kibitzr Documentation

Release 7.0.0

Peter Demin

Feb 25, 2023

Contents

1 Documentation Contents **3**

1.1 Installation . . . 3
1.2 Configuration . . . 4
1.3 Credentials . . . 7
1.4 Usage . . . 7
1.5 Browser automation . . . 8
1.6 Schedule . . . 10
1.7 Transforms . . . 12
1.8 Notifiers . . . 14
1.9 Python support . . . 15
1.10 Shell support . . . 17
1.11 Stash . . . 19
1.12 Extensions . . . 20
1.13 Recipes . . . 21
1.14 Contributing . . . 23
1.15 Contributors . . . 25

Python Module Index **27**

Index **29**

Kibitzr periodically runs checks described in `kibitzr.yml` file. Each check has following steps:

1. Fetch content;
2. Pass it through sequence of *Transforms*;
3. Run set of *Notifiers* with transformed content.

CHAPTER 1

Documentation Contents

1.1 Installation

1.1.1 Stable release

To install kibitzr, run these commands in your terminal:

```
$ virtualenv venv
$ source venv/bin/activate
$ pip install kibitzr
```

This is the preferred method to install kibitzr, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.1.2 Dependencies

Kibitzr has many integrations and depending on what features are used may require additional setup.

The recommended way to have all dependencies installed and configured is to use Docker.

1.1.3 Docker

Make sure Docker is installed.

Run the following commands to pull docker image, create example configuration and run kibitzr:

```
mkdir kzr-example
cd kzr-example
docker run -v $PWD:/root --rm peterdemin/kibitzr init
docker run -v $PWD:/root --rm peterdemin/kibitzr run
```

1.1.4 Manual installation

The hard way is to install all dependencies. Consult Dockerfile and gcp tutorial on required steps.

Kibitzr uses several Python packages, that have C extensions. When installed through pip, they are compiling libraries. This process requires gcc (which is almost always present) and Python header files (which are not installed on vanilla Linux).

You can either install those dependencies using OS installer:

```
apt install python-lazy-object-proxy python-yaml
```

or install Python headers:

```
apt install python-dev
```

1.1.5 Optional dependencies

Some of the dependencies are used only when corresponding features are used in `kibitzr.yml`.

1. `changes` transform. Requires git.
2. **`delay` and `scenario` - triggers for using Firefox as a fetcher.** Installing Firefox can be cumbersome, please refer to FireFox installation guide.
3. HTML selectors `xpath`, `css` and `tag` require `lxml` which compiles low-level extensions during pip installation. So again, you either install `python-dev`, or install `lxml` from OS repo:

   ```
   apt install python-lxml
   ```

1.2 Configuration

1.2.1 Location

`kibitzr` reads configuration from `kibitzr.yml` file. It tries to find it in following places:

1. `./kibitzr.yml` - current working directory.
2. `~/.config/kibitzr/kibitzr.yml`
3. `~/kibitzr.yml`

`kibitzr-creds.yml` can be used to store credentials, it must be placed in the same directory as `kibitzr.yml`.

1.2.2 Format

`kibitzr` serves list of `checks`.

Each check may have a `name`. If `name` is present it **must be unique**. If no name is provided, it will be auto-generated.

The name is used in notifications and internally as a check identifier.

Check may have `url`. If it is provided, it will be used to fetch data. Optionally `verify-cert` can be set to `False` to skip verificaiton of the SSL certificate. Alternativelly data can be fetched by `script`, which is an arbitrary shell script.

Check will be executed every `period` seconds and/or on every `schedule`. See *Schedule documentation* for a complete list of possibilities.

Fetched data from `url` (or `script` output) is passed to a pipeline of transformations defined under `transform` key. See *Transforms documentation* for a complete list of supported transformations.

Finally transformed data is passed to a list of notifiers defined under `notify` key. See *Notifier documentation* for a complete list of supported notifiers.

Kibitzr supports browser interactions. They can be activated by using any of keys:

1. `delay` - number of seconds to wait after page loaded in browser to process JavaScript.
2. `scenario` - python scenario acting on selenium driver after page load.
3. `form` - shorthand for simple selenium scenarios.

Browser interaction is a strong side of Kibitzr and a tough article in itself. Please refer to *Browser automation* documentation.

1.2.3 Environment variables

Kibitzr provides read access to environment variables in a number of ways.

Inside *Python support* scripts, use Pythonic builtin module `os`:

```
import os
os.environ['NAME']
```

In shell scripts use bash syntax:

```
echo ${NAME}
```

Jinja templates have `env` dictionary in their context:

```
{{ env.NAME }}
```

`kibitzr-creds.yml` supports bash-like environment interpolation provided by yamlenv library:

```
service:
    username: ${ USERNAME }
    password: ${ PASSWORD }
```

1.2.4 Example break down

Let's start with something simple. It's not very useful check, but it shows the basics.

```
checks:
  - name: Current Time
    url: https://www.worldtimeserver.com/current_time_in_US-NY.aspx
    transform:
      - css: "span#theTime"
      - text
    notify:
      - python: print(content)
    period: 15
```

Copy paste it to your `kibitzr.yml` and launch `kibitzr`. You will see something like this:

```
$ kibitzr once
2017-03-28 22:02:39,465 [INFO] kibitzr.checker: Fetching Current Time at https://www.
↪worldtimeserver.com/current_time_in_US-NY.aspx
2017-03-28 22:02:39,687 [INFO] kibitzr.notifier.custom: Executing custom notifier
10:02:39 pm
EDT
2017-03-28 22:02:39,687 [INFO] kibitzr.main: Scheduling checks for 'Current Time'␣
↪every 15 seconds
2017-03-28 22:02:39,688 [INFO] kibitzr.main: Starting infinite loop
2017-03-28 22:02:54,705 [INFO] schedule: Running job Every 15 seconds do check()␣
↪(last run: [never], next run: 2017-03-28 22:02:54)
2017-03-28 22:02:54,705 [INFO] kibitzr.checker: Fetching Current Time at https://www.
↪worldtimeserver.com/current_time_in_US-NY.aspx
2017-03-28 22:02:54,823 [INFO] kibitzr.notifier.custom: Executing custom notifier
10:02:54 pm
EDT
```

Let's follow the configuration file line-by-line to see how it works.

On the first line we define a dictionary key `checks`:

```
checks:
```

Then, starting with indentation and dash goes the name of the first check:

```
- name: Current Time
```

It's an arbitrary string, the only constraint is that it must be **unique** within the `checks` list.

Right after name, we define URL:

```
url: https://www.worldtimeserver.com/current_time_in_US-NY.aspx
```

Please note, that all keys are in lower case.

So far so good, we came to transformations:

```
transform:
  - css: "span#theTime"
  - text
```

`transform` value must be a list (as denoted by dashes). Please note how list items indentation is deeper, than of `transform`.

Each `transform` item can be a simple `transform` name (like `text`, which extracts text from HTML), or a `name: argument` pair (like `css: "#qlook > div"` which crops HTML using CSS selector `"#qlook > div"`)

As you can see, we first crop whole page to a single HTML tag and then extract plain text from it.

Having all the hard job behind, we came to notification settings. `kibitzr` supports *many different notifiers*, but here we are using the one, that does not require credentials management - arbitrary Python script.

```
notify:
  - python: print(content)
```

It is exactly the code, that produced

```
10:02:39 pm
EDT
```

in the `kibitzr` output.

Last line of configuration file is the `period`:

```
period: 15
```

The number of seconds to wait between (*start of*) checks. Kibitzr understands time to the extent, you can write `1 hour` instead of `3600`. For the more complete list of available formats refer to pytimeparse docs.

1.3 Credentials

It's always a good idea to store general configuration separately from sensetive information, like usernames and password. On the one hand, it's good from security perspective, on the other hand it allows creating common check definitions.

1.3.1 Plain YAML storage

Kibitzr loads arbitrary data structures from `kibitzr-creds.yml` and makes it available inside transforms, fetchers and notifies through `creds` variable. It's not the most secure way of storing passwords. The good idea is to make it accessible only by owner:

```
$ chmod 600 kibitzr-creds.yml
```

While it does not feel safe to store bank accounts this way, it's a good fit for API keys (like Telegram, or Slack).

1.3.2 System Keyring

All modern operating systems provide some form of secure credentials storage. But they usually require additional configuration.

Kibitzr provides access to keyrings through python keyring. To enable, install Kibitzr extension kibitzr-keyring:

```
$ pip install kibitzr-keyring
```

And follow the keyring instructions.

Once configured, keyring values will be available in `creds.keyring` dictionary.

1.4 Usage

```
$ kibitzr --help
Usage: kibitzr [OPTIONS] COMMAND [ARGS]...

  Run kibitzr COMMAND --help for detailed descriptions

Options:
  -l, --log-level [debug|info|warning|error]
                                  Logging level
  --help                          Show this message and exit.

Commands:
```

(continues on next page)

(continued from previous page)

```
  clean          Clean change history
  firefox        Launch Firefox with persistent profile
  init           Create boilerplate configuration files
  once           Run kibitzr checks once and exit
  reload         Send signal to reload configuration for kibitzr process
  run            Run kibitzr in the foreground mode
  telegram_chat  Return chat id for the last message sent to...
  version        Print version
```

CLI reads its configuration from `kibitzr.yml` file in current working directory. Optionally `kibitzr-creds.yml` can be used to separate credentials from general configuration.

Please refer to *configuration documentation* for `kibitzr.yml` format.

For commands `run` and `once` one or more `NAME`'s can be supplied to limit execution of configuration file to a subset of tasks.

`kibitzr` doesn't have daemon mode. Instead it can be launched with supervisord. See Running kibitzr as a daemon for details.

1.5 Browser automation

Kibitzr uses Firefox browser and Selenium Python library for browser automation.

Installing Firefox can be cumbersome, please refer to FireFox installation guide.

Simple HTML forms can be filled using `form` key, but complex scenarios require full power of Python (Selenium) scripting.

1.5.1 Filling simple forms

Imagine, for example, that you need to authorize on a site before fetching content. For common case the check will look like:

```yaml
checks:
  - name: Bank account balance
    url: https://bank.com
    form:
      - id: login
        creds: bank.login
      - id: password
        creds: bank.password
    ... (transform and notify) ...
```

Key `id: login` means that HTML element fill be found using ID selector `login`. Key `creds: bank.login` means that input's value will be taken from `creds` dictionary using `bank.login` as a hierarchy path. Check assumes that `kibitzr-creds.yml` contains:

```yaml
bank:
  login: mr.robot
  password: 123&dSLHj*sdfa
```

1.5.2 Available Form Selectors

Field can be selected using one of the three selectors: id, css, xpath. (Make sure to use lowercase).

1.5.3 Available Field Value Generators

As in example above, field can be filled from `creds` dictionary. Another option is to provide Jinja2 template in key `value`. Template will have access to `conf` and `creds`. However any plain text value can be passed as well. For example, the same value, as in creds example can be rendered by:

```
checks:
  - name: Bank account balance
    url: https://bank.com
    form:
      - id: login
        value: "{{ creds['bank']['login'] }}"
      - id: password
        value: "{{ creds['bank']['password'] }}"
    ... (transform and notify) ...
```

Note: don't forget to wrap Jinja2 template in quotes, since curly bracket is a valid YAML markup for dictionary. Please refer to Jinja2 template documentation for details.

1.5.4 Python scenarios with Selenium

For complex cases Kibitzr provides access to Selenium driver. Here is an example of filling current date into form field:

```
checks:
  - name: Daily updates
    url: https://daily.com
    scenario: |
      import datetime
      today = datetime.date.today()
      element = driver.find_element_by_id('datefield-1')
      element.send_keys(today.strftime('%m/%d/%Y')
      run = driver.find_element_by_id('run-button')
      run.click()
    ... transforms and notify ...
```

1.5.5 Wait for Javascript to render contents

Sometimes web page uses some complex Javascript to render a page after it is loaded. These pages don't require form filling, or complex scenarios, simple delay will do. To define delay add `delay` key with number of seconds to wait:

```
checks:
  - url: https://www.producthunt.com/posts/kibitzr
    delay: 1
    ... transforms and notify ...
```

1.5.6 Working around two-factor authentication

Some sites require entering code sent in a SMS for logging from the new device. 2-FA can't be automated without weakening security. But Kibitzr can use persistent Firefox profile. Start persistent Firefox session with

```
$ kibitzr firefox
```

Than authenticate on all sites, that require first-login 2-FA. When ready, hit Return in the terminal prompt. New profile will be saved in `firefox_profile` directory. If this directory exists, kibitzr will load it for each following run.

Note: if running kibitzr remotely through SSH, use X11 forwarding.

1.5.7 Debugging/Troubleshooting

Writing robust Selenium scenarios is no easy task, and most likely it won't work from the first time. Kibitzr has a few options to help with debugging.

1. See what happens in Firefox by running in foreground mode. Just add

   ```
   checks:
     - url: ...
       scenario:
         ...
       headless: false
   ```

   to check dictionary.

2. Launch Pdb within scenario and explore step-by-step.

   ```
   checks:
     - url: https://javascript-labyrinth.io
       scenario:
         import pdb; pdb.set_trace()
         ...
   ```

3. Experiment inside Jupyter notebook. See *notebook example*.

1.6 Schedule

Kibitzr checks are scheduled to run according to the `period` and `schedule` configuration options. When no `period` or `schedule` is defined, a check has the default period of 5 minutes.

Keep in mind that Kibitzr runs checks sequentially, so there is no guarantee on the precise start time of a check, one long-running check delays the next one.

1.6.1 Period

The number of seconds to wait between (start of) checks. The `period` option can handle any of the formats supported by pytimeparse (e.g., 37 minutes, 2 hours)

```
checks:
  - name: Current Time
    ...
    period: 15
```

(continues on next page)

(continued from previous page)

```
  - name: Fancy check
    ...
    period: 2 hours
```

1.6.2 Schedule

`schedule` option provides finer control over start time, it can be set as a single item or as a list of items. A check runs for each `schedule` item configured.

The syntax is:

- `every` (integer) - interval length, required;
- `unit` (string) - unit of the interval;
- `at` (string) - time to run the check in HH:MM format. Applicable only if unit is "days".

The rule mimics the pattern: "Every `every` `unit` at `at`."

```
checks:
  ...
  schedule:
     every: 1
     unit: days
     at: "12:00"

  ...
  schedule:
     every: 1
     unit: hours
```

`unit` is one of `seconds`, `minutes`, `hours`, `days` or `weeks`. When `every` is set to `1` it can be condensed with the single version of the `unit`:

```
checks:
  ...
  schedule:
     every: day
     at: "12:00"
```

Optionally, `every` can also be one of `monday`, `tuesday`, `wednesday`, `thursday`, `friday`, `saturday` or `sunday`.

```
checks:
  ...
  schedule:
     every: thursday
     at: "12:00"
```

1.6.3 Examples

```
checks
  - name: Late alarm
```

(continues on next page)

(continued from previous page)

```
    ...
    schedule:
      every: 1
      unit: day
      at: "20:30"

  - name: Crazy scheduling
    ...
    schedule:
    - every: day
      at: "15:30"
    - every: hour
    - every: saturday
      at: "12:13"
```

For a detailed list of scheduling options, see schedule documentation which powers the Kibitzr scheduler.

1.7 Transforms

Each Kibitzr transform modifies content and passes it forward. Transforms can be divided into following groups: HTML, plain text, JSON.

1.7.1 HTML

- `tag: tagname` - crop HTML to contents of the first matching HTML tag.
- `css: selector` - crop HTML to the first encountered outer HTML matching passed CSS selector.
- `css-all: selector` - crop HTML to the concatenated list of all matching elements.
- `xpath: path` - crop HTML to contents of the passed XPath.
- `xpath-all: path` - crop HTML to the concatenated list of all matching elements.
- `text` - strip all HTML tags and return only text.

1.7.2 Plain text

- `changes` - Compare to the previous version of the content and return difference report.
- `changes: verbose` - Same as `changes`, but in human-friendly format.
- `changes: word` - Same as `changes`, but highlight changes within a string.
- `jinja: template` - Render Jinja2 template. See *jinja transform* for reference.

1.7.3 Code

- `python: code` - Execute arbitrary *Python *code** on passed content.
- `shell: code` - Execute arbitrary *Shell support code* on passed content. Call `grep`, `awk` or `sed`, for example.

1.7.4 JSON

- `json` - Pretty print JSON content.
- `jq` - Apply jq JSON transformation (jq must be installed).

1.7.5 Jinja Transform

Kibitzr supports Jinja2 templates. Following variables are passed into a context:

- `conf` - check configuration dictionary
- `stash` - global persistent key-value storage; See *Stash* for details
- `content` - input as plain text
- `lines` - input as a list of lines
- `json` - input parsed from JSON
- `css` - crop input HTML to CSS selector, similar to `css-all` transform
- `xpath` - crop input XML to XPath selector, similar to `xpath` transform
- `env` - environment variables dictionary.

Also set of built-in Jinja filters is extended with:

- `text` - strip all HTML tags and return only text
- `float` - remove all characters except numbers and point.
- `int` - convert text or float to integer

Because Jinja transform uses general-purpose template engine, it can supersede simpler transforms. However greater powers come with more points of failure. Debugging of failed Jinja2 template might be challenging. Generally I recommend using it only if you can't achieve desired effect without it.

1.7.6 Examples

Here is a sequence of transformations, that will

1. Crop HTML page to CSS selector `#plugin-description > div > p > a`
2. Transform it's contents to text
3. Compare it to previous value and report difference in human-readable form.

```
- css: "#plugin-description > div > p > a"
- text
- changes: verbose
```

Complete `kibitzr.yml` could look like this:

```
checks:
  - name: JetPack updates
    url: https://wordpress.org/plugins/jetpack/
    transform:
      - css: "#plugin-description > div > p > a"
      - text
      - changes: verbose
```

(continues on next page)

(continued from previous page)

```
    notify:
      - smtp: me@gmail.com
    period: 3600
```

When launched first time, it will send e-mail to me@gmail.com with contents:

```
Download Version 4.6
```

Once page contents changes, on next kibitzr launch the e-mail will be:

```
Previous value:
Download Version 4.6
New value:
Download Version 4.7
```

Next config will notify on new Kibitzr releases published on GitHub:

```
checks:
  - name: Kibitzr releases
    url: https://api.github.com/repos/kibitzr/kibitzr/releases
    transform:
      - jq: ".[] | .tag_name + \" \" + .name"
      - changes
    notify:
      - slack
    period: 3600
```

Example Slack message:

```
@@ -1,2 +1,3 @@
+ "v2.6.2 Added jq transformer"
  "2.6.1 Fixed git repo configuration"
  "2.6.0 Added \"changes: verbose\" transformer"
```

1.8 Notifiers

If *transformation sequence* produced non-empty text, list of notifiers will be called.

Kibitzr supports following notifier types:

1. `smtp` - Send an e-mail through any SMTP server; See SMTP notifier docs for details
2. `mailgun` - or send it through mailgun API
3. `slack` - Trigger Slack Incoming Webhook
4. `telegram` - Send message through private Telegram Bot
5. `zapier` - Trigger Zapier Catch Hook
6. `gitter` - Post to gitter's chat
7. `gotify` - Push notification via Gotify
8. `python` - Run *Python script*
9. `shell` - Run *shell script*
10. `stash` - Save to persistent global key-value storage; See *Stash* for details

Each notifier requires different configuration. For the sake of security, sensitive information like API tokens, usernames and passwords can (and should) be stored in separate file - `kibitzr-creds.yml` It's recommended to restrict access to this file to the owner.

1.8.1 Example configurations

```
smtp:
    host: smtp.gmail.com
    port: 587
    user: kibitzrrr@gmail.com
    password: (sat;hfsDA5wa@$%^jh

mailgun:
    key: key-asdkljdiytjk89038247102380
    domain: sandbox57895483457894350345.mailgun.org
    to: John Doe <john.doe@gmail.com>

slack:
    url: https://hooks.slack.com/services/T5665TUV/B21J7KCTX/Ov2xUt84atxi4yjvBnEqMIKX

gitter:
    url: https://webhooks.gitter.im/e/24a1042f49211ca9504a

telegram:
    token: 343558405:ABHCRh_rnzO554skSlISotUnNFWt3p8P004

zapier:
    url: https://hooks.zapier.com/hooks/catch/1670195/9asu13/

gotify:
    url: https://gotify.example.de/
    token: A0dIInnCs1J1zNN
```

1.9 Python support

Kibitzr check accepts Python code in 4 places:

- Script (fetcher)
- Browser automation scenario
- Transform
- Notify

Here is a simplistic example of `kibitzr.yml` file, that uses all three:

```
checks:
  - name: Python example
    script:
       python: |
                content = "\n".join([str(x**2) for x in range(1, 4)])
    transform:
          - python: content = " ".join(reversed(content.splitlines()))
    notify:
          - python: print(content)
```

Once executed with debug log level it will generate following output:

```
$ kibitzr -l debug once
2017-04-22 10:47:04,401 [DEBUG] kibitzr.conf: Loading settings from /home/kibitzr/
↪kibitzr.yml
2017-04-22 10:47:04,404 [DEBUG] kibitzr.conf: Loading credentials from /home/kibitzr/
↪kibitzr-creds.yml
2017-04-22 10:47:04,406 [INFO] kibitzr.checker: Fetching 'Python example' using script
2017-04-22 10:47:04,406 [INFO] kibitzr.fetcher.script: Fetch using Python script
2017-04-22 10:47:04,406 [DEBUG] kibitzr.fetcher.script: content = "\n".
↪join([str(x**2) for x in range(1, 4)])

2017-04-22 10:47:04,406 [INFO] kibitzr.transformer: Python transform
2017-04-22 10:47:04,406 [DEBUG] kibitzr.transformer: content = " ".
↪join(reversed(content.splitlines()))
2017-04-22 10:47:04,407 [DEBUG] kibitzr.checker: Sending report: u'9 4 1'
2017-04-22 10:47:04,407 [INFO] kibitzr.notifier.custom: Executing custom notifier
2017-04-22 10:47:04,407 [DEBUG] kibitzr.notifier.custom: print(content)
9 4 1
```

Let's break it down.

1.9.1 Python Fetcher

To fetch content with a script instead of URL, check must have no `url` key, and have `script` defined. If `script`'s value is a string, it will be used as shell script. Alternatively `script` can hold a dictionary of one item. Item's key can be `shell` (for *Shell fetcher*) or `python`. If `script`'s only key is `python`, then it's value will be executed as a Python script. Script is an arbitrary Python code with few constraints:

1. Script can define `ok` boolean variable, which is either `True` or `False`. When ok is `True` it means that content was fetched without errors. When ok is `False`, content should hold error message. By default `ok` is `True`.
2. Script must define `content` string variable. `content` will be passed to through `transform` list to `notify` list.
3. Script has access to check's configuration in `conf` global variable and credentials dictionary in `creds`.

If fetching script raises an exception, the fetcher will return `ok=False` and `content` will contain full traceback.

1.9.2 Browser Automation Scenario

Kibitzr allows writing browser automation scenarios using Selenium library. Scenario is an arbitrary Python code, which is executed after page is loaded in the browser. Scenario has access to following global variables:

1. Check's configuration in `conf` global variable.
2. Credentials dictionary in `creds`.
3. Selenium driver in `driver`

Example scenario that authenticates in online account of Bank of America:

```
checks:
  - name: BofA
    url: https://www.bankofamerica.com/
    scenario: |
        login = driver.find_element_by_id("onlineId1")
        login.send_keys(creds['bofa']['login'])
```

(continues on next page)

(continued from previous page)

```
        password = driver.find_element_by_id("passcode1")
        password.send_keys(creds['bofa']['password'])
        button = driver.find_element_by_id("hp-sign-in-btn")
        button.click()
```

Using Selenium is an advanced topic on it's own with a plenty of documentation and many pitfalls.

1.9.3 Python Transform

Python transform is similar to Python fetcher with one difference. It accepts `content` variable and it puts transformed result in the same `content` variable.

```
transform:
  - python: |
      content = content.replace("election", "eating contest")
```

1.9.4 Python Notifier

Python notify is similar to Python fetcher with one difference. It does not return anything.

1.9.5 Troubleshooting

To put break point inside Python code, just add following line:

```
import pdb; pdb.set_trace()
```

It will stop Kibitzr execution and start Pdb session. You will have access to all variables and full execution Stack. However, Pdb won't show current line of code, which is not convenient, but manageable, since you know exactly where break point stands.

1.10 Shell support

Kibitzr accepts shell scripts in 3 places:

- Fetch
- Transform
- Notify

Execute code from notifier with transformation result passed via stdin.

Here is a simplistic example of `kibitzr.yml` file, that uses all three:

```
checks:
  - name: Shell example
    script: |
      for i in seq 1 3
      do
          echo "Number $i"
      done
    transform:
```

(continues on next page)

(continued from previous page)

```
    - shell: grep 2
  notify:
    - shell: tac
```

Let's break it down.

1.10.1 Shell Fetcher

If `script`'s value is a string, it will be used as shell script. Alternatively `script` can hold a dictionary of one item. Item's key can be `shell` (or `python` for *Python fetcher*). If `script`'s only key is `shell`, then it's value will be executed as a Shell script. Under Linux, executor is `bash`, under Windows - `cmd.exe`. Script is an arbitrary shell code. It's output will be passed to transforms. If exit code is not zero, check will be aborted. Shell scripts don't have access to credentials, but inherit Kibitzr environment.

1.10.2 Shell Transform and Notifier

Transform and notifier are similar to fetcher. Except that they receive content via stdin, and notifier's stdout is ignored.

1.10.3 Example

Returning to the example, execution will go as follows:

```
[DEBUG] kibitzr.conf: Loading settings from /home/deminp/kibitzr/tmp/kibitzr.yml
[INFO] kibitzr.fetcher.loader: Fetching 'Shell' using script
[DEBUG] kibitzr.bash: Saving code to '/tmp/tmpTTPSxA.bat'
[DEBUG] kibitzr.bash: Launching script '/tmp/tmpTTPSxA.bat'
[DEBUG] kibitzr.bash: Command exit code: 0
[DEBUG] kibitzr.bash: Command stdout: Number 1
Number 2
Number 3

[DEBUG] kibitzr.bash: Command stderr:
[DEBUG] kibitzr.bash: Saving code to '/tmp/tmpV4Grg8.bat'
[DEBUG] kibitzr.bash: Launching script '/tmp/tmpV4Grg8.bat'
[DEBUG] kibitzr.bash: Command exit code: 0
[DEBUG] kibitzr.bash: Command stdout: Number 2
[DEBUG] kibitzr.bash: Command stderr:
[DEBUG] kibitzr.notifier.factory: Sending report: u'Number 2'
[DEBUG] kibitzr.bash: Saving code to '/tmp/tmpm6sRVx.bat'
[DEBUG] kibitzr.bash: Launching script '/tmp/tmpm6sRVx.bat'
[DEBUG] kibitzr.bash: Command exit code: 0
[DEBUG] kibitzr.bash: Command stdout: 2 rebmuN
[DEBUG] kibitzr.bash: Command stderr:
```

Fetcher script produced output:

```
Number 1
Number 2
Number 3
```

Shell transform filtered lines that contain 2:

```
Number 2
```

Notifier echoed reversed line:

```
2 rebmuN
```

Notifier's stdout is ignored, so we don't see it along Kibitzr output.

And here is what happens when shell script produces error:

```
$ cat kibitzr.yml
checks:
  - name: Shell
    script: ls /non-existing
    notify:
      - shell: rev

$ kibitzr -l debug once
[DEBUG] kibitzr.conf: Loading settings from /home/deminp/kibitzr/tmp/kibitzr.yml
[INFO] kibitzr.fetcher.loader: Fetching 'Shell' using script
[DEBUG] kibitzr.bash: Saving code to '/tmp/tmpyNakOP.bat'
[DEBUG] kibitzr.bash: Launching script '/tmp/tmpyNakOP.bat'
[ERROR] kibitzr.bash: Command exit code: 2
[ERROR] kibitzr.bash: Command stdout:
[ERROR] kibitzr.bash: Command stderr: ls: cannot access '/non-existing': No such file␣
↪or directory
[DEBUG] kibitzr.transformer.factory: Notifying on error
[DEBUG] kibitzr.notifier.factory: Sending report: u"ls: cannot access '/non-existing
↪': No such file or directory"
[DEBUG] kibitzr.bash: Saving code to '/tmp/tmpqdZwKI.bat'
[DEBUG] kibitzr.bash: Launching script '/tmp/tmpqdZwKI.bat'
[DEBUG] kibitzr.bash: Command exit code: 0
[DEBUG] kibitzr.bash: Command stdout: yrotcerid ro elif hcus oN :'gnitsixe-non/'␣
↪ssecca tonnac :sl
[DEBUG] kibitzr.bash: Command stderr:
```

1.11 Stash

1.11.1 Overview

Kibitzr maintains persistent key-value storage - `stash`. All data inside `stash` is accessible inside all checks and can be referred from *Python Fetcher* and *Jinja Transform*.

Stash keys are populated in notify. Use `stash` notifier and provide it key-value dictionary. Each value is a Jinja template. It has access to the same context as Jinja transform.

Stored values can be printed with command:

```
$ kibitzr stash
```

1.11.2 Example

Good application for stash is checks aggregation. Consider this news digest example:

```
checks:
  - url: http://www.foxnews.com/
    transform:
      - jinja: '{{ css("h1")[-1] }}'
      - text
      - jinja: '{{ lines | join(" ") }}'
    notify:
      - stash:
          fox: '{{ content }}'

  - url: https://www.nytimes.com/
    transform:
      - xpath: //*[starts-with(@id, "topnews-")]/h2/a
      - text
    notify:
      - stash:
          nytimes: '{{ content }}'

  - name: Headlines
    script:
      python: |
        content = (
          "Fox News: {0}\n"
          "NY Times: {1}"
        ).format(stash["fox"], stash["nytimes"])
    notify:
      - python: print(content)
```

First check, `Fox News`, will fetch headline from foxnews.com. Second - from nytimes.com. They both will save headings in stash under respective keys.

Last check, `Headlines`, uses Python script that will print something similar to:

> Fox News: LASHING OUT AT LEAKS Trump calls US disclosures in UK bombing 'deeply troubling'
> NY Times: Trump Calls for U.S. Inquiry Into Leaks on Manchester

1.11.3 Implementation

Under the hood stash uses python built-in shelve module. It stores all data in `stash.db` file in working directory. Writes are atomic - if one of values fails rendering, none will be written.

1.12 Extensions

Kibitzr is built for extendability. It uses stevedore for loading external python libraries in preset entry points. Official extensions live in GitHub organization. Here is a short overview of them.

1. kibitzr-sentry - send all errors and exceptions to the Sentry.
2. kibitzr-email - make checks for incoming emails (IMAP only).
3. kibitzr-keyring - use OS keyring for storing credentials.

1.13 Recipes

1.13.1 GitHub API release notification

```
checks:
  - name: Kibitzr GitHub release
    url: https://api.github.com/repos/kibitzr/kibitzr/releases/latest
    transform:
      - jq: .tag_name + " " + .name
      - changes: verbose
    notify:
      - slack
```

Example message:

> **Kibitzr**
> Previous value:
> v2.6.6 Added batch syntax to configuration file
> New value:
> v2.6.7 Invoke jq with –raw-output

1.13.2 Wordpress Plugin update (featuring batch syntax)

```
checks:
  - batch: "Wordpress plugin {0} updates"
    transform:
        - xpath: '//*[@id="changelog"]/h4[1]'
        - text
        - changes: verbose
    notify:
        - slack
    period: 3600
    url-pattern: "https://wordpress.org/plugins/{0}/"
    items:
      - advanced-custom-fields
      - akismet
      - better-wp-security
      - black-studio-tinymce-widget
      - contact-form-7
      - disable-comments
      - duplicate-post
```

1.13.3 Travis CI build status

```
checks:
  - name: Kibitzr Build Status
    url: https://travis-ci.org/kibitzr/kibitzr
    transform:
      - css: div.build-info > h3
      - text
      - changes
    delay: 1
```

(continues on next page)

(continued from previous page)

```
    period: 600
    notify:
      - slack
```

1.13.4 TeamCity build status change

```
checks:
  - name: TeamCity Build
    template: teamcity-build
    url: https://teamcity/viewQueued.html?itemId=10270004

templates:
  teamcity-build:
    form:
      - xpath: '//*[@id="pageContent"]/form/table/tbody/tr[4]/td/span/a[1]'
        click: true
    delay: 3
    transform:
      - xpath: //*[@id="buildResults" or contains(@class, "statusBlock")]//table/
↪tbody/tr[1]/td[2]
      - text
      - jinja: "{{ lines | join(' ') }}"
      - changes: new
    period: 30 seconds
```

1.13.5 BitBucket pull request ready to merge

```
checks:
  - name: PR ready to merge
    template: bitbucket-pr-ready
    url: https://bitbucket/repos/kibitzr/pull-requests/307/overview

templates:
    bitbucket-pr-ready:
        xpath: //*[@class="plugin-section-primary"]
        format: text
        period: 30
        delay: 5
        scenario: bitbucket-login

scenarios:
    bitbucket-login: |
        from selenium.common.exceptions import NoSuchElementException
        try:
            driver.find_element_by_id("j_username").send_keys("username")
            driver.find_element_by_id("j_password").send_keys("password")
            driver.find_element_by_id("submit").click()
        except NoSuchElementException:
            # Second time session will be already authorized
            pass
```

1.13.6 Air Pollution in Paris via Telegram

```
checks:
  - name: Air Quality Today in Paris
    url: https://www.airparif.asso.fr/accueil-airparif
    delay: 3
    transform:
      - css-all: ".indice-color.text-light"
      - text
      - jinja: |
            Pollution in Paris
          **Today**
          Ozone:  {{ lines.0.lower() }}
          Dioxyde d'Azote: {{ lines.1.lower() }}
          Particules PM10:{{ lines.2.lower() }}
          Particules PM2: {{ lines.3.lower() }}
          **Forecast for Tomorrow**
          Ozone:  {{ lines.4.lower() }}
          Dioxyde d'Azote: {{ lines.5.lower() }}
          Particules PM10:{{ lines.6.lower() }}
          Particules PM2: {{ lines.7.lower() }}
    notify:
      - telegram
```

1.14 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.14.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/kibitzr/kibitzr/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

kibitzr could always use more documentation, whether as part of the official kibitzr docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/kibitzr/kibitzr/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.14.2 Get Started!

Ready to contribute? Here's how to set up *kibitzr* for local development.

1. Fork the *kibitzr* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/kibitzr.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv kibitzr
$ cd kibitzr/
$ pip install -e . -r requirements/dev.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 kibitzr tests
$ pytest
$ tox
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push -u origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.14.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.

1.14.4 Tips

To run a subset of tests pass it as an argument to pytest:

```
$ pytest tests/unit/transforms
```

1.15 Contributors

People, who contributed to Kibitzr in arbitrary order:

- Igor Sobolev
- Delirious Lettuce
- Martin Virtel
- Attila Nagy
- Niklas Heer
- Francesco Barresi
- Maciek Starzyk
- ColdIce
- Jesaja Everling
- egvimo
- fi-do
- QJKX

Python Module Index

k

kibitzr.bash, 17

Index

K

kibitzr.bash (*module*), 17