watcher

Release 0.2.0

Raciel Hernandez B.

Mar 16, 2022

FIRST STEPS

Watcher simplifies the integration of non-connected systems by detecting changes in data and facilitates the development of monitoring, security and process automation applications. Think of Watcher as an intercom or a bridge between different servers or between different applications on the same server. Or you can simply take advantage of Watcher's capabilities to develop your project.

"Watch everything" Currently the functionality of detecting changes in the file system is implemented. However, the project has a larger scope and we invite you to collaborate with us to achieve the goal of "Watch Everything". One step at a time! Come on and join us.

Starting with the file system Yes, we have started implementing watcher to observe and detect changes in the file system. You can use watcher to discover changes related to file creation, file deletion and file alteration. You can find out more about our all the *File Watcher Features* in these pages.

Watcher is Free, Open Source and User Focused Our code is free and open source. We like open source but we like socially responsible software even more. Watcher is distributed under MIT license.

CHAPTER

ONE

FIRST STEPS

Your project needs to process inputs that trigger your business logic but those inputs are out of your control? Do you want to integrate your project based on detection of file system changes? Learn about the great options Watcher offers for advanced change detection that you can leverage for your project development.

- **Getting started**: *Installation Prerequisites* | *Quick Installation* | *Feature Overview*

CHAPTER

TWO

FILE WATCHER FEATURES

Currently **Watcher** comprises the following features: *Single File & Folders*, *Multiples File Groups*, *File Patterns*, *Non-Bloking Execution*, *Blocking Execution*, *Bulk File Processing*, *Advanced File Deletion*, *Advanced File Creation*, *Advanced File Alteration*, *Watcher for Any Alteration*, *Watcher for Specific Alteration*, *Decoupled Execution*, *Novelty Detection*, *Qualitative Response*, *Check File Stability*, *Big Amounts of Files*, *Atomic Function Injection*, *Folder Recursion*, *Selective Path Level*, *Watcher Monitoring*

2.1 Prerequisites

If you like containers we have one ready in **DocketHub** (watcher) but if you want to install **Watcher** you should consider the following requirements:

2.1.1 Tarantool

Watcher runs on Tarantool. Tarantool is an In-memory computing platform. For installation follow the instructions on the tarantool.io website.

Before you begin, ensure you have met the following requirements:

- **Tarantool**: >= `1.7`.

Note: If you already have Tarantool installed you can skip this step.

2.1.2 Supported Platforms

- **POSIX Compliant**: `Unix`, `MacOsx`, `Linux`, `Freebsd`.
- **POSIX for Windows**: `Cygwin`, `Microsoft POSIX Subsystem`, `Windows Services for UNIX`, `MKS Toolkit`.

Warning: Watcher has not been tested on POSIX Windows Systems.

2.2 Quick Instalation

There are several ways to install **Watcher** on your server. Choose the option that suits you best and go ahead!

2.2.1 From Docker

Get **Watcher** container from a docker image:

```
docker pull racherb/watcher:latest
docker run -i -t racherb/watcher
```

Note: **Use docker volumes**. If you want to look at the host or remote machine's file system then start a container with a `volume`.

The following example enables a volume on the temporary folder `/tmp` of the host at path `/opt/watcher/host/` of the container.

`docker run -i -t -v /tmp/:/opt/watcher/host/tmp racherb/watcher`

2.2.2 From DEB Package

Quick installation from DEB Package:

```
curl -s https://packagecloud.io/install/repositories/iamio/watcher/script.deb.sh | sudo␣
↪bash
sudo apt-get install watcher
```

Note: *DEB Quick install* is available for the following distributions:

- **Debian**: `Lenny`, `Trixie`, `Bookworm`, `Bullseye`, `Buster`, `Stretch`, `Jessie`.
- **Ubuntu**: `Cosmic`, `Disco`, `Hirsute`, `Groovy`, `Focal`.
- **ElementaryOS**: `Freya`, `Loki`, `Juno`, `Hera`.

2.2.3 From RPM Package

First install the repository:

```
curl -s https://packagecloud.io/install/repositories/iamio/watcher/script.rpm.sh | sudo␣
↪bash
```

And install the package:

- For **RHEL** and **Fedora** distros: `sudo yum install watcher-0.2.1-1.noarch`.
- For **Opensuse** and **Suse Linux Enterprise**: `sudo zypper install watcher-0.2.1-1.noarch`.

Note: *RPM Quick install* is available for the following distributions:

- **RHEL**: 7, 6, 8.

- **Fedora**: 29, 30, 31, 32, 33.
- **OpenSuse**: 15.1, 15.2, 15.3, 42.1, 42.2, 42.3.
- **Suse Linux Enterprise**: 12.4, 12.5, 15.0, 15.1, 15.2, 15.3.

2.2.4 From Tarantool

Quick installation from **Utility Tarantool**:

Install watcher through Tarantool's `tarantoolctl` command:

```
tarantoolctl rocks install avro-schema
tarantoolctl rocks install https://raw.githubusercontent.com/racherb/watcher/master/
↪watcher-scm-1.rockspec
```

2.2.5 From LuaRocks

Make sure you have Luarocks installed first.

From the `terminal` run the following command:

```
luarocks install https://raw.githubusercontent.com/racherb/watcher/master/watcher-scm-1.
↪rockspec
```

2.3 Quick Start

Detection of `creation`, `deletion` and `alteration` of **single files** or **single folders** in the file system.

```
fwa = require('watcher').file                --for file-watcher
fwa.creation({'/path/to/single_file'})       --watching file creation
fwa.deletion({'/path/to/single_folder/'})    --watching folder deletion
fwa.alteration('/path/to/single_folder/*')   --watching file alteration
```

2.4 File Watcher Features

The **Watcher** module has been designed with the typical use cases of the Banking and Telecommunications industry in mind for *IT Batch Processing*.

If you know of a use case that is not covered by watcher, please tell us about it in the GitHub Discussions Section .

Currently **Watcher** comprises the following features: *Single File & Folders*, *Multiples File Groups*, *File Patterns*, *Non-Bloking Execution*, *Blocking Execution*, *Bulk File Processing*, *Advanced File Deletion*, *Advanced File Creation*, *Advanced File Alteration*, *Watcher for Any Alteration*, *Watcher for Specific Alteration*, *Decoupled Execution*, *Novelty Detection*, *Qualitative Response*, *Check File Stability*, *Big Amounts of Files*, *Atomic Function Injection*, *Folder Recursion*, *Selective Path Level*, *Watcher Monitoring*

Note: The lines of code used to exemplify each feature of watcher assume the following:

```
fwa = require('watcher').file   --for file-watcher
mon = require('watcher').monit  --for watcher monitoring
```

2.4.1 Single File & Folders

Detection of `creation`, `deletion` and `alteration` of **single files** or **single folders** in the file system.

```
fwa.creation({'/path/to/single_file'})    --watching file creation
fwa.creation({'/path/to/single_folder/'}) --watching folder creation
```

2.4.2 Multiples File Groups

Multiple groups of different files can be watched at the same time. The input list of watchable files is a Lua table type parameter.

```
fwa.deletion(
    {
        '/path_1/to/group_file_a/*',  --folder
        '/path_2/to/group_file_b/*'   --another
     }
 )
```

2.4.3 File Patterns

```
fwa.creation({'/path/to/files_*.txt'})
```

Note: The *watch-list* is constructed with a single flag that controls the behavior of the function: **GLOB_NOESCAPE**. For details type `man 3 glob`.

2.4.4 Non-Bloking Execution

By default the **Watcher** run is executed in non-blocking mode through tarantool fibers. Fibers are a unique Tarantool feature *"green threads"* or coroutines that run independently of operating system threads.

2.4.5 Blocking Execution

The `waitfor` function blocks the code and waits for a watcher to finish.

```
waitfor(fwa.creation({'/path/to/file'}).wid) --wait for watcher
```

2.4.6 Bulk File Processing

Watcher has an internal mechanism to allocate fibers for every certain amount of files in the watcher list. This amount is determined by the `BULK_CAPACITY` configuration value in order to optimize performance.

2.4.7 Advanced File Deletion

Inputs

Table 1: File Watcher Deletion Parameters

Param	Type	Description
wlist	`table`, `required`	Watch List
maxwait	`number`, `otional`, `default-value: 60`	Maximum wait time in seconds
interval	`number`, `otional`, `default-value: 0.5`	Verification interval for watcher in seconds
options	`table`, `optional`, `default-value: {'NS', 0, 0}`	List of search options
recursion	`table`, `optional`, `default-value: nil` or `{false, {0}, false}`	Recursion paramaters

wlist

It is the list of files, directories or file patterns to be observed. The data type is a Lua table and the size of tables is already limited to `2.147.483.647` elements.

An example definition is the following:

```
wlist = {'path/file', 'path', 'pattern*', ...} --arbitrary code
```

maxwait

Maxwait is a numeric value that represents the maximum time to wait for the watcher. Watcher will terminate as soon as possible and as long as the search conditions are met. The default value is `60 seconds`.

interval

Interval is a numerical value that determines how often the watcher checks the search conditions. This value must be less than the maxwait value. The default value is `0.5` seconds.

options

The options parameter is a Lua table containing 3 elements: `sort`, `cases` and `match`.

- The first one `sort` contains the ordering method of the `wlist`.
- The second element `cases` contains the number of cases to observe from the wlist.
- and the third element `match` indicates the number of cases expected to satisfy the search.

By default, the value of the option table is `{sort = 'NS', cases = 0, match = 0}`.

Table 2: The list of possible values for `sort`

Value	Description
`'NS'`	No sort
`'AA'`	Sorted alphabetically ascending
`'AD'`	Sorted alphabetically descending
`'MA'`	Sorted by date of modification ascending
`'MD'`	Sorted for date of modification descending

Note: The value `'NS'` treats the list in the same order in which the elements are passed to the list `wlist`.

recursion

To enable directory recursion you must define the recursion parameter. The recursion works only for an observable of type directory.

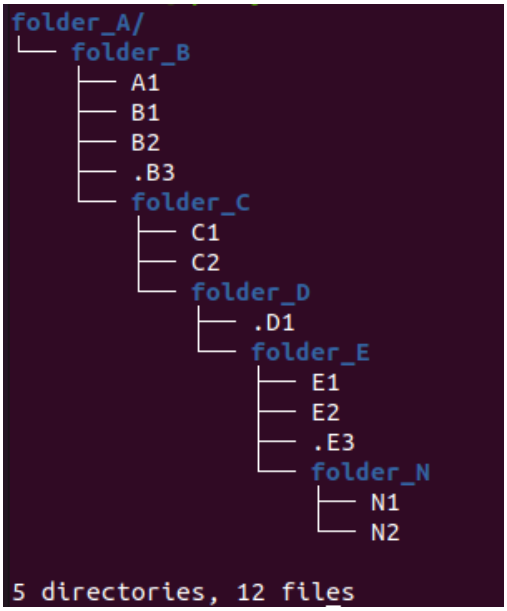
The recursion value is a Lua table type composed of the following elements `{recursive_mode, {deep_levels}, hidden_files}`:

- **recursive_mode**: Boolean indicating whether or not to activate the recursive mode on the root directory. The default value is `false`.
- **deep_levels**: Numerical table indicating the levels of depth to be evaluated in the directory structure. The default value is `{0}`
- **hidden_files**: Boolean indicating whether hidden files will be evaluated in the recursion. The default value is `false`.

How do the recursion levels work?

To understand how levels work in recursion, let's look at the following example.

Imagine you have the following directory structure and you want to observe the deletion of files from the path **'/folder_A/folder_B/'**.



The levels are determined from the object path or root path that will be used as input in the watcher expression. In this case the path **'/folder_A/folder_B/'** has level zero and, for each folder node a level will be added according to its depth. The result is shown in the following summary table, which contains the list of files for each level.

Table 3: Identification of the levels of recursion

	[Input] Level 0 `{0}`	Level 1 `{1}`	Level 2 `{2}`	Level 3 `{3}`	Level 4 `{4}`
folder	`'/folder_A/folder_B/'`	`'folder_C'`	`'folder_D'`	`'folder_E'`	`'folder_N'`
files	`{A1}` `{B1, B2, .B3}`	`{C1, C2}`	`{.D1}`	`{E1, E2, .E3}`	`{N1, N2}`

Note: The files, `.B3`, `.D1` and `.E3` are hidden files.

Now that we know how to set the recursion level, let's see an example of the observable files depending on different values of the **recursion** parameter for the above mentioned example.

Table 4: Observable files depending on the recursion level

`recursion` value	Composition of the list of observable files `wlist`
`{true, {0}, false}`	`{A1, B1, B2}`
`{true, {0}, true}`	`{A1, B1, B2, .B3}`
`{true, {0, 1}, false}`	`{A1, B1, B2, C1, C2}`
`{true, {0, 1}, true}`	`{A1, B1, B2, .B3, C1, C2}`
`{true, {2}, false}`	`nil`
`{true, {2}, true}`	`{.D1}`
`{true, {0, 1, 2, 3, 4}, false}`	`{A1, B1, B2, C1, C2, E1, E2, N1, N2}`
`{true, {0, 1, 2, 3, 4}, true}`	`{A1, B1, B2, .B3, C1, C2, .D1, E1, E2, .E3, N1, N2}`

Output

2.4.8 Advanced File Creation

Inputs

Table 5: File Watcher Creation Parameters

Param	Type	Description
wlist	`table`, `required`	Watch List
maxwait	`number`, `otional`, `default-value: 60`	Maximum wait time in seconds
interval	`number`, `otional`, `default-value: 0.5`	Verification interval for watcher in seconds
minsize	`number`, `optional`, `default-value: 0`	Value of the minimum expected file size
stability	`table`, `optional`, `default-value: {1, 15}`	Minimum criteria for measuring file stability
novelty	`table`, `optional`, `default-value: {0, 0}`	Time interval that determines the validity of the file's novelty
nmatch	`number`, `optional`, `default-value: 0`	Number of expected files as a search sufficiency condition

wlist

It is the list of files, directories or file patterns to be observed. The data type is a Lua table and the size of tables is already limited to `2.147.483.647` elements.

An example definition is the following:

```
wlist = {'path/file', 'path', 'pattern*', ...} --arbitrary code
```

maxwait

Maxwait is a numeric value that represents the maximum time to wait for the watcher. Watcher will terminate as soon as possible and as long as the search conditions are met. The default value is `60 seconds`.

interval

Interval is a numerical value that determines how often the watcher checks the search conditions. This value must be less than the maxwait value. The default value is `0.5` seconds.

minsize

Minsize is a numerical value representing the minimum expected file size. The default value is `0`, which means that it is sufficient to just generate the file when the minimum size is unknown.

Important: Regardless of whether the expected file size is `0 Bytes`, watcher will not terminate until the file arrives in its entirety, avoiding edge cases where a file is consumed before the data transfer is complete.

---

stability

The `stability` parameter contains the elements that allow to evaluate the stability of a file. It is a Lua table containing two elements:

- The `interval` that defines the frequency of checking the file once it has arrived.
- The number of `iterations` used to determine the stability of the file.

The default value is: `{1, 15}`.

novelty

The `novelty` parameter is a two-element Lua table that contains the time interval that determines the validity of the file's novelty. The default value is `{0, 0}` which indicates that the novelty of the file will not be evaluated.

nmatch

`nmatch` is a number of expected files as a search sufficiency condition.

2.4.9 Advanced File Alteration

Inputs

Table 6: File Watcher Alteration Parameters

Param	Type	Description
wlist	`table`, `required`	Watch List
maxwait	`numeric`, `otional`, `default-value: 60`	Maximum wait time in seconds
interval	`numeric`, `otional`, `default-value: 0.5`	Verification interval for watcher in seconds
awhat	`string`, `optional`, `default-value: '1'`	Type of file alteration to be observed
nmatch	`number`, `optional`, `default-value: 0`	Number of expected files as a search sufficiency condition

wlist

It is the list of files, directories or file patterns to be observed. The data type is a Lua table and the size of tables is already limited to `2.147.483.647` elements.

An example definition is the following:

```
wlist = {'path/file', 'path', 'pattern*', ...} --arbitrary code
```

maxwait

Maxwait is a numeric value that represents the maximum time to wait for the watcher. Watcher will terminate as soon as possible and as long as the search conditions are met. The default value is `60 seconds`.

interval

Interval is a numerical value that determines how often the watcher checks the search conditions. This value must be less than the maxwait value. The default value is `0.5` seconds.

awhat

Type of file alteration to be observed. See *File Watcher Alteration Parameters*.

Table 7: File Watcher Alteration Parameters

Type	Value	Description
ANY_ALTERATION	'1'	Search for any alteration
CONTENT_ALTERATION	'2'	Search for content file alteration
SIZE_ALTERATION	'3'	Search for file size alteration
CHANGE_TIME_ALTERATION	'4'	Search for file `ctime` alteration
MODIFICATION_TIME_ALTERATION	'5'	Search for file `mtime` alteration
INODE_ALTERATION	'6'	Search for file `inode` alteration
OWNER_ALTERATION	'7'	Search for file `owner` alteration
GROUP_ALTERATION	'8'	Search for file `group` alteration

nmatch

`nmatch` is a number of expected files as a search sufficiency condition.

2.4.10 Watcher for Any Alteration

```
fwa.alteration({'/path/to/file'}, nil, nil, '1')
```

2.4.11 Watcher for Specific Alteration

```
fwa.alteration({'/path/to/file'}, nil, nil, '2') --Watcher for content file alteration
fwa.alteration({'/path/to/file'}, nil, nil, '3') --Watcher for content file size
↪alteration
fwa.alteration({'/path/to/file'}, nil, nil, '4') --Watcher for content file ctime
↪alteration
--explore other options for 'awhat' values
```

See table *File Watcher Alteration Parameters* for more options.

2.4.12 Decoupled Execution

The `create`, `run` function and the `monit` options have been decoupled for better behavior, overhead relief and versatility of use.

2.4.13 Novelty Detection

Watcher implements the detection of the newness of a file based on the `mtime` modification date. This is useful to know if file system items have been created in an expected time window.

> **Warning:** Note that the creation of the files may have been done preserving the attributes of the original file. In that case you should consider the novelty rank accordingly.

```
date_from  = os.time() - 24*60*60 --One day before the current date
date_to    = os.time() + 24*60*60 --One day after the current date
os.execute('touch /tmp/novelty_file.txt') --The file is created on the current date
fwt.creation({'/tmp/novelty_file.txt'}, 10, nil, 0, nil, {date_from, date_to})
```

Note:

For known dates you can use the Lua function **os.time()** as follows:

```
date_from = os.time(
    {
        year = 2020,
        month = 6,
        day = 4,
        hour = 23,
        min = 48,
        sec = 10
    }
)
```

2.4.14 Qualitative Response

Watcher leaves a record for each watchable file where it provides qualitative nformation about the search result for each of them. To explore this information see the *Watcher Monitoring* `match` and `nomatch` functions.

```
NOT_YET_CREATED = '_'                --The file has not yet been created
FILE_PATTERN = 'P'                   --This is a file pattern
HAS_BEEN_CREATED = 'C'               --The file has been created
IS_NOT_NOVELTY = 'N'                 --The file is not an expected novelty
UNSTABLE_SIZE = 'U'                  --The file has an unstable file size
UNEXPECTED_SIZE = 'S'                --The file size is unexpected
DISAPPEARED_UNEXPECTEDLY = 'D'       --The file has disappeared unexpectedly
DELETED = 'X'                        --The file has been deleted
NOT_EXISTS = 'T'                     --The file does not exist
NOT_YET_DELETED = 'E'                --The file has not been deleted yet
NO_ALTERATION = '0'                  --The file has not been modified
ANY_ALTERATION = '1'                 --The file has been modified
CONTENT_ALTERATION = '2'             --The content of the file has been altered
SIZE_ALTERATION = '3'                --The file size has been altered
CHANGE_TIME_ALTERATION = '4'         --The ctime of the file has been altered
MODIFICATION_TIME_ALTERATION = '5'   --The mtime of the file has been altered
INODE_ALTERATION = '6'               --The number of inodes has been altered
OWNER_ALTERATION = '7'               --The owner of the file has changed
GROUP_ALTERATION = '8'               --The group of the file has changed
```

2.4.15 Check File Stability

Enabled only for file creation. This feature ensures that the **watcher** terminates once the file creation is completely finished. This criterion is independent of the file size.

See usage for parameter *stability*

2.4.16 Big Amounts of Files

In the following example, watching the file deletion from the path "/" recursively down to depth level 3 (`levels={0,1,2,3}`) yields a total of **163,170 watchable files**. Note that the execution takes 85 seconds (on a typical desktop machine) but the maximum timeout of the watcher has been specified as low as 10 seconds. This means that 88% of the time is consumed in creating the watcher due to recursion.

```
 tarantool> test=function() local ini=os.time() local fwa=fw.deletion({'/'},␣
↪10, nil, {'NS', nil, 2}, {true, {0,1,2,3}, false}) print(os.time()-ini)␣
↪print(fwa.wid) end
 tarantool> test()
 85
 1620701962375155ULL
 ---
 tarantool> mon.info(1620701962375155ULL)
 ---
 - ans: true
 match: 72
 what: '{"/"}'
 wid: 1620701962375155
```

(continues on next page)

(continued from previous page)

```
  type: FWD
  nomatch: 163098
  status: completed
  ...
```

2.4.17 Atomic Function Injection

Atomic function injection allows you to perform specific tasks on each element of the watchable list separately. In the example, the atomic function afu creates a backup copy for each element of the watchlist.

```
afu = function(file) os.execute('cp '..file..' '..file..'_backup') end --Atomic Funcion
cor = require('watcher').core
wat = cor.create({'/tmp/original.txt'}, 'FWD', afu) --afu is passed as parameter
res = run_watcher(wat)
```

2.4.18 Folder Recursion

You can enable recursion on directories to detect changes in the file system. Recursion is enabled based on a directory entry as a parameter that is considered as a root directory. Starting from this root directory, considered as level zero, you can selectively activate the observation of successive directory levels.

```
fwa.deletion(
    {'/tmp/folder_1'}, --Observed directory is considered a zero level root directory
    nil,               --Maxwait, nil to take the value by omission
    nil,               --Interval, nil to take the value by omission
    nil,               --Options, nil to take the value by omission
    {
        true,          --Activate recursion
        {0, 1, 2},     --Levels of directories to be observed (root and levels 1 & 2)
        false          --Includes hidden files
    }
)
```

For more info see *How do the recursion levels work?*.

2.4.19 Selective Path Level

The recursion levels is a list of numerical values so you can specify (selectively) the directory level you want to observe and ignore others. This is useful in situations where the full path to the file is unknown but the depth or level of the file is known.

```
fwa.deletion(
    {'/bac/invoices'},
    nil,
    nil,
    nil,
    {
        true,          --Activate recursion
        {3},           --Selective level 3
```

(continues on next page)

(continued from previous page)

```
        false          --Includes hidden files
    }
)
```

See use case ...

2.4.20 Watcher Monitoring

`monit` for Watcher monitoring allows you to monitor and explore the running status of a watcher.

info

The output is a Lua table containing the following elements:

- **ans** is a boolean value containing the response of the watcher. `true` means that the watcher has detected the expected changes that are defined in the parameters.
- **match** is the number of cases that match the `true` value of **ans**.
- **nomatch** is the number of cases that do not belong to the set of `true` **ans**.
- **what** is a string containing the obserbables parameter.
- **wid** is the unique identifier of the watcher.
- **type** is the type of the watcher
- **status** is the execution status of the watcher.

```
mon.info(1620701962375155ULL)

  {
      ans: true
      match: 72
      what: '{"/"}'
      wid: 1620701962375155
      type: 'FWD'
      nomatch: 163098
      status: 'completed'
  }
```

match

nomatch