sentinel Documentation

Release 0.1

Piyush Harsh

Nov 29, 2017

Contents:

1 Installation **3**
- 1.1 Download sentinel . . . 3
- 1.2 Using docker-compose . . . 3
- 1.3 Install from source . . . 5

2 Using sentinel APIs **7**
- 2.1 Key concepts . . . 7
- 2.2 API return codes at a glance . . . 7
- 2.3 Header fields at a glance . . . 8
- 2.4 APIs in details . . . 8

3 Sentinel agents **15**
- 3.1 common configuration . . . 15
- 3.2 docker stats agent . . . 16
- 3.3 system stats agent . . . 16
- 3.4 logfile agent . . . 16
- 3.5 inline logger guidelines . . . 16

Sentinel is a framework for monitoring anything anytime. It does not differentiate between logs or metrics, everything is first class element within sentinel. The framework developers provide sentinel agents to cover most common use cases, and a lot many others are being planned for future release.

CHAPTER 1

Installation

Sentinel can be easily installed using docker. The docker compose file is provided for convenience.

1.1 Download sentinel

```
git clone https://github.com/elastest/elastest-platform-monitoring.git
```

Now that you have the source code, you can either use docker to start sentinel, or you can build and package from source.

1.2 Using docker-compose

Change into 'docker-support' subfolder under the root folder of the git repo clone. Then execute docker-compose as shown below:

```
docker-compose up
```

This command brings all the dependencies needed for sentinel:

- Grafana - grafana/grafana:4.6.1
- InfluxDB - influxdb:1.2.4-alpine
- Java8 - rolvlad/alpine-oraclejdk8
- Kafka - spotify/kafka:latest

The sentinel framework allows certain parameters to be set via environment variables. An example environment block is shown next:

```
      - STREAM_ADMINUSER=root
      - STREAM_ADMINPASS=pass1234
      - STREAM_DBENDPOINT=influxdb:8086
```

```
- STREAM_ACCESSURL=localhost:8083
- STREAM_DBTYPE=influxdb
- ZOOKEEPER_ENDPOINT=kafka:2181
- KAFKA_ENDPOINT=kafka:9092
- TOPIC_CHECK_INTERVAL=30000
- INFLUX_URL=http://influxdb:8086
- INFLUX_URL_GRAFANA=http://localhost:8086
- GRAFANA_URL=http://grafana:3000
- GRAFANA_ADMIN=admin
- GRAFANA_PASSWORD=1ccl@b2017
- INFLUX_USER=root
- INFLUX_PASSWORD=pass1234
- SENTINEL_DB_ENDPOINT=/data/sentinel.db
- ADMIN_TOKEN=somevalue
- DASHBOARD_TITLE=elastest
- DASHBOARD_ENDPOINT=localhost:3000
```

Currently, sentinel works only with InfluxDB time-series backend. Support for emerging alternatives such as Timescaledb is planned and will be added very soon.

- STREAM_ADMINUSER - the admin user for InfluxDB
- STREAM_ADMINPASS - choose a secure password for the just declared admin user
- STREAM_DBENDPOINT - the API endpoint of InfluxDB service, typically it is at port 8086
- STREAM_ACCESSURL - the InfluxDB UI URL that sentinel will return back to users, if your service is running on an externally accessible node, change localhost with the FQDN or the IP of the node.
- ZOOKEEPER_ENDPOINT - the endpoint of the Zookeeper service
- KAFKA_ENDPOINT - the endpoint where Kafka cluster is reachable by sentinel
- TOPIC_CHECK_INTERVAL - defined in milliseconds, denotes the time interval between Kafka Topic query by topic manager in Sentinel.

The next 8 variables are for prepopulating Grafana dashboard using the InfluxDB credentials and the **dashboard.json** file located in the source repository.

- INFLUX_URL - address where InfluxDB API service can be reached
- INFLUX_GRAFANA_URL - the InfluxDB URL as would have been entered by an user if s/he was using the admin dashboard
- GRAFANA_URL - address where the Grafana service can be accessed
- GRAFANA_ADMIN - the admin user that should be created
- GRAFANA_PASSWORD - the desired admin account password for the Grafana dashboard
- INFLUX_USER - the user account in InfluxDB to be used in Grafana
- INFLUX_PASSWORD - the account password used in Grafana while setting up the DB account
- SENTINEL_DB_ENDPOINT - the location of the sqlite db file so that preconfigured user, monitoring space and series entries are created for **ElasTest** services to start using sentinel monitoring seamlessly

The last 3 fields are to configuring sentinel admin account and the dashboard rendering by the service.

- DASHBOARD_TITLE - the title for the Grafana embedded dashboard
- DASHBOARD_ENDPOINT - the Grafana service url from which the dashboard is embedded as iframe by sentinel

1.2.1 Configuring Kafka container

The kafka container allows certain parameters to be set via environment block.

```
      - ADVERTISED_PORT=9092
      - ADVERTISED_HOST=kafka
```

Care must be taken in defining **ADVERTISED_HOST** value. The best solution is to provide a FQDN or a public IP if Kafka is to be accessed by external processes which will be the most common use-case of sentinel. Setting an incorrect value of this parameter may leave your kafka cluster unreachable for external services, or even sentinel process running in a container.

Our recommendation is to setup kafka cluster is a separate node entirely, and configure **KAFKA_ENDPOINT** parameter for sentinel as a FQDN string.

1.3 Install from source

Sentinel framework is written in Java and requires Oracle Java 8 for proper working. OpenJDK 8 should also work but the codebase has not been tested with openJDK 8.

1.3.1 Requirements

- Maven 3.0.5 and higher
- Oracle Java 8

1.3.2 Packaging

```
mvn clean package
```

The self-contained jar file is created under ./target/ folder. Unless the pom file was changed, the self contained jar file is named **sentinel-0.1.jar**

To execute, simply run the jar as follows -

```
$ java -jar /path/to/jar/sentinel-0.1.jar
```

The above assumes that the **application.properties** file is in the classpath, or in the same folder as the jar file. In case the **application.properties** file is kept some other location, please use the following command instead -

```
$ java -jar /path/to/jar/sentinel-0.1.jar --spring.config.location=/path/to/config/
↪application.properties
```

1.3.3 Configuration options

All application configuration is provided via **application.properties** file. A sample file content is listed below.

```
spring.thymeleaf.mode=LEGACYHTML5
logging.level.org.springframework.web=WARN
logging.level.org.hibernate=ERROR
logging.level.org.apache.kafka=WARN
logging.level.org.jooq=WARN
```

```
spring.mvc.throw-exception-if-no-handler-found=true
logging.file=sentinel.log
server.port=9000
server.ssl.enabled=true
server.ssl.key-alias=sentinel
server.ssl.key-store=keystore.p12
server.ssl.key-store-type=PKCS12
server.ssl.key-store-password=pass1234
server.ssl.key-password=pass1234
displayexceptions=true
sentinel.db.type=sqlite
sentinel.db.endpoint=sentinel.db
kafka.endpoint=localhost:9092
kafka.key.serializer=StringSerializer
kafka.value.serializer=StringSerializer
zookeeper.endpoint=localhost:2181
topic.check.interval=30000
# stream.db.type=postgres
# stream.db.endpoint=localhost:5432
# stream.db.adminuser=postgres
# stream.db.adminpass=postgres
stream.dbtype=influxdb
stream.dbendpoint=localhost:8086
stream.accessurl=localhost:8083
stream.adminuser=root
stream.adminpass=1ccl@b2017
admin.token=eedsR2v5n4uh7Gjy
series.format.cache.size=100
published.api.version=v1
dashboard.title=elastest
dashboard.endpoint=localhost:3000
```

Many of the entries in the **application.properties** file are self-explanatory. A few non-obvious ones are explained next -

- server.port - on what port number sentinel APIs are accessible
- displayexceptions - set this to **true** if you want to include exceptions full trace in the log outputs
- sentinel.db.type - currently only *sqlite* is supported
- sentinel.db.endpoint - relative or absolute path of the DB file
- topic.check.interval - value in milliseconds indicating the gap between checking list of monitoring spaces for subscription
- stream.dbtype - the time series DB where the monitor stream will be stored, currently only *influxdb* is supported
- stream.accessurl - the url /IP where the InfluxDB admin UI is accessible to the user (if enabled), this should be an externally accessible FQDN ideally
- stream.adminuser - the name of the admin account in the stream DB (here influxdb), this value is meaningful only when *authentication* and *authorization* is enabled in InfluxDB, otherwise the values are not enforced by the DB
- admin.token - this is the master token using which a new user account can be created within sentinel, this value should be accessible only to the administrators of the system, or within the API engine in case you wish to support self registration by general public.
- series.format.cache.size - number of series signatures to be maintained in the in-memory cache of sentinel

CHAPTER 2

Using sentinel APIs

Sentinel monitoring exposes a rich set of APIs for user and space management. The current release of sentinel has APIs supporting bare-minimal features and as the features set get richer, so will be the APIs. Below are the list of APIs currently offered by the framework -

- /v1/api/ - shows list of supported APIs
- /v1/api/user/ - everything to do with user management
- /v1/api/space/ - management of monitoring space
- /v1/api/series/ - management of series with any space
- /v1/api/key/ - API to retrieve user's API key if forgotten
- /v1/api/endpoint - API to retrieve Sentinel's data interface parameters

2.1 Key concepts

Space: Think of it as a collection of metrics belonging to different streams but somehow belonging to the same scope, application or service. A space could be allocated to metrics of smaller services making up a larger application or service.

Series: A series in Sentinel is a stream of metrics coming from the same source.

2.2 API return codes at a glance

API endpoint	Verb	Return codes	Comments
/v1/api/	GET	200	ok
		500	service down
/v1/api/user/	POST	201	created

Continued on next page

Table 2.1 – continued from previous page

API endpoint	Verb	Return codes	Comments
		400	check data
		401	valid admin token needed
		409	user account already exists
		500	system error
/v1/api/user/{id}	GET	200	ok
		401	unauthorized
		400	check data
/v1/api/space/	POST	201	created
		400	check data
		401	invalid api key
		409	space already exists for user
		500	system error
/v1/api/series/	POST	201	created
		400	check data
		401	invalid api key
		409	series already exists for user
		500	system error
/v1/api/key/{id}	GET	200	ok
		400	no such user exist
		401	invalid password
/v1/api/endpoint	GET	200	ok
		401	invalid api key
/v1/dashboard/	GET	200	ok
		500	system error
/v1/api/pingback/	POST	201	created
		400	check data
		401	invalid api key
		500	system error
/v1/api/pingback/{id}	GET	200	ok
		401	invalid api key

2.3 Header fields at a glance

field key	value / interpretations
Content-Type	application/json is typical
x-auth-token	admin user master token
x-auth-password	password associated with user
x-auth-login	username or userid
x-auth-apikey	api key associated with user

2.4 APIs in details

Now that we have all the basic building buildings in place, lets explore each API endpoint in more details. In the following subsections lets assume that the sentinel API service is available at https://localhost:9000/. Also API example will be provided as a valid cURL command.

2.4.1 /v1/api/ GET

This API allows a quick check on the health status, if the service is alive a 200 status code is returned along with a list of supported API endpoints.

```
curl -X GET https://localhost:9000/v1/api/
```

The response is similar to one shown below -

```
[
 {
   "endpoint": "/v1/api/",
   "method": "GET",
   "description": "get list of all supported APIs",
   "contentType": "application/json"
 },
 {
   "endpoint": "/v1/api/user/",
   "method": "POST",
   "description": "add a new user ",
   "contentType": "application/json"
 },
 {
   "endpoint": "/v1/api/user/{id}",
   "method": "GET",
   "description": "retrieve info about existing user",
   "contentType": "application/json"
 },
 {
   "endpoint": "/v1/api/space/",
   "method": "POST",
   "description": "register a new monitored space",
   "contentType": "application/json"
 },
 {
   "endpoint": "/v1/api/series/",
   "method": "POST",
   "description": "register a new series within a space",
   "contentType": "application/json"
 },
 {
   "endpoint": "/v1/api/key/{id}",
   "method": "GET",
   "description": "retrieve the api-key for an user",
   "contentType": "application/json"
 },
 {
   "endpoint": "/v1/api/endpoint",
   "method": "GET",
   "description": "retrieve the agent's connection endpoint parameters",
   "contentType": "application/json"
 },
 {
      "endpoint": "/v1/api/pingback/{id}",
      "method": "GET",
      "description": "retrieve the healthcheck/pingback object data",
      "contentType": "application/json"
  },
```

```
  {
      "endpoint": "/v1/api/pingback/",
      "method": "POST",
      "description": "registers a new healthcheck/pingback object",
      "contentType": "application/json"
  }
]
```

The output above is representative, and the actual API supported by sentinel varied during the time of writing of this document.

2.4.2 /v1/api/user/ POST

Use this API to create a new user of sentinel. User account creation is an admin priviledged operation and the *admin-token* is required as header for the call to be executed successfully.

```
curl -X POST https://localhost:9000/v1/api/user/ --header "Content-Type: application/
↪json"
--header "x-auth-token: <admin-token>" -d '{"login":"username", "password":"some-
↪password"}'
```

If the user already exists, you will get a *409 Conflict* status response back. An example response upon successful creation of an account looks as shown below, the actual value is for representation purposes only -

```
{
  "login": "username",
  "apiKey": "b6af63b9-f699-4259-8548-2a60e0d88661",
  "id": 2,
  "accessUrl": "/api/user/2"
}
```

The *apiKey* and *id* values should be saved as they are needed in some of the management API requests as you will see later.

2.4.3 /v1/api/user/{id} GET

Use this API to retrieve the complete information about an user account, the monitoring spaces and series info included. A valid *api-key* needs to be provided as a header field while making this call.

```
curl -X GET https://localhost:9000/v1/api/user/{id} --header "Content-Type:␣
↪application/json"
--header "x-auth-apikey: valid-api-key"
```

If the call succeeds then the complete details of the account is returned back. A sample value returned is shown next.

```
{
  "apiKey": "f3549958-8884-4649-9661-8ca338dfe141",
  "id": 1,
  "accessUrl": "/api/user/1",
  "spaces": [
      {
          "id": 1,
          "accessUrl": "/api/space/1",
          "topicName": "user-1-cyclops",
          "name": "cyclops",
```

```
            "seriesList": [
                {
                    "id": 1,
                    "accessUrl": "/api/series/1",
                    "name": "app-logs",
                    "msgFormat": "unixtime:s msgtype:json"
                }
            ],
            "dataDashboardUrl": "http://localhost:8083/",
            "dataDashboardUser": "user1cyclops",
            "dataDashboardPassword": "qkDaFQ8gJEokApS6"
        }
    ]
}
```

2.4.4 /v1/api/space/ POST

Use this API to create a new monitored space for a given user account in sentinel. A matching *username* and the *api-key* needs to be provided as header fields. The body just contains the *name* of the space that one wishes to create.

```
curl -X POST https://localhost:9000/v1/api/space/ --header "Content-Type: application/
↪json"
--header "x-auth-login: username" --header "x-auth-apikey: some-api-key"
-d '{"name":"space-name"}'
```

If the call is successful, the *space id* is returned back as confirmation. A sample response is shown next.

```
{
  "id": 3,
  "accessUrl": "/api/space/3",
  "topicName": "user-1-new-space",
  "name": "new-space",
  "dataDashboardUrl": "http://localhost:8083/",
  "dataDashboardUser": "user1new-space",
  "dataDashboardPassword": "GeMHPDUwKc5621ZI"
}
```

2.4.5 /v1/api/series/ POST

A space by itself does not handle data streams, it is a container and needs a series to be defined before the metrics sent to it can be persisted and analyzed later. This API allows creation of a *series* within an existing *space*. The *msgSignature* allows sentinel to parse the incoming messages properly.

If the message being sent into sentinel is a single level JSON string, the *unixtime:s msgtype:json* value is sufficient.

```
curl -X POST https://localhost:9000/v1/api/series/ --header "Content-Type:␣
↪application/json"
--header "x-auth-login: username" --header "x-auth-apikey: some-api-key"
-d '{"name":"series-name", "spaceName":"parent-space-name", "msgSignature":"msg-
↪signature"}'
```

If the call is successful, a *series id* is returned. An example response block is shown for completeness.

```
{
  "id": 2,
  "accessUrl": "/api/series/2",
  "name": "some-app-logs"
}
```

2.4.6 /v1/api/key/{id} GET

One can use this API if there is a need to retrieve the user api-key. The *username* should be a registered account and the *some-password* header field should be the matching password for this account.

```
curl -X GET https://localhost:9000/v1/api/key/{username}
--header "Content-Type: application/json"
--header "x-auth-password: some-password"
```

If the call is successful, the API-key is returned. A sample response is shown next.

```
{
  "apiKey": "f3549958-8884-4649-9661-8ca338dfe141",
  "id": 1,
  "accessUrl": "/api/user/1"
}
```

2.4.7 /v1/api/endpoint GET

This API call can be used to retrieve the connection parameters for the sentinel agents to send data streams to. The call is available only to registered accounts, therefore a valid *username* and *api-key* needs to be supplied as header fields.

```
curl -X GET https://localhost:9000/v1/api/endpoint --header "Content-Type:␣
↪application/json"
--header "x-auth-login: username" --header "x-auth-apikey: some-api-key"
```

If the call succeeds, the parameter block is returned that can be used to properly configure the sentinel agents. A sample response is shown next.

```
{
  "endpoint": "kafka:9092",
  "keySerializer": "StringSerializer",
  "valueSerializer": "StringSerializer"
}
```

2.4.8 /v1/dashboard/ GET

This API call can be used to get the Grafana dashboard as an embedded iFrame from the sentinel framework. One may need to login into the dashboard using the Grafana account credentials. This API call is not authenticated. Authentication is enforced by Grafana.

If the call succeeds, an HTML codeblock is returned which renders the dashboard. A sample response is shown next.

```
<html><head></head><body style="background-color:black;"><script>
function resizeIframe(obj) {
  obj.style.height = obj.contentWindow.document.body.scrollHeight + 'px';
```

```
}
</script><div style="width:100%; background-color:black;"><img src="http://elastest.
↪io/images/intense/elastest-logo-dark.png" align="left"></div><br><br><br><br>
↪<iframe onload="resizeIframe(this)" width="99%" height="90%" style="border:none;␣
↪display:block;" src="http://localhost:3000/dashboard/db/elastest?refresh=30s&
↪orgId=1&theme=light"></iframe></body></html>
```

2.4.9 /v1/api/pingback/ POST

This call is used to register a new pingback entry with sentinel. Think of it as a health check service. It allows one to register an endpoint to be monitored, and if the check fails then a callback to be made to another endpoint. This feature only supports GET calls. The call is available only to registered accounts, therefore a valid *username* and *api-key* needs to be supplied as header fields.

```
curl -X POST http://localhost:9000/v1/api/pingback/ \
-H 'x-auth-apikey: some-api-key' -H 'x-auth-login: username' \
-d '{
  "pingURL":"some-service-endpoint",
  "reportURL":"some-reporting-endpoint",
  "periodicity":30000,
  "toleranceFactor":2,
  "method":"body,status,up"
}'
```

In the call above, **periodicity** is specified as multiples of 30 seconds in nanoseconds, **toleranceFactor** is the number of successive failed checks that will trigger a callback to the **reportURL** endpoint. Currently only two **methods** are supported -

- code - the HTTP response code is the sole criteria for success or failure
- body,{field},{intended value} - in this mode, the response body is treated

as a JSON value, and **{field}** specifies the JSON field to be checked for the value specified by **{intended value}** field. If the defined value is not found, it is treated as a failure condition.

Note: since comma (,) is used as a separator character, **{field}** and **{intented value}** parameters must not contain a comma (,) in them.

If the call succeeds, the details of the registered pingback endpoint is returned. An example response is shown next -

```
{
  "id": 1,
  "pingURL": "some-service-endpoint",
  "reportURL": "some-reporting-endpoint",
  "periodicity": 30000,
  "toleranceFactor": 2,
  "method": "body,status,up",
  "accessUrl": "/api/pingback/1"
}
```

2.4.10 /v1/api/pingback/{id} GET

This call allows the user to retrieve the details of a particular pickback object. The call is available only to registered accounts, therefore a valid *username* and *api-key* needs to be supplied as header fields.

```
curl -X GET http://localhost:9000/v1/api/pingback/1 \
-H 'x-auth-apikey: some-api-key' -H 'x-auth-login: some-username'
```

If the API call is successful, the details of the pingback is returned along with last 10 healthcheck results along with the timestamp. An example response is shown next -

```
{
  "id": 1,
  "pingURL": "some-service-endpoint",
  "reportURL": "some-reporting-endpoint",
  "periodicity": 30000,
  "toleranceFactor": 2,
  "method": "body,status,up",
  "callHistory": [
      {
          "eventTime": 1511866036619,
          "status": "NOK"
      },
      {
          "eventTime": 1511866018280,
          "status": "OK"
      }
  ],
  "accessUrl": "/api/pingback/1"
}
```

CHAPTER 3

Sentinel agents

Currently three sentinel agents are available and more are being planned and will be released in the near future.

- docker stats agent
- system stats agent
- logfile agent
- inline logging guidelines (for self instrumented code)

In addition, it is very easy to use inline code to directly send your logs into sentinel. All agents are written in **Python3** and need **pip3** to install all dependencies.

3.1 common configuration

The agent configuration file is called **sentinel-agent.conf**. Depending on the type of agent you are executing / configuring, the configuration sections may be different. Here we present the common sections present in every configuration file:

```
[kafka-endpoint]
endpoint = kafka-endpoint:9092
keySerializer = StringSerializer
valueSerializer = StringSerializer
```

- endpoint - set this to the end point of your kafka cluster associated with sentinel framework

```
[sentinel]
topic = some-topic-name
seriesName = your-series-name
```

- topic - the kafka topic-name that was allocated to sentinel user for a given monitoring space in sentinel
- seriesname - the series that was created within the monitoring space

Unless one is writing their own sentinel agent, there is probably no need to change any other configuration parameters under **[kafka-endpoint]** or **[sentinel]** sections. Agent specific configuration parameters are covered in each agents subsection next.

3.2 docker stats agent

The agent is located in the *sentinel-agents/dockerstats/* subdirectory in the downloaded git repository. To install all dependencies please use -

```
# pip3 install -f requirements.txt
```

The agent can simply be executed via this command -

```
$ python3 sentinel-docker-agent.py
```

3.3 system stats agent

The agent is located in the *sentinel-agents/systemstats/* subdirectory in the downloaded git repository. To install all dependencies please use -

```
# pip3 install -f requirements.txt
```

The agent can simply be executed via this command -

```
$ python3 sentinel-sys-agent.py
```

3.4 logfile agent

The agent is located in the *sentinel-agents/logparsing/* subdirectory in the downloaded git repository. To install all dependencies please use -

```
# pip3 install -f requirements.txt
```

The agent can simply be executed via this command -

```
$ python3 sentinel-log-agent.py
```

3.5 inline logger guidelines

In case you wish to send application metrics and logs explicitly and directly to sentinel, you should basically marshall your data point as JSON value with the following structure -

```
{
    "agent":"sentinel-internal-log-agent",
    "level":"log-level such as error/debug/info/warn",
    "msg":"log message",
    ...
}
```

The requirement is that you should send a flat JSON data as shown above. **agent** field is necessary and it's value MUST be set to **sentinel-internal-log-agent**, rest all JSON fields can be flexibly named and set as per your application needs.

This data point and subsequent points should be sent using any kafka client with kafka-topic and key set to sentinel topic and series that you should have defined already. Use **string serialization** to send data into kafka endpoint of sentinel.

It is important that the series *signature* be set to **unixtime:s msgtype:json** so that sentinel knows how to properly parse the incoming data stream.