXPRESS SETUP

const app = express()

app.use(bodyParser.json())

// ENABLE CORS

app.use(cors())

The bodyParser middleware is *very* necessary. When you POST a query to Grafana (which happens implicitly by embedding a panel in an iframe), the POST body will be formatted with the JSON.

18. Finally, set up proxy middleware and be sure to intercept the request before it goes out and append the Authorization header with the API key in Step 12.

  // GRAFANA AUTHENTICATED PROXY

app.use('/proxy/grafana/getAuthPanel', proxy("http://localhost:8080/", {

proxyReqOptDecorator: function(proxyReqOpts, srcReq) {

      proxyReqOpts.headers['Authorization'] = `Bearer ${helper.grafanaApiKey()}`;

      return proxyReqOpts;

  }

}))

NOTE #1: *helper* is a simple javascript object that returns the API key, among other things.

NOTE #2: Don’t forget to *app.liste*n! See source code on Github for full server file!

19. The only thing left to do is a little configuration on Grafana’s end. Go back to the *custom.ini* file in the /conf/ folder from the Grafana installation. Find and update the following properties,

*root\_url* = <http://localhost:NODE_PORT/PROXY_PATH>  
*serve\_from\_sub\_path* = true

replacing ‘*NODE\_PORT’* with the port you set Node to listen on and ‘*PROXY\_PATH’* with the path you mapped the proxy middleware to (in the above case, it would ‘/*proxy/grafana/getAuthPanel*/’). These settings are required for the redirects that will be issued for the requests that come the iframe’s *src.*

**Embed Dashboard in Html With Iframe Element**

20. We are now ready to embed an iframe with the Grafana dashboard panel we created! Create a new html document and insert the following element.

<iframe src="http://localhost:8000/proxy/grafana/getAuthPanel/d-solo/8wjwzDhZk/dfas-demo?orgId=1&from=1507131768&to=1570021893064&panelId=2" width = 400 height = 400></iframe>

Serve up the HTML document in the Node server and navigate to it in your browser. You should see the Grafana panel embedded in an iframe! Style as you desire and embed in a large application to take full advantage of the capabilities offered by Grafana!