Apache Kibble Documentation

Release 0.1

The Apache Kibble Community

Jan 16, 2021

Contents:

CHAPTER 1

Setting up Apache Kibble

1.1 Understanding the Components

Kibble consists of three major components:

- **web application - this is the user facing part of Apache Kibble. Via this** ui users can create organizations, configure scanners and most importantly view and analyze the data.
- **scanners - as the name suggest are application designed to work** with a specific type of resource (a git repo, a mailing list, a JIRA instance etc) and push compiled data objects to the Kibble Server. Some resources only have one scanner plugin, while others may have multiple plugins capable of dealing with specific aspects of a resource.
- **database - an instance of ElasticSearch used by both web application and** scanners to share the information.

The following diagram shows Kibble architecture:

1.2 Component Requirements

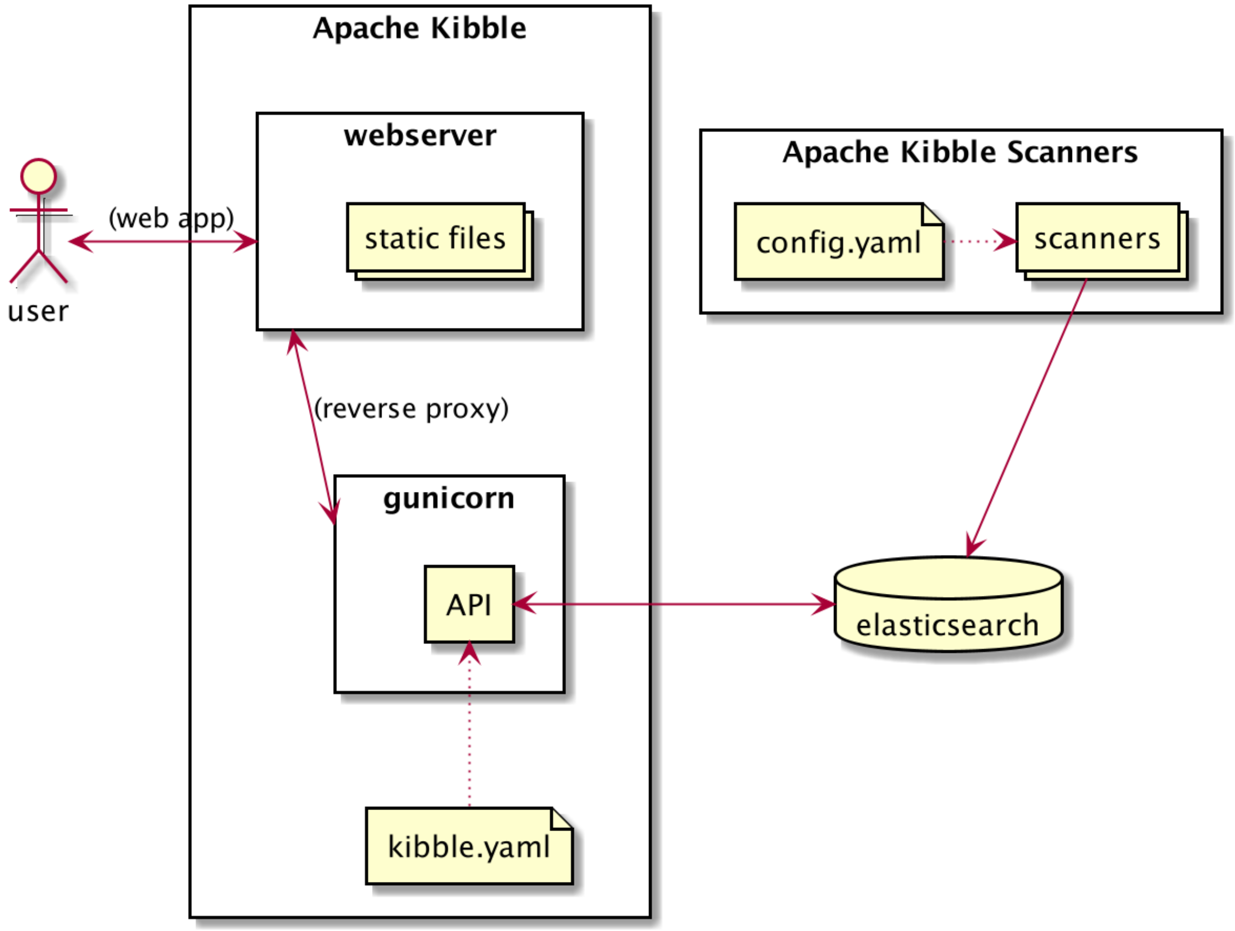
1.2.1 Server Component

The Kibble Server is a hub for scanners, and as such, is only ever needed on one machine. It is recommended that, for large instances of kibble, you place the application on a machine or VM with sufficient resources to handle the database load and memory requirements.

As a rule of thumb, the Server does not require a lot of disk space (enough to hold the compiled database), but it does require CPU and RAM. The scanners require more disk space, but can operate with limited CPU and RAM.

As an example, let us examine the Apache Kibble demo instance:

- 100 sources (git repos, mailing lists, bug trackers and so on)
- 3,5 million source objects currently (commits, emails, tickets etc)


Apache Kibble
webserver
static files
user
(web app)
(reverse proxy)
gunicorn
API
kibble.yaml
Apache Kibble Scanners
config.yaml
scanners
elasticsearch

- 10 concurrent users (actual people uing the web UI)

The recommended minimal specs for the Server component on an instance of this size would be approximately 4-8GB RAM, 4 cores and at least 10GB disk space. As this is a centralized component, you will want to spec this to be able to efficiently deal with the entire database in memory for best performance.

1.2.2 Scanner Component

The scanner components can either consist of one instance, or be spread out in a clustered setup. Thus, the requirements can be spread out on multiple machines or VMs. Scanners will auto-adjust the scanning speed to match the number of CPU cores available to it; a scanner with two cores available will run two simultaneous jobs, whereas a scanner with eight cores will run eight simultaneous jobs to speed up processing. A scanner will typically require somewhere between 512 and 1GB of memory, and thus can safely run on a VM with 2GB memory (or less).

1.3 Source Code Location

Apache Kibble does not currently have any releases. You are however welcome to try out the development version.

For the time being, we recommend that you use the `main` branch for testing Kibble. All source code can be found in our repository at: https://github.com/apache/kibble

1.4 Installing Kibble

1.4.1 Pre-requisites

Before you install the Kibble, please ensure you have the following components installed and set up:

- Python 3.8
- git binaries (GPL License)
- cloc version 1.76 or later (GPL License)
- An ElasticSearch instance, version 6.x or newer (5.x is supported for existing databases, but not for new setups). Does not have to be on the same machine, but it may help speed up processing.
- A web server of your choice (Apache HTTP Server, NGINX, lighttp etc)

1.4.2 Configuring and Priming the Kibble Instance

Once you have the components installed and Kibble downloaded, you will need to prime the ElasticSearch instance and create a configuration file.

To install `kibble` do the following

```
git clone https://github.com/apache/kibble.git
cd kibble
pip install .
```

As a good practice it is recommended to use virtual environment for installation.

Once `kibble` is installed you may wish to adjust the `kibble.ini` configuration file, especially the `elasticsearch` section which is required to connect to database.

Then you can run the following command to configure the database and create initial administrator account for the UI:

```
kibble setup --autoadmin --skiponexist
```

1.4.3 Setting up the Web UI

Once you have finished the initial setup, you will need to enable the web UI. Kibble is built as a WSGI application, and as such you can use mod_wsgi for apache, or proxy to Gunicorn. In this example, we will be using the Apache HTTP Server and proxy to Gunicorn:

- Make sure you have mod_proxy and mod_proxy_http loaded (on debian/ubuntu, you would run: *a2enmod proxy_http*)
- Set up a virtual host in Apache:

```
<VirtualHost *:80>
   # Set this to your domain, or add kibble.localhost to /etc/hosts
   ServerName kibble.localhost
   DocumentRoot /var/www/kibble/ui/
   # Proxy to gunicorn for /api/ below:
   ProxyPass /api/ http://localhost:8000/api/
</VirtualHost>
```

- Launch gunicorn as a daemon on port 8000 (if your distro calls gunicorn for Python3 *gunicorn3*, make sure you use that instead):

```
cd /var/www/kibble/api/
gunicorn -w 10 -b 127.0.0.1:8000 -t 120 -D kibble.api.handler:application
```

Once httpd is (re)started, you should be able to browse to your new Kibble instance.

1.4.4 Configuring a Scanners

Scanners are configured via `kibble.ini` configuration file.

Remember that the scanner must have enough disk space to fully store any resources you may be scanning. If you are scanning a large git repository, the scanner should have sufficient disk space to store it locally.

If you plan to make use of the optional text analysis features of Kibble, you should also configure the API service you will be using (Watson/Azure/picoAPI etc).

1.4.5 Balancing Load Across Machines

If you wish to spread out the analysis load over several machines/VMs, you can do so by specifying a `scanner.balance` on each node. The balance directive uses the syntax X/Y, where Y is the total number of nodes in your scanner cluster, and X is the ID of the current scanner. Thus, if you have decided to use four machines for scanning, the first would have a balance of 1/4, the next would be 2/4, then 3/4 and finally 4/4 on the last machine. This will balance the load and storage requirements evenly across all machines.

1.4.6 Running a Scan

Once you have both scanners and the data server set up, you can begin scanning resources for data. Please refer to *Configuring Data Sources* for how to set up various resources for scanning via the Web UI.

Scans can be initiated manually, but you may want to set up a cron job to handle daily scans of resources. To start a scan on a scanner machine, run the following:

```
kibble scan
```

This will load all plugins and use them in a sensible order on each resource that matches the appropriate type. The collected data will be pushed to the main data server and be available for visualizations instantly.

It may be worth your while to run the scanner inside a timer wrapper, as such: `time kibble scan` in order to gauge the amount of time a scan will take, and adjusting your cron jobs to match this.

1.5 Docker Image

If you want to spin up a development instance of Apache Kibble you can do:

```
docker-compose -f docker-compose-dev.yaml run kibble setup --autoadmin --skiponexist
docker-compose -f docker-compose-dev.yaml up ui
```

The ui should be available under `http://0.0.0.0:8000` or `http://localhost:8000`. To log in you can use the dummy admin account `admin@kibble` and password `kibbleAdmin`.

You can also start only the API server:

```
docker-compose -f docker-compose-dev.yaml up api
```

To trigger scanners run:

```
docker-compose -f docker-compose-dev.yaml run kibble scan
```

CHAPTER 2

Managing Apache Kibble

2.1 Creating an Organisation

The first thing you will need to set up, in order to use Kibble, is an organisation that will contain the projects you wish to survey. You can have multiple organisations in Kibble, and all organisations will be scanned, but the UI will only display statistics for the current (default) organisation you are using. You may switch between organisations at your leisure in the UI.

To create your first organisation:

1. Go to the “Organisation” tab in the top menu
2. Locate the Create a new organisation‘ field set
3. Enter the details required for the new organisation

This will set up a new organisation and set it as your default (current) one.

Once an organisation has been created, you can then add resources and users to it.

2.2 Configuring Data Sources

After you have created an organisation, you can add sources to it. A source is a destination to scan; it can be a git repository, a JIRA instance, a mailing list and so on. To start adding sources, click on the *Sources* tab in the left hand menu on the *Organisation* page.

With all resource types, you can speed up things by adding multiple sources in one go by simply adding one source per line in the source text field.

The currently supported resource types are:

GitHub This resource consists of GitHub repositories as well as issues/PRs that are contained within. Currently, you will need to add the full URL to the repo, including the *.git* part of it, such as: `https://github.com/apache/clerezza.git`. **NOTE**: If you intend to use more than 60 API calls per hour, which you probably do, you will need to add the credentials of a GitHub user to the source, in order to get a higher rate limit of 6,000 API calls per hour. You may use any anonymous account for this.

Git This is a plain git repository (such as those served by the standard git daemon), and only scans repositories, not PRs/Issues. If basic auth is required, fill our the user/pass credentials, otherwise leave it blank.

PiperMail This is the standard MailMan 2.x list service. The URL should be the full path to the directory that shows the various months

Pony Mail This is a Pony Mail list. It should be in the form of *list.html?foo@bar.baz* and you *should* include a session cookie in order to bypass email address anonymization where applicable. If the Pony Mail instance does not apply anonymization, you may leave the cookie blank.

Gerrit This is a gerrit code review resource, and will scan for tickets, authors etc.

BugZilla This is a BugZilla ticket instance. You should add one source for each BugZilla project you wish to scan. It should point to the JSONRPC CGI file followed by the project you wish to scan. If you wish to just add everything as one source, you can do so by pointing it at `jsonrpc.cgi *` which will scan everything in the BugZilla database. If you want to be able to look at individual projects, it's recommended that you scan them individually.

JIRA This is a JIRA project. Most JIRA instances will require the login credentials of an anonymous account in order to perform API calls.

Twitter This is a Twitter account. Currently not much done there. WIP.

Jenkins CI This is a Jenkins CI instance. One URL is required, and all sources will be scanned.

Buildbot CI This is a Buildbot instance. One URL is required, and all sources will be scanned in one go.

Once you have added the resource URLs you wish to analyse, you can obtain data by following the instructions in the chapter *Running a Scan*.

2.3 Adding New Users

MORE TODO

CHAPTER 3

Indices and tables

- genindex
- modindex
- search