Logging using ELK Stack

Tiki has comprehensive built in logging capabilities as part of Action Log and to a lesser extent System Log.

Nevertheless, high volume sites might find these logging capabilities insufficient, or more importantly not performant enough, since these internal logging mechanisms save their data in the MySQL database that is used by Tiki as a whole.

One popular logging solution used by many large sites on the Internet uses the ELK stack, which not only provides a highly performant way to handle log processing, it provides a very flexible way to query and analyze data from the logs themselves. It is log from Tiki to the ELK stack, which would be useful for advanced system administrators managing large sites.

There are already numerous articles explaining how and why to setup logging using the ELK stack available on the Internet, including on Elastic's own website, so the purpose of this documentation is to focus on how to use such a setup with Tiki, rather than repeating the information that is out there.

Available from Tiki 16

Overall architecture

The flow of the logging information is as follows:
Tiki -> HTTP headers -> Apache/other webserver log -> Filebeat (optional) -> Logstash -> Elasticsearch

And then the logs can be queried or analyzed using Kibana and related tools that work with Elasticsearch.

Step 1: make sure you have ELK installed.

You are going to need Elasticsearch, Logstash and Kibana installed. You can install all of these on the same machine or separate machines, or even use a cloud service to get instances. The configurations that are explained here have been tested with:

- Elasticsearch version 2.3.1
- Logstash version 2.3.1

Step 2: setup your Tiki server to log information to the Apache logs

This example is based on Apache 2.x as the web server but you could use similar directives to achieve the same thing in other web servers like NGINX.

In you Apache VirtualHost configuration, switch the log format to the tikicombined log format. Reload

Apache to refresh the configuration.

Note that the index name of the Elasticsearch index you want the logs to be indexed into is specified as "log_yourindexname" in the following example.

```
# CustomLog ${APACHE_LOG_DIR}/access_log combined CustomLog ${APACHE_LOG_DIR}/access_log tikicombined LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" \"%{X-Remote-User}o\" \"%{X-Current-Object}o\" \"%{X-Tiki-Events}o\" %{X-Tiki-Id}o log_yourindexname" tikicombined
```

In the .htaccess file in the Tiki webroot that you want to send logs to the web server log, uncomment the following lines as follows. Specify a TIKI_HEADER_REPORT_ID (must be alphanumeric with underscore, no spaces) to identify that Tiki instance in the logs.

.htaccess

```
# Turning on user and object information to be passed to server logs SetEnv TIKI_HEADER_REPORT_ID my_tiki SetEnv TIKI_HEADER_REPORT_USER on SetEnv TIKI_HEADER_REPORT_OBJECT on SetEnv TIKI_HEADER_REPORT_EVENTS on
```

Step 3: setup Filebeat to send logs to Logstash

Although it is possible to load Apache log files directly into Logstash, this example will use Filebeat (which is part of the ELK stack) to send files over to your Logstash server. The main advantage of using Filebeat is to be able to separate out the CPU intensive part of log processing (Logstash) on another centralized log processing server, and keep merely the "sending the files" part, i.e. Filebeat on the web server itself.

First make sure Filebeat is installed and running on your server. And then make sure it is using a configuration file containing the minimum something like the following:

filebeat.yml

```
filebeat: prospectors: - paths: - /var/log/apache2/access_log input_type: log document_type: tikicombined output: logstash: hosts: ["logstash.yourdomain.com:5043"] # SSL settings to connect to the logstash server if needed tls: # List of root certificates for HTTPS server verifications certificate_authorities: ["/etc/filebeat/logstash_ssl_cert.pem"] # Note this is the full concatenated cert+chain that is for the logstash server shipper: # The name of the shipper that publishes the network data. It can be used to group # all the transactions sent by a single shipper in the web interface. # If this options is not defined, the hostname is used. name: your_site_name
```

You should also setup the logging section of the configuration if you want filebeat errors to be logged on the server.

Step 4: setup Logstash configuration file

On the logstash server, you are going to need a configuration file that processes Tiki log files, and also pick up the log files sent by FileBeat.

tiki.conf

```
input { beats { port => "5043" ssl => true # set to false if ssl if not needed ssl_certificate => "logstash_ssl_cert.pem" ssl_key => "/etc/logstash/logstash_private_key.pem" # note that this key should be kept only on this server and be readable only by the logstash daemon and root. } } filter { # read supported document types set in filebeat if [type] == "tikicombined" { grok { match => { "message" => "%{COMBINEDAPACHELOG} %{DATA:remoteuser} %{QS:currentobject} %{QS:events} %{WORD:tiki_instance} %{WORD:[@metadata][index]}" } } } else { # unsupported document type mutate { add_tag => "drop" } } # flag error logs to be stored in different index if [response] =~ /^5\d\d/ { mutate { add_tag => [ "error", "5xx" ] } } if [response] =~ /^4\d\d/ { mutate { add_tag => [ "error", "4xx" ] } } # index needs to be specified to work if ![@metadata][index] { mutate { add_tag => "drop" } } # get log filename without path grok { match => { "source" => "%{GREEDYDATA}/%{GREEDYDATA:logfilename}" } break_on_match => false } # get query string parameters grok { match => { "request" => "%{URIPARAM:querystring}" } # do not add _grokparsefailure tag if query string is not found tag_on_failure => [] } mutate { gsub => [ "querystring", "\?", "" ] } kv { source => 'querystring' field_split => '&' target => 'queryparams' } # get referrer query string parameters if [referrer] { grok { match => { "referrer" => "%{URIPARAM:referrer_querystring}" } # do not add _grokparsefailure tag if query string is not found tag_on_failure => [] } mutate { gsub => [ "referrer_querystring", "\?", "" ] } kv { source => 'referrer_querystring' field_split => '&' target => 'referrer_queryparams' } } # replace dash value with nothing mutate { gsub => [ "referrer", "\"-\"", "", "remoteuser", "\"-\"", "", "currentobject", "\"-\"", "", "events", "\"-\"", "" ] } # clean up escaped json quotes mutate { gsub => [ "events", "\\\"", '"' ] } # clean up double quotes mutate { gsub => [ "referrer", "^\"", "", "remoteuser", "^\"", "", "currentobject", "^\"", "", "events", "^\"", "" ] } mutate { gsub => [ "referrer", "\"$", "", "remoteuser", "\"$", "", "currentobject", "\"$", "", "events", "\"$", "" ] } # throttle repeated events if desired (performance warning: disables multithreaded pipelining) # throttle { # after_count => 1 # period => 600 # key => "%{agent}%{clientip}%{request}%{remoteuser}%{currentobject}" # add_tag => "throttled" # } if "throttled" in [tags] { mutate { add_tag => "drop" } } else if [tiki_instance] == "" { # drop non Tiki generated logs mutate { add_tag => "drop" } } else { # get Tiki object information mutate { split => { "currentobject" => ":" } } if [currentobject][0] and [currentobject][1] { mutate { add_field => { "object_type" => "%{[currentobject][0]}" } add_field => { "object_id" => "%{[currentobject][1]}" } } } else { mutate { add_field => { "object_type" => "" } add_field => { "object_id" => "" } } } # replace anonymous username if [remoteuser] == "" { mutate { replace => { "remoteuser" => "-anonymous-" } } } # convert events to json json { source => events target => events } # get user location info from ip geoip { source => "clientip" target => "geoip" } # parse useragent info useragent { source => "agent" target => "useragent" } } # drop unneeded if "drop" in [tags] { drop { } } } output { # if it is an error, log to a different index. (if desired) if "error" in [tags] { elasticsearch { #this is an example of authenticated access at Elastic Cloud. Modify as necessary hosts => ["https://whatever.us-east-1.aws.found.io:9243"] user => "logstash" password => "whatever" index => "%{[@metadata][index]}_errors" manage_template => false } } else { elasticsearch { #this is an example of authenticated access at Elastic Cloud. Modify as necessary hosts => ["https://whatever.us-east-1.aws.found.io:9243"] user => "logstash" password => "whatever" index => "%{[@metadata][index]}" manage_template => false } } # stdout { # codec => rubydebug { # metadata => true # } # } # uncomment the stdout for debugging purposes. If logstash is run from the console you should see the output, if run as a service it should log somewhere like /var/log/logstash/logstash.stdout (or similar). }
```

For more information on logstash, check https://www.elastic.co/guide/en/logstash/index.html.

Step 5: Setup the events you wish to log.

All the read events (that would normally be in the Apache log) should already be logged from all the Tiki that are setup as above with a TIKI_HEADER_REPORT_ID set. if you look closely, you will note that the

tiki.conf configuration file drops those with no such parameter set.

In order to log Tiki events in addition to the username, object type, object ID, URL, query string etc which are more standard parameters, you have to configure events to be Logged, similar to how they are configured to be Recorded in an PluginActivityStream.

To setup events to be logged:

1. Go to Control Panels... Community
2. Click on Activity Rules
3. First check out the information contained in the event (if you are not already familiar with that event).
    1. Click on Create... Sample Rule
    2. Select an event to look at from the drop down (e.g. tiki.forumpost.reply)
    3. Provide a description for your own reference (e.g. Forum Post Reply Sample), and then Save
    4. Go and actually reply to a forum post on your site.
    5. On the Activity Rules page, edit the Sample Rule you just created - it will show you all the arguments and values that can be logged for that event, e.g. in this case you will see args.forum_id, args.name, args.parent_title, args.content etc...
4. Then you are ready to create the logging rule
    1. Click on Create.. .Advanced Rule
    2. Select an event to log from the drop down (e.g. tiki.forumpost.reply)
    3. Provide a description for your own reference (e.g. Log forum replies)
    4. Under Rule, enter *(event-log event args)* to log all the information in the event. Save the rule.
    5. If you simply wanted to log certain arguments, you would instead enter *(event-log event args +forum_id&+name&+parent_title)* for example, to include only the arguments args.forum_id, args.name, and args.parent_title.
    6. If you wanted to log all the arguments excepting certain ones you can enter *(event-log event args -content&-index_handled)* for example, to exclude args.content and args.index_handled.

Tips

- If you choose to log an event such as tiki.forumpost.save, it will log all events such as tiki.forumpost.reply, tiki.forumpost.update and tiki.forumpost.create. So you won't need to create 3 event rules, just one.
- Some events contain a lot of data which you might not want to log, as it is makes the log lines in the log files too large, and may also contain information that you don't want in the log files (privacy reasons). For example, for tiki.forumpost.save, you would want to exclude "args.content".
- Activity Rules can be exported or imported as profiles. See Profile Export.