publicator Documentation

Release 0.1.0

Huseyin Yilmaz

Mar 09, 2017

Contents

publish-subscribe server for web.

Contents:

CHAPTER 1

About Publicator

publicator is a scalable publish-subscribe server that publishes data to users of your website. Is can easyly do server-to-client, client-to-client or client-to-server message sending.

CHAPTER 2

Publicator Installation

For now there is no package available for publicator, You have to build it from the source code. (Which is pretty easy though)

How to build a release

To use publicator in a production system, you should package it. Your package will include all dependencies including erlang runtime itself. So you won't need erlang installed on your production system in order to run it. To generate a package first download source code from github.

```
$ git clone git://github.com/huseyinyilmaz/publicator.git
```

Now you can generate a package for your system.

```
$ make configure
$ make
```

publicator.tar.gz will be generated in project root directory (current directory). You can untar it anywhere in your target system and run it with

```
$ bin/publicator start
```

How to run for development

If you want to checkout the code you can configure the system for development.

```
$ make configure
$ make configure-dev
```

This will download development dependencies. After this you can start server in development mode with

```
$ make start
```

You can also run eunit tests with the following.

```
$ make test
```

If you do not want to full compile before starting to test use following

```
$ make eunit
```

To do dialyzer checks first you have to create plt files. Because plt files takes to much times to generate, plt file generation seperated two different parts.

First of all you have to generate plt files for erlang standard library

```

```

CHAPTER 3

Configuring publicator server

Clustering:

Publicator can run as a cluster. Creating a cluster is pretty easy. You can just use '/bin/publicator connect' command to connect to a node. When you connect to a node, You will automatically connect to othernodes in the cluster. So connecting to only one node will be enough.

```
$ bin/publicator node    # return current node name
$ bin/publicator nodes   # return connected node list
$ bin/publicator connect other_publicator_node@127.0.0.1
# Connect to cluster that node named other_publicator_node@127.0.0.1 belongs to.
```

CHAPTER 4

Usage

Running publicator

After building publicator you will have a file named publicator.tar.gz. extract it anywhere you want by running

```
$ tar -xzvf publicator.tar.gz
```

After that you can run several commands.

```
$ bin/publicator start # start server
$ bin/publicator stop # stop server
$ bin/publicator restart # restart serve
$ bin/publicator node # get current node name
$ bin/publicator nodes # get connected node list (for clustering)
$ bin/publicator connect publicator2@127.0.0.1 # connect to given node's cluster
```

Using web interface

Publicator web interface is a one page web app that can connect publicator server and facilitates client capibilities with a web interface. This client can be used for debugging purposes. Publicator-web-interface can be found at https://github.com/huseyinyilmaz/publicator-web-interface

Using publicator javascript client

Publicator consumers can connect to server with javascript client. This client uses websocket to connect to server. It will fallback to http interface if connection problems occur with the connection.

Javascript client can be found at https://github.com/huseyinyilmaz/publicator-js-client

When You get client code from given repository, You must make sure that version number of your client code is same as version number of your publicator server code.

Using publicator python client

Publicator consumers can connect to server with python client. This client uses http interface for connection.

Python client can be found at https://github.com/huseyinyilmaz/publicator-python-client

When You get client code from given repository, You must make sure that version number of your client code is same as version number of your publicator server code.

Using publicator from http interface

If You need to connect publicator from an interface that is not provided. You can roll your own with simple http interface of publicator. First thing you should do to use publictor is to get a session id from the system. You can get your session id from session/

```
$ curl --request POST localhost:8766/session/ --include --data "{\"some_auth_data\": \
↪"bla\"}"
HTTP/1.1 200 OK
connection: keep-alive
server: Cowboy
date: Mon, 24 Feb 2014 09:49:16 GMT
content-length: 42
content-type: application/json; charset=utf-8

{"type":"session_created","data":"session123"}
```

Here, We send a post request to server. Body of the post request is a json that will be send to authentication backend as a "auth_info".

In result, You get a session id "session123" from the system. System will recognize You with this session id and associates your subscribed channels to You.

To subscribe a channel:

```
$ curl --request POST http://localhost:8766/$SESSION_ID/subscribtions/$CHANNEL_CODE/ \
--include \
--header "Content-Type: application/json"

HTTP/1.1 204 No Content
connection: keep-alive
server: Cowboy
date: Sun, 29 Sep 2013 10:29:07 GMT
content-length: 0
content-type: text/html
```

To unscribe a channel:

```
$ curl --request DELETE http://localhost:8766/$SESSION_ID/subscribtions/$CHANNEL_CODE/
↪ \
--include \
--header "Content-Type: application/json"

HTTP/1.1 204 No Content
connection: keep-alive
server: Cowboy
date: Sun, 29 Sep 2013 10:43:00 GMT
```

```
content-length: 0
content-type: text/html
```

To send a message to a channel:

```
$ curl --request POST http://localhost:8766/$SESSION_ID/messages/$CHANNEL_CODE/ \
--include \
--header "Content-Type: application/json" \
--data "message=Message1"

HTTP/1.1 204 No Content
connection: keep-alive
server: Cowboy
date: Sun, 29 Sep 2013 10:47:38 GMT
content-length: 0
content-type: text/html
```

To check for incoming messages that is coming from your subscribed channels:

```
$ curl --request GET http://localhost:8766/$SESSION_ID2/messages/ \
--include \
--header "Content-Type: application/json"

HTTP/1.1 200 OK
connection: keep-alive
server: Cowboy
date: Sun, 29 Sep 2013 10:48:46 GMT
content-length: 25
content-type: text/plain
vary: accept

{"channel1":["Message1"]}
```

Please beware that message publishers do not receive messages they sent. Thats why in this example we are receiving messages from different session id. Format of message url is channel_code to message list mapping. for instance

```
{"channel_name1": ["msg1", "msg2",......],
 "channel_name2": ["msg3", "msg4",......],
 .....
}
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search