[2]p()*1/2-ˆ [1]p1-ˆ TJ
310000pt

Grafana Provisioning Documentation

José Moreno Hanshing

Feb 28, 2019

README

CHAPTER

ONE

SERVER CONFIGURATION

To be used on CentOS 7 with Grafana 5.4.2 or greater. Grafana is expected to be installed on `/etc/grafana`.

1.1 Configuration

The project will be stored and ran in a directory in the server. This is called the `main directory` and will be set on *installation*. All relative paths in this document are relative to the `main directory`.

1.1.1 General Configuration

The *config.yaml* file contains the following settings, which are all required:

- ***dashboardsDir***: Base directory where Grafana provisioned dashboards will be stored. The provisioning project will create subfolders and symlinks inside to organize Grafana organizations and their provisioned folders.
- ***provisioningDir***: *`main directory`, where the project's files reside.
- ***timeout***: Maximum time in seconds to wait for API requests when not receiving a reply. Will end the program execution if reached.
- ***updateIntervalSeconds***: How often Grafana will scan for changed dashboards. Once a folder is provisioned, if the need arises to change this it must be done so manually in a YAML file inside Grafana's installation directory.

* This functionality has been made optional because now the program detects its directory automatically. Since there was not enough time to test this thoroughly, the option to set it manually is still available. Only set this if you are seeing file not found errors pointing to the wrong directory.

1.1.2 Grafana Admin Accounts

This project provisions two Grafana Admin accounts. To set their account details like login name and password, you need to configure them in `admins/_superAdmins.yaml`.

`grafanaAdmin` refers to the configuration of Grafana's Main admin account, which is the default account that is created when you install Grafana (id=1).

`api` refers to the account that will be used by this script to make API calls to the Grafana server and provision organizations, accounts, datasources, folders and dashboards.

1.1.3 Default Organization

The project treats the first organization (id=1) as a special case. You can configure its name as well as the default accounts that will be part of it in `admins/_kiosk.yaml`.

To add dashboards to this organization, you will need to add an input directory in `inputs/` *just like with any other organization*, the *org.yaml* file must contain a hash with a key as the name of the org, but the list of other accounts that can be in it *can* be blank.

In the same way, the accounts that this organization provisions are set in `admins/_kioskAccounts.yaml`. If you create input for this organization, you still need have

1.2 Installation

Note: *Some steps require previous steps to have been completed before being executed. The syntax* `rn` *on one step means that this step requires step* `n` *to have been completed,* `r1` *on step 2 means that step 2 requires step 1 to complete. Similarly,* `r6, r8` *on step 9 means that step 9 requires both step 6* **and** *step 8 to complete.*

For the following examples, the `main directory` will be */etc/grafana/lsst*.

1.2.1 Dependencies

1. Install EPEL repository:

```
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.
↪rpm -y
```

2. *r1* Install Python 3.6 with the requests and pyyaml libraries:

```
yum install python36 python36-requests.noarch -y
python36 -m ensurepip
pip3.6 install --upgrade pip
pip3.6 install pyyaml
```

3. Install Grafana plugins, this requires Grafana v5.3.0 or newer. You need to restart Grafana for new plugins to be recognized:

```
grafana-cli plugins install yesoreyeram-boomtable-panel
```

1.2.2 Provisioning Setup

4. Place the project in a secure directory:

```
mv GrafanaProvisioning /etc/grafana/lsst
```

5. *r4* Configure the directories the project will use. See *Configuration* for more details:

```
vim /etc/grafana/lsst/config.yaml
```

5.1. Configure the directory where the project resides in as *provisioningDir* inside *config.yaml* which is in the `main directory`:

```
provisioningDir: /etc/grafana/lsst
```

5.2. Set *dashboardsDir* to another secure directory, where Grafana will look for dashboards:

```
dashboardsDir: /var/lib/grafana/dashboards
```

1.2.3 Provisioning

If any of the scripts close because of an error, do not continue with the following steps.

6. *r2*, *r5* Run *gpSetup.py*:

```
python36 /etc/grafana/lsst/gpSetup.py
```

7. *r6* Configure firewall rules to use Grafana (open port 3000).
8. *r4* For each organization/use case/department for which you wish to provision Grafana, add a directory with the correct *Input Structure* inside the *inputs/* directory.
9. *r6*, *r8* Run *gpInputs.py*:

```
python36 /etc/grafana/lsst/gpInputs.py
```

10. *r9* Run *gpAccounts.py*:

```
python36 /etc/grafana/lsst/gpAccounts.py
```

11. *r3*, *r9* Restart `grafana-server`. See *Criteria to Restart Grafana* to understand when Grafana should be restarted:

```
systemctl restart grafana-server.service
```

1.3 Criteria to Restart Grafana

Grafana does not need to be restarted on every Puppet execution. You only need to restart grafana-server if any of the following cases has happened since the last Puppet execution:

- If a Grafana plugin has been installed.
- If one of the `datasources.yaml` files inside any organization's input folder has changed.
- If the contents of the first level inside one of the `dashboards/` directories in any organization's input has changed.

The last two are hard to track with Puppet, so instead a file called `restart.txt` is created, which contains the string "restart". If Puppet finds this file it can schedule grafana-server for restarting and delete the file.

1.4 FAQ

1.4.1 Can I make permanent links to my dashboards?

While Grafana uses and allows a unique id (uid) for their dashboards making it easy to link them over the web, we do not allow persistent UIDs on provisioned dashboards. This is to avoid conflicting UIDs, since if two dashboards are provisioned with the same uid, the second one will overwrite the first one and remove the latter from the database. It is possible to write a script that manages a local file with taken UIDs corresponding to specific files, but this has not been implemented in this version of this project.

Grafana will generates random UIDs for dashboards which do not have them on startup. Sometimes when Grafana is *restarted*, the uid of provisioned dashboards will change. This means that you *can* have direct links to your dashboards, but it is not guaranteed that they will work in the long term.

As per Grafana's documentation:

> "**Note.** Provisioning allows you to overwrite existing dashboards which leads to problems if you re-use settings that are supposed to be unique. Be careful not to re-use the same `title` multiple times within a folder or `uid` within the same installation as this will cause weird behaviors."

1.4.2 What software and versions does this project use?

- CentOS 7
- Grafana v5.4.2 (commit: d812109)
- Boom Table Plugin (Grafana) v0.4.6
- EPEL 7-11
- Python 3.6.6-1.el7
- python36-requests 2.12.5-2.el7
- pyyaml 3.13

1.5 Recommendations

1.5.1 Grafana Settings

Grafana's default settings can be found on:

```
/usr/share/grafana/conf/defaults.ini
```

They need to be changed directly through Puppet, as it is the one which manages Grafana's deployment.

1.5.1.1 DNS

Sometimes clicking a link in Grafana will redirect users to `localhost`. This likely due to the misconfigured DNS setting in Grafana. It is intended to be set to what the address in the browser will use as a DNS. Reference:

```
[server]
   domain: 10.0.0.252:3000
```

1.5.1.2 Viewers Can Edit

It is worth considering activating this option. When set to `true`, viewers are allowed to edit/inspect dashboard settings in the browser, but not to save the dashboard. This would allow users to easily create and export new dashboards. Reference:

```
[users]
   viewers_can_edit: true
```

1.5.2 Grafana Internal Database

Grafana uses an internal database to manage users, dashboards, organizations and other data. By default this database is SQLite, which has limitations which include concurrent queries. If the system ever grows to a scale where the platform becomes unstable or unreliable, you may want to consider switching this database to MySQL or PostgreSQL.

1.5.3 Time Related Issues

There are a few settings that influence how you see data in Grafana depending on the frequency that the data is collected, and how it is saved and queried.

1.5.3.1 Datasource Minimum Time Interval

When configuring your datasources, you should set the minimum time interval to the frequency that you are collecting data. If you set an smaller interval, you will see empty graphs when you zoom in too much in Grafana and are using dynamic time intervals.

When provisioning, Grafana accepts setting `timeInterval` for Prometheus, Elasticsearch, InfluxDB, MySQL, PostgreSQL and MSSQL datasources:

```
jsonData:
   timeInterval: 1m
```

1.5.3.2 InfluxDB Row Limit

For this database, a setting can be configured to limit the size of a query's response through HTTP. This can cause confusing behavior in Grafana when zooming out too much. InfluxDB is set to respond with a maximum of 10,000 rows to prevent server overload:

```
[http]
   max-row-limit = 10000
```

This means that in a graph, Grafana will plot up to 10,000 points regardless of how many lines are drawn. When the query exceeds this limit Grafana will just display the incomplete graph, but if you open the query inspector you will find `partial:  true` in the metadata inside certain objects.

To avoid this you can group by larger time intervals in your query, which will produce a smaller amount of results. Additionally, you can create a variable with different time intervals and use it in your query like:

```
GROUP BY time($var)
```

so that you can change it when you zoom out.

Related Jira ticket

1.5.3.3 Telegraf Precision

When the system clock shifts over time and is readjusted by chronyd, Telegraf's internal clock is unaffected. This results in Telegraf saving measurements to with shifted timestamps. Additionally, when Grafana queries InfluxDB for these shifted measurements between time ranges and groups them by time, there is a significant chance that some results will not be returned because the measurement will correspond to a time after the specified time range.

To avoid this, you can set the precision in Telegraf to the interval that you are gathering data, so that timestamps are automatically rounded to the nearest interval. This is not perfect, but it can alleviate the problem:

```
[agent]
   precision = "1m"
```

Related Jira ticket

Bug report

CHAPTER

TWO

PROVISIONING GRAFANA ORGANIZATIONS

2.1 Introduction

2.1.1 Grafana Basic Concepts

This is a summary of what you need to know to understand this page. For more details, you can read Grafana's Basic Concepts.

Grafana works with internal `organizations` which function as user groups in which users can add, view and generally manage their dashboards. Users have their own `account` each, which have permissions depending on what that user will use that account for. Accounts can be members of an organization in which they can have three roles: *Admin*, *Editor* and *Viewer*; they can also be members of more than one organization although it is usually not the case.

`Dashboards` are pages where you can add `panels`, which are types of graphs, tables and other utilities. All dashboards must belong to an organization. `Datasources` are Grafana's way of naming and configuring how it connects to different databases available, they must be declared on each organization and dashboards must gather data from datasources declared in the same organization which they belong to.

2.1.2 What Provisioning Your Organization Means

When you provision something to Grafana, you must provide access to your `Input Files` in an online repository. The server will download your files every time it detects there are changes and will then update your Grafana Organization.

Once your files are provisioned, your own Grafana organization will be created where your dashboards will be displayed categorized in as many Grafana Folders as you choose, your datasources will be added, and the accounts you specify will belong to it. If the accounts are new they will be created for you, and if they already exist they will just be added to the organization.

The advantage of having all of this done automatically is that we can easily recover from losing a server or other mishaps. User mistakes can be avoided:

- If someone deletes a dashboard or folder on accident, it will be automatically restored.
- Datasources cannot be edited directly through the web client, they must be edited in the provisioning configuration.
- Dashboards can only be temporarily edited by editors. Users can't save over provisioned dashboards, but they can save new ones if their permissions allow them to.

You can add new dashboards, datasources and users to your organization. But you can rest knowing that the basic configuration will always be there.

2.2 Input Structure

The script which processes the input files is *gpInputs.py*, it is meant to run on every Puppet execution. It will process files provided by users which should be stored in folders inside the *inputs/* directory. These folders represent Grafana organizations and should come with three files and a directory inside them:

```
My Org's Repo/
├── org.yaml
├── accounts.yaml
├── datasources.yaml
└── dashboards/
    ├── Your Grafana Folder/
    │   ├── DashboardsGoHere.json
    │   └── My Dashboard.json
    └── YourOtherGrafanaFolder/
        └── DashboardsGoHereToo.json
```

The specifications of each file are explained below. You should also `download the example files` and modify them to fit your own requirements. After you are done, upload them to an online repository and make them available to the server admin so that they can add your organization to the provisioning configuration. You can later add new folders, dashboards and users to the files in your repository and the server will update them automatically.

2.2.1 Organization

The *org.yaml* file should contain a hash with only one key, which should be the organization's name. As the value of that key, there should be a list of hashes, each with two keys:

- **login** [*str*] Username of a provisioned account inside an *accounts.yaml* file. This account could be configured inside a different organization.
- **role** [{`Admin`, `Editor`, `Viewer`}] The role that the user will have inside this organization

Organization names must be unique in each Grafana server. Check with a server admin if your name for a new organization is available.

2.2.2 Accounts

The *accounts.yaml* file should contain new accounts that need to be created and maintained with the provisioning of this organization. The structure is a list of hashes, each with four string keys, some of which must be unique inside this Grafana installation:

- login : Username of the new account (`unique`).
- password : Password of the new account.
- name : Name of the user.
- email : Email associated with the new account (`unique`).

2.2.3 Datasources

The *datasources.yaml* file should contain a list of datasources to configure according to what Grafana requires for different database engines to provision them, minus the fields `orgId` and `editable` since they will be added by this script. If the file comes with settings for `deleteDatasources` this field will not be added to the provisioned file.

Note that if you have more than one datasource in your organization, it is important that only one of them has the `isDefault: true` setting.

2.2.4 Folders and Dashboards

The *dashboards/* directory must contain one directory for each Grafana Folder that will need to be provisioned for this organization. The Grafana Folders will take the name of these directories and will contain the dashboards stored inside them. You must place your JSON files with Grafana Dashboards in these directories.

The dashboards must be already configured to use their corresponding datasources and with the correct time range because they will be displayed exactly as they are provided.

2.2.4.1 Exporting Dashboards

You can create your own dashboards in Grafana and then export them to your repository. To export a dashboard:

1. Open your dashboard in Grafana.
2. Depending on the Grafana version there are two ways to export dashboards: if you can't see the checkbox on the first one, you should use the second one.

- The simple way:
  a. Click the *Share dashboard* button.
  b. Go to the *Export* tab.
  c. **Uncheck** *Export for sharing externally*.
  d. Save to file.
- The long way. This method can be used for any Grafana version:
  a. Click on *Settings*.
  b. Go to the *View JSON* or *JSON Model* tab at the bottom.
  c. Click inside the JSON code.
  d. Press `ctrl + a` to select all of the JSON text.
  e. Copy the entire JSON.
  f. Open a text editor like *Notepad*.
  g. Paste the JSON into the editor.
  h. Save the file as a `.json`. The file name is not important but should be descriptive. For example `temperature.json`.

2.3 Program Behavior with Input Files

2.3.1 What the Input Files Do

2.3.1.1 org.yaml

- Create a new organization with the configured name if it doesn't exist.
- Add configured users to organization, with given roles.

- Report an error if the organization name is already configured in another set of input files.

2.3.1.2 accounts.yaml

- Create configured accounts that don't exist.
- Report an error if a configured account is already provisioned by another organization.

2.3.1.3 datasources.yaml

- Add the configured datasources to the organization.
- Remove previously provisioned datasources that were removed from the configuration.

2.3.1.4 dashboards/

- Detects folders inside it and provisions a Grafana Folder with the contained dashboards for each one.
- Restore provisioned folders and dashboards that have been deleted.

2.3.2 What the Input Files Can't Do

2.3.2.1 org.yaml

- Change the name of an organization. If you change it the server will just create a new organization with the provisioning configuration and stop provisioning the old one.
- Add accounts to an organization, which are not part of any organization's provisioning files.

2.3.2.2 accounts.yaml

- Edit an existing account

2.3.2.3 dashboards/

- Delete Grafana Folders and dashboards that have been removed from the configuration. You must remove these Folders or dashboards manually through the web client with an *Editor* or *Admin* account to delete them from Grafana's database.
- Provision dashboards with given UIDs. IDs and UIDs will be removed before provisioning and Grafana will assign new ones.

CHAPTER

THREE

YAML UTILITY MODULE

Module to read from and write to our yaml configuration files.

This module is intended to be used by the Grafana provisioning scripts to read information from certain yaml configuration files, and write to others. The functions getYamlContent and writeYamlContent are generic enough so that they can be used with any yaml file. It is implemented with pyyaml 3.13.

3.1 Functions

`yamlUtility.`**`getApiCredentials`**`()`

Return Grafana's main admin's username and password.

From the information stored in /admins/_superAdmins.yaml, return the main admin's `login` and `password`. This user should usually have id=1.

Returns

admlogin [*str*] Username, or `login`, of the main Grafana Admin account.

admPasswd [*str*] `password` of the main Grafana Admin account.

Raises

yaml.YAMLError Raised if the super admins file does not contain a valid YAML format.

PermissionError: Raised if the module does not have read permissions on the super admins file.

FileNotFoundError: Raised if the super admins file doesn't exist or if the file cannot be accessed by the module.

See also:

`loadConfig`, *`getYamlContent`*

`yamlUtility.`**`getSuperAdminLogin`**`()`

Obtain the username of Grafana's Main admin account.

Returns

login [*str*] Username of Grafana's Main admin account.

Raises

yaml.YAMLError Raised if the super admins file does not contain a valid YAML format.

PermissionError: Raised if the module does not have read permissions on the super admins file.

FileNotFoundError: Raised if the super admins file doesn't exist or if the file cannot be accessed by the module.

See also:

loadConfig, *getYamlContent*

`yamlUtility.`**`getYamlContent`**`(`*file*`)`

Return YAML file translated to a Python data structure.

Parameters

file [*str*] Path of the YAML file whose content will be loaded.

Returns

yamlConfig [Usually *dict* or *list*] Contents of the YAML configuration file translated to Python, can be any type of value or structure supported by YAML.

Raises

yaml.YAMLError Raised if the given file does not contain a valid YAML format.

PermissionError: Raised if the module does not have read permissions on the given file.

FileNotFoundError: Raised if the given file doesn't exist or if the file cannot be accessed by the module.

Notes

Since YAML files are data structures mainly composed of dictionaries and lists, the caller of this function must know what data structure to expect as a return value for a given file.

`yamlUtility.`**`loadConfig`**`()`

Return config.yaml as a Python data structure and set global timeout.

This function loads the content of config.yaml into Python. These are settings for the Grafana provisioning scripts to use. It also sets the global value of `to`, which is a timeout configuration used by `req()`.

Returns

config [*dict*] The contents of config.yaml as a Python dictionary. Contains routes to directories used by the provisioning scripts, and other settings such as timeout limit for API requets.

Raises

yaml.YAMLError Raised if the config file does not contain a valid YAML format.

PermissionError: Raised if the module does not have read permissions on the config file.

FileNotFoundError: Raised if the config file doesn't exist or if the file cannot be accessed by the module.

See also:

getYamlContent

`yamlUtility.`**`writeYamlContent`**`(`*file*, *data*`)`

Write contents of a Python data structure to a YAML file.

Parameters

file [*str*] Path of the YAML to which *data* will be written.

data [Usually *dict* or *list*] Python data structure to be written in *file*, can be any type of value or structure supported by YAML.

Raises

PermissionError: Raised if the module does not have write permissions on the given file or directory.

CHAPTER

FOUR

GRAFANA API MODULE

Module to use as interface with Grafana's API.

This module is intended to be used by the Grafana Provisioning scripts to communicate with Grafana's API through HTTP requests. If you wish to use HTTPS all you need to do is add an *s* to the string returned by the _apiUrl function, note that this is not tested though.

4.1 Notes

The default timeout for API requests is 5 seconds. This can be changed through the global variable *timeout*.

4.2 Functions

exception `grafanaAPI.`**`APIError`**(*user*, *password*, *response=None*)

Represents that an error occurred when executing an API request.

This exception should be raised when an API request returns with a status code in the 4XX or 5XX range. This mirrors the behavior of the default requests.HTTPError, except that it doesn't print the user and password contained in the url constituting the request (within the API, Grafana's admin can only authenticate using the default authentication method for all of the admin only functions, although this might change in the future).

When raised with an HTTP `response`, the exception prints out the status code returned, the method used for the request (e.g. POST), the url to which the request was made (without user and password) and the content of the response made by Grafana.

Used together, these four items can give enough information to understand what request was being made and what went wrong with it.

Parameters

- **user** [*str*] `login` of the user who was making the API request. To be erased from the url.
- **password** [*str*] `password` of the user who was making the API request. To be erased from the url.
- **response** [*requests.Response*, optional] Response to the API request containing error information. If ignored, the exception will not print an error message.

`grafanaAPI.`**`createAccount`**(*accountData*, *user*, *password*)

Create a Grafana user account.

Parameters

accountData [*dict*] Dictionary with the data that Grafana can receive for account creation with the API. Grafana 5.4.2 supports the following fields: - `login`: Username of the new user (*str*). - `password`: Password of the new user (*str*). - `name`: Name of the new user (*str*). - `email`: Email of the new user (*str*).

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

userId [*int*] Grafana `id` of the newly created user.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: The user already exists, email already in use, invalid password or email, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_req`, *`removeFromOrg`*

`grafanaAPI.`**`createGrafanaAdmin`**(*accountData*, *user*, *password*)

Create an account with Grafana Admin privileges.

Parameters

accountData [*dict*] Dictionary with the data that Grafana can receive for account creation with the API. Grafana 5.4.2 supports the following fields: - `login`: Username of the new user (*str*). - `password`: Password of the new user (*str*). - `name`: Name of the new user (*str*). - `email`: Email of the new user (*str*).

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

userId [*int*] Grafana `id` of the newly created user.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: The user already exists, email already in use, invalid password or email, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`createAccount`, `_req`

`grafanaAPI.`**`createOrg`**(*orgName*, *mainAdmin*, *user*, *password*)

Create a new Grafana organization.

Parameters

orgName [*str*] Name of the Grafana organization to be created.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

orgId [*int*] Number corresponding to the new organization's `id` in Grafana.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: an organization with that name already exists, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_req`, *`setUserRoleOrg`*, *`getExistingUserId`*, *`removeFromOrg`*

`grafanaAPI.`**`getExistingUserId`**`(`*login*, *user*, *password*`)`

Get the Grafana `id` of a user that already exists

Parameters

login [*str*] Grafana (`login`) of the user for whom the `id` is being retrieved.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

id [*int*] Grafana `id` of the existing user.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: user doesn't exist, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_req`

Notes

If the user doesn't exist and there is another user with an email that is equal to this user's login name, that user will be retrieved instead.

`grafanaAPI.`**`getOrgId`**`(`*orgName*, *user*, *password*`)`

Obtain the `id` of a Grafana organization using its name.

Parameters

orgName [*str*] Name of the Grafana organization for which the `id` will be retrieved.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

orgId [*int*] Number corresponding to the organization's `id` in Grafana.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: an organization with that name doesn't exist, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_req`

`grafanaAPI.`**`removeFromOrg`**(*orgId*, *userId*, *user*, *password*)

Remove a user from a Grafana organization.

Parameters

orgId [*int*] Number corresponding to the organization's `id` in Grafana.

userId [*int*] `id` of the user that is being removed from the Grafana organization.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

r [*requests.Response*] Response object containig the data returned by Grafana, including JSON data as a string which can be converted to a dictionary with r.json(), and a status code.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_req`

Notes

This function is intended to be used when creating new accounts to remove them from the default organization, and when creating new organizations to remove the API account from them.

`grafanaAPI.`**`request`**(*method*, *api*, *user*, *password*, *jsn=None*)

Make an api request with a string HTTP method, using basic authentication.

Wrapper for internal _req function. This function receives an HTTP request method as a string and passes it onto _req as a function.

Parameters

method [{'get', 'post', 'put', 'delete', 'patch'} (*str*)] An HTTP request method as a string.

api [*str*] Suffix of the url. This should be the variable part of the API path that is required in the url, i.e. what comes after `http://[...]/api/`.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

jsn [*dict*] Contains the metadata that will be passed in JSON format with the API request. This should be data that Grafana is prepared to receive. Optional.

Returns

response [*requests.Response*] Response object containig the data returned by Grafana, including JSON data as a string which can be converted to a dictionary with r.json(), and a status code.

Raises

ValueError Raised if the method passed does not correspond to one of the accepted parameters.

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: incomplete or wrong data passed with *jsn*, the *api* url or the *method* is wrong, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_apiUrl`, `_req`

Notes

This function makes the script importing this module free to call a request with an HTTP method as string instead of as a library function. This way we separate implementation details from functionality.

The *methods* dictionary is defined globally so that it does not need to be created every time a request is made. It only contains the HTTP methods currently used by the scripts.

`grafanaAPI.`**`setUserRoleOrg`**`(`*orgId*, *userId*, *login*, *newRole*, *user*, *password*`)`

Add a user to an org, with a role. Or set the role for a user in an org.

Parameters

orgId [*int*] Number corresponding to the organization's `id` in Grafana.

userId [*int*] `id` of the user who is being assigned a role on the Grafana org.

login [*str*] Grafana username (`login`) of the user who is being assingned a role.

newRole [{'Admin', 'Editor', 'Viewer'}] The role to be assigned to the user inside the organization.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Raises

APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: invalid *newRole*, inexistant organization or user, wrong *login* regarding the given *userId*, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`_req`

Notes

In the future, if needed login can be turned into an optional parameter and it can be fetched inside this function using *userId*. Currently the function assumes that the *login* corresponds to the correct *userId* in Grafana. Passing a *login* of a different user might produce unexpected behavior.

CHAPTER

FIVE

SETUP SCRIPT

Configures Grafana Admin accounts and default org for a fresh installation.

This script is meant to run it's main body only on a fresh installation of Grafana. Still, it is designed to be able to be ran with every Puppet execution with the help of the file `lastInitialization.txt` which will be stored in the same directory where this is ran from. When this script completes successfully, it will write the timestamp of when it initialized Grafana.

This script will load configurations stored in `admins/_kiosk.yaml` and `admins/_superAdmins.yaml`. It changes the `password` and account name (`login`) for the default account (which has `super admin` privileges). It will then create another account, which will be used by the script from now on to make requests to Grafana's API. With that account, the name of the default organization will be changed.

5.1 Notes

Since Grafana's API can be accessed over the network through the same port by which the web client is accessed, it is `IMPORTANT` that this script is ran before opening Grafana's port (default is 3000) on the firewall.

5.2 Functions

`gpSetup.`**`changeAdminPassword`**`(`*newPasswd*, *user*, *password*`)`

Change Grafana Admin's password.

Parameters

newPasswd [*str*] New password for the Admin account

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

See also:

`grafanaAPI`, *`renameAdminUser`*

Notes

It can be acceptable for the request to fail, if the execution of this script was interrupted before and it is still necessary to complete the final steps.

When testing, changing the username and password did not kick the logged in user from the website.

`gpSetup.`**`renameAdminUser`**(*data*, *user*, *password*)

Change Grafana admin's username.

Parameters

data: 'dict'

Contains the fields accepted by Grafana API's `User Update`:

- login: Change the username of the account (*str*).
- name: Change the name of the account (*str*).
- email: Change the email of the account (*str*).
- theme: Change the theme of the account {`light`, `dark`}.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

See also:

`grafanaAPI`, *`changeAdminPassword`*

Notes

It can be acceptable for the request to fail, if the execution of this script was interrupted before and it is still necessary to complete the final steps.

When testing, changing the username and password did not kick the logged in user from the website.

`gpSetup.`**`renameKioskOrg`**(*user*, *password*)

Rename the default organization (id=1) to what is in `admins/_kiosk.yaml`.

Parameters

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Raises

FileNotFoundError Raised if `admins/_kiosk.yaml` can't be found by this script. The file must contain the name of the default organization and a list of users which will be registered to it.

ValueError Raised if 0 or more than 1 organizations exist in `admins/_kiosk.yaml`. The number of organizations in this file must be exactly 1.

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: The first organization does not exist, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`grafanaAPI.request`

Notes

Affects the current context organization for the user.

CHAPTER

SIX

INPUT SCRIPT

Creates files/symlinks to provision orgs, users, datasources and dashboards.

6.1 Input Files

This script is meant to run it's main body on every Puppet execution. It will process files provided by users which should be stored in folders inside the `inputs` directory. These folders represent Grafana organizations and should come with three files and a directory inside them:

1. `org.yaml`
2. `accounts.yaml`
3. `datasources.yaml`
4. `dashboards/`

The specifications for these files are described on *Input Structure*.

6.1.1 Organization

The script will create a symlink to this file in the `orgs/` directory.

6.1.2 Accounts

The script will create a symlink to this file in the `accounts/` directory.

6.1.3 Datasources

The script will create the final version of this file inside */etc/grafana/provisioning/datasources* which **will be read by grafana-server when it starts**.

6.1.4 Dashboards

The script will create a directory in the configured *dashboardsDir* where a directory for each folder will be created, inside them the JSON files will be copied, without the ID and UID of their dashboards. A YAML file will also be created inside */etc/grafana/provisioning/dashboards/* which will contain the list of directories corresponding to this organization. The YAML file with the folder structure **will be read by grafana-server when it starts**, the dashboards inside them will be checked for changes inside the provided folders with a time period configured by *updateIntervalSeconds*.

6.2 Notes

Provisioning configurations are stored in `config.yaml`.

6.3 Functions

`gpInputs.`**`copyDashboardWithoutIds`**(*source*, *dest*)

Load source JSON file, delete ID and UID, and save the dashboard at dest.

Parameters

source [*str*] Path to input JSON file containing a dashboard.

dest [*str*] Path to output JSON file, the dashboard to be provisioned at dashboardsDir.

Raises

FileNotFoundError: Raised if *source* does not exist or it cannot be accessed by the script.

PermissionError: Raised if the script does not have permission to read from *source* or to write to *dest*.

JSONDecodeError: Raised if *source* does not contain a valid JSON format.

`gpInputs.`**`getDirList`**(*top*)

Get list of first level directories at *top*, excluding hidden folders.

Parameters

top [*str*] Path of the directory which will be scanned.

Returns

dirs [*list* of *str*] List containing the directories found in the first level of *top* excluding the ones that start with a period.

See also:

`os.walk`

`gpInputs.`**`needsProvisioning`**(*inputFile*, *currentFile*)

Check if source is newer than dest.

Parameters

inputFile [*str*] File in the input folder.

currentFile [*str*] File created by this script for provisioning Grafana.

Returns

inputIsNewer [*bool*] True if input file is newer than current provisioning file, else False.

`gpInputs.`**`provisionDashboards`**(*orgName*, *grafanaFolders*, *orgInputDir*, *dashboardsDir*)

Maintain the correct dashboards inside the folders in dashboardsDir.

Copy dashboards to dashboardsDir without ID and UID when they need to be copied. Delete dashboards from dashboardsDir which do not exist in the input.

Parameters

orgName [*str*] Name of the org in Grafana to which the dashboards will be provisioned.

grafanaFolders [*list* of *str*] List containing the names of the folders that are going to be provisioned. Just the folder names, not full paths.

orgInputDir [*str*] The directory where the inputs for the given org are stored.

dashboardsDir [*str*] The directory where Grafana will look for provisioned dashboards.

Raises

FileNotFoundError: Raised if one or more of the parameters conforms a path that doesn't exist.

PermissionError: Raised if the script does not have permission to read dashboards from the input, or to write to dashboards in the dashboardsDir.

JSONDecodeError: Raised if an input dashboard does not contain a valid JSON format.

`gpInputs.`**`provisionDatasources`**`(`*orgId*, *orgName*, *dSrcYaml*`)`

Create datasources config, which is read by grafana-server when it starts.

Loads the configuration provided in `org.yaml`, adds the corresponding orgId and sets `editable = false` to each datasource. Also removes the `deleteDatasources` field.

Parameters

orgId [*int*] ID of the org in Grafana to which the datasources will be provisioned.

orgName [*str*] Name of the org in Grafana to which the datasources will be provisioned.

dSrcYaml [*dict*] Contains the data required by Grafana to provision datasources. The two required keys are:

- apiVersion: Version of the input `datasources.yaml` file (*int*).
- **datasources: List of datasources, with all the necessary configurations** except for *orgId* and `editable`, which are added here.

Raises

PermissionError: Raised if the script does not have write permissions to `/etc/grafana/provisioning/datasources/```orgName```_datasources.yaml`

See also:

`yamlUtility.writeYamlContent`

`gpInputs.`**`provisionFolders`**`(`*orgId*, *orgName*, *grafanaFolders*, *orgInputDir*, *dashboardsDir*`)`

Create folder structure in dashboardsDir and configure each folder route.

Uses `dashboardRoutesTemplate.yaml` as a template to create the folder routes.

Parameters

orgId [*int*] ID of the org in Grafana to which the dashboards will be provisioned.

orgName [*str*] Name of the org in Grafana to which the dashboards will be provisioned.

grafanaFolders [*list* of *str*] List containing the names of the folders that are going to be provisioned. Just the folder names, not full paths.

orgInputDir [*str*] The directory where the inputs for the given org are stored.

dashboardsDir [*str*] The directory where Grafana will look for provisioned dashboards.

Raises

yaml.YAMLError Raised if the template contains an invalid YAML format.

PermissionError: Raised if the script does not have read permissions on the template, or if it doesn't have write permissions to the configured `dashboardsDir` or `/etc/grafana/provisioning/dashboards/`

FileNotFoundError: Raised if the template doesn't exist or it can't be accessed by the script.

See also:

`yamlUtility.getYamlContent`, *`yamlUtility.writeYamlContent`*, `os.makedirs`, `os.symlink`

Notes

The folder configuration is read by grafana-server when it starts. The dashboards in the folders will be checked periodically for updates by Grafana.

`gpInputs.`**`provisionOrg`**(*orgInputDir*, *user*, *password*, *provisionedOrgs*)

Makes sure that the organization in Grafana is provisioned.

If the organization already exists, get the org's id from Grafana. Else, create the org in Grafana and the symlink to `org.yaml` inside `orgs/`. Returns the org's id and name.

Parameters

orgInputDir [*str*] The directory where the inputs for this org are stored.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

provisionedOrgs [*dict*] Dictionary containing all orgs in the provisioning configuration. Consists of one key per org, where the key is the org's name and the value True for all provisioned organizations. We use a dictionary instead of a list because it should be faster for searching if an org is provisioned.

Returns

orgId [*int*] `id` of the organization inside Grafana.

orgName: 'str' Name of the Grafana organization.

Raises

ValueError Raised if there is more than one organization with the same name in the YAML configuration files or if `org.yaml` doesn't contain exactly one organization. We do not support this feature. Different organizations should come separately in different inputs.

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: the symlink in `orgs/` exists but the organization in Grafana doesn't (in this case delete the symlink manually), the org already exists in Grafana and the symlink doesn't, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`grafanaAPI.getOrgId`, *`grafanaAPI.createOrg`*

Notes

The existance of the symlink should represent the state of the Grafana org: if the symlink doesn't exist then the org shouldn't exist either, and if the symlink exists the org should as well. If the creation of the org is interrupted halfway through it will need to be deleted manually (or else we could be deleting already existing orgs).

CHAPTER

SEVEN

ACCOUNTS SCRIPT

Provisions non-default orgs and non-admin accounts in Grafana using the API.

This script is meant to run it's main body on every Puppet execution.

It will make sure that there are no duplicate organizations or accounts in the YAML configuration files, and create them in Grafana in case they do not exist yet. Each account will be created with its given username, password, name and email. Each organization will be created with their given name, and all accounts in their list will be added to that organization with their given role. Note that accounts can belong to more than one organization, but must only be declared once in the YAML files.

This script loads the basic info of all accounts and organizations into memory, and makes many requests to the API, so it can be a bit slow. Further testing is necessary to determine if, how and where it can be optimized.

7.1 Functions

`gpAccounts.`**`createAndReviewOrgs`**`(`*provOrgs*, *existingUsers*, *user*, *password*`)`

Loop through non existing orgs, create them and provision their accounts.

Parameters

provOrgs [*dict*] Dictionary containing all orgs in the provisioning configuration that have not been created yet. Consists of one key per org, where the key is the org's name and the value is a list with the accounts to be provisioned for the organization.

existingUsers [*dict*] Dictionary containing all the Grafana users in the provisioning configuration. Consists of one key per user, where the key is the user's username and the value is the user's ID in Grafana.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Raises

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: wrong parameters, invalid *role*, inexistant user, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`reviewExistingOrgs`, *`reviewOrgUsers`*, *`grafanaAPI.createOrg`*

Notes

Relies on deleting the existing orgs from provOrgs

`gpAccounts.`**`getOrCreateUser`**`(`*account*, *user*, *password*`)`

Return a user's ID. If the account doesn't exist, create it.

Query for accounts containing the login name inside their login, email or name. The default limit is 1000 results per page, it is not expected that a query will match more than 1000 results. This is done instead of getting a single user because this doesn't return a 404 code when the user isn't found.

Parameters

account [*dict*] Dictionary with the data that Grafana can receive for account creation with the API. This is the same data which is found on an org's `accounts.yaml` file. Grafana 5.4.2 supports the following fields: - `login`: Username of the new user (*str*). - `password`: Password of the new user (*str*). - `name`: Name of the new user (*str*). - `email`: Email of the new user (*str*).

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

userId [*int*] Grafana `id` of the user. Either of an existing account or a new one.

Raises

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: Trying to create an account with an email already in use, invalid password or email, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`grafanaAPI.request`, *`grafanaAPI.createAccount`*

`gpAccounts.`**`loadProvisionedOrgs`**`(`*orgsDir*`)`

Load orgs from YAML config into dict indexed by name, storing user lists.

Parameters

orgsDir [*str*] Path to the directory where the orgs YAML files (symlinks) are stored.

Returns

provOrgs [*dict*] Dictionary containing all orgs in the provisioning configuration. Consists of one key per org, where the key is the org's name and the value is a list with the accounts to be provisioned for the organization.

Raises

ValueError Raised if there is more than one organization with the same name in the YAML configuration files or if there isn't exactly one org in a given `org.yaml` file.

yaml.YAMLError Raised if an org file does not contain a valid YAML format.

PermissionError: Raised if the script does not have read permissions on an org file.

See also:

`yamlUtility.getYamlContent`

`gpAccounts.`**`loadProvisionedUsers`**`(`*accountsDir*, *user*, *password*`)`

Load users from YAML config, create them if the don't exist, get their IDs.

Parameters

accountsDir [*str*] Path to the directory where the accounts YAML files (symlinks) are stored.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Returns

existingUsers [*dict*] Dictionary containing all the Grafana users in the provisioning configuration. Consists of one key per user, where the key is the user's username and the value is the user's ID in Grafana.

Raises

ValueError Raised if there is more than one user with the same username in the YAML configuration files. This could happen if two orgs are trying to provision the same user, in this case only one of them should provision the user in the `accounts.yaml` file, but both of them should have the user in their `org.yaml` file.

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: email already in use, invalid password or email, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

yaml.YAMLError Raised if an accounts file does not contain a valid YAML format.

PermissionError: Raised if the script does not have read permissions on an accounts file.

See also:

`getOrCreateUser`, *`yamlUtility.getYamlContent`*

`gpAccounts.`**`reviewExistingOrgs`**`(`*grafOrgs*, *provOrgs*, *existingUsers*, *user*, *password*`)`

Loop through all existing orgs and make sure their accounts are provisioned.

Loop through all of Grafana's existing orgs and make sure that the ones which belong to the provisioning have their provisioned accounts associated to them, with the correct role. Remove the org from the *provOrgs* dictionary after finishing to be able to create and review the remaining ones later.

Parameters

grafOrgs [*list* of *dict*] List returned by Grafana containing all of its existing organizations.

provOrgs [*dict*] Dictionary containing all orgs in the provisioning configuration. Consists of one key per org, where the key is the org's name and the value is a list with the accounts to be provisioned for the organization.

existingUsers [*dict*] Dictionary containing all the Grafana users in the provisioning configuration. Consists of one key per user, where the key is the user's username and the value is the user's ID in Grafana.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Raises

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: wrong parameters, invalid *role*, inexistant user, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`reviewOrgUsers`, `grafanaAPI.setUserRoleOrg`

`gpAccounts.`**`reviewOrgUsers`**(*orgId*, *provOrgUserList*, *existingUsers*, *user*, *password*)

Make sure each user for the given org belongs to it with the correct role.

For the given org, go through each user in the list and make sure they are in the org and have the correct role. Assumes the users exists.

Parameters

orgId [*int*] ID of the organization to review users for.

provOrgUserList [*list* of *dict* of *str*] List of the users for each org. This is equivalent to the structure found in the `org.yaml` input files. Each dict has two keys: - login: Username of the account belonging to the org (*str*). - role: Role that the account has in this organization (*str*).

existingUsers [*dict*] Dictionary containing all the Grafana users in the provisioning configuration. Consists of one key per user, where the key is the user's username and the value is the user's ID in Grafana.

user [*str*] `login` of the Grafana account that is making the API request.

password [*str*] `password` of the Grafana account that is making the API request.

Raises

grafanaAPI.APIError Raised if the request replies with a status code in the 4XX or 5XX range. The causes include: invalid *role*, inexistant organization or user, wrong *login* regarding the given *userId*, invalid credentials (*user* and *password*), the user doesn't have permission to make this request or the server is not responding. Check the error messages for more information.

See also:

`grafanaAPI.setUserRoleOrg`

Notes

If an account is configured in an `org.yaml` file but it is not found in any `accounts.yaml` file, a warning message will be printed and the program will continue normally.

If an account is found more than once in an `org.yaml` file, only the first time it appears on the file will be considered for provisioning and a warning message will be printed. The program will continue normally.

CHAPTER EIGHT

SITE MAP

- genindex
- modindex

PYTHON MODULE INDEX

INDEX