Alignak Web Services Documentation

Release 0.1.1

Frédéric MOHIER

Sep 10, 2018

Contents

Contents:

CHAPTER 1

Introduction

This module for Alignak exposes some Alignak Web Services:

- *GET /* will return the list of the available endpoints
- *GET /alignak_map* that will return the map and status of all the Alignak running daemons
- *POST /alignak_command* that will notify an external command to the Alignak framework
- *PATCH /host/<host_name>* that allows to send live state for an host and its services, update host custom variables, enable/disable host checks

CHAPTER 2

Installation

2.1 Requirements

To use this module, you first need to install some Python modules that are listed in the `requirements.txt` file:

```
# Needing six for python 2.7/3 compatibility
six

# Nothing special except:
requests
alignak_backend_client

# Same version as the one used by Alignak
CherryPy==15.0.0
```

Note: if you proceed to an end-user installation with pip, the required modules are automatically installed.

2.2 Installation with PIP

Note that the recommended way for installing on a production server is mostly often to use the packages existing for your distribution. Nevertheless, the pip installation provides a startup script using an uwsgi server and, for FreeBSD users, rc.d scripts.

2.2.1 End user installation

You can install with pip:

```
sudo pip install alignak-module-ws
```

The required Python modules are automatically installed if not they are not yet present on your system.

2.2.2 From source

You can install it from source:

```
git clone https://github.com/Alignak-monitoring/alignak-module-ws
cd alignak-module-ws
pip install .
```

CHAPTER 3

Configuration

On installation, this module ships its own configuration file in the */usr/local/etc/alignak* directory. The default configuration file is *alignak.module-ws.ini*. This file is commented to help configure all the parameters.

To configure an Alignak daemon (*receiver* is the recommended one) to use this module:

- edit your Alignak configuration file (eg. *alignak.ini*)
- declare the module alias (defaults to *web-services*) in the *modules* parameter of the daemon
- add the example module configuration file content into the Alignak configuration file.

Note that currently the SSL part of this module has not yet been tested!

3.1 Alignak backend

The Alignak backend configuration part requires to set the Alignak backend endpoint and some login information. The login information are not mandatory because the module will use the credentials provided by the Web Service client when one will request on an endpoint with some credentials.

3.2 Alignak arbiter

The Alignak arbiter configuration part is not mandatory. It will only be used by the module to get the Alignak daemons states to populate the */alignak_map* endpoint. Thus, you should only configure this part if you intend to use this endpoint to get some information.

The default `mod-ws.cfg` file:

The default `alignak.module-ws.ini` file:

CHAPTER 4

Services

The application runs without any extra configuration file with its default parameters. Nevertheless, the application is best used when suited to user's neeeds ;)

4.1 HTTP authorization

As a default, all the WS endpoints require the client to provide some credentials. You can provide those credentials directly in the HTTP Authorization header or you can use the */login* and */logout* endpoints to create a WS session.

To provide the credentials you can use the token delivered by the Alignak backend when you are logging-in. If you do not have this information, you can provide your `username` and `password` to authenticate near the Alignak backend.

```
$ curl -H "Content-Type: application/json" -X POST -d '{"username":"admin","password":
↪"admin"}' http://127.0.0.1:8888/login
{'_status': 'OK', '_result': ["1442583814636-bed32565-2ff7-4023-87fb-34a3ac93d34c"]}
```

Logging out will clear the session on the server side.

```
$ curl -H "Content-Type: application/json" -X GET http://127.0.0.1:8888/logout
```

Example 1 (direct credentials provided):

```
$ curl -X GET -H "Content-Type: application/json" --user "1442583814636-bed32565-2ff7-
↪4023-87fb-34a3ac93d34c:" http://127.0.0.1:8888/alignak_logs
```

Example 2 (login session):

```
$ curl -H "Content-Type: application/json" -X POST -d '{"username":"admin","password":
↪"admin"}' http://127.0.0.1:8888/login
{'_status': 'OK', '_result': ["1442583814636-bed32565-2ff7-4023-87fb-34a3ac93d34c"]}

$ curl -X GET -H "Content-Type: application/json" --user "1442583814636-bed32565-2ff7-
↪4023-87fb-34a3ac93d34c:" http://127.0.0.1:8888/alignak_logs
```

Note that using the login / logout session is an easy thing with a python library like requests with its session mechanism ;) Or with any client that handles sessions ...

4.2 Get Alignak state

To get Alignak daemons states, GET on the *alignak_map* endpoint:

```
$ wget http://demo.alignak.net:8888/alignak_map

$ cat alignak_map
{

    "reactionner": {
        .../...
    },
    "broker": {
        .../...
    },
    "arbiter": {
        "arbiter-master": {
            "passive": false,
            "polling_interval": 1,
            "alive": true,
            "realm_name": "",
            "manage_sub_realms": false,
            "is_sent": false,
            "spare": false,
            "check_interval": 60,
            "address": "127.0.0.1",
            "manage_arbiters": false,
            "reachable": true,
            "max_check_attempts": 3,
            "last_check": 0,
            "port": 7770
        }
    },
    "scheduler": {
        .../...
    },
    "receiver": {
        .../...
    },
    "poller": {
        .../...
    }

}
```

The state of the all the Alignak running daemons is returned in a JSON object formatted as the former example. each daemon type contains an object for each daemon instance with the daemon configuration and live state.

4.3 Get Alignak history

To get Alignak history, GET on the *alignak_logs* endpoint:

```
$ wget http://demo.alignak.net:8888/alignak_logs

$ cat alignak_logs
{
    "_status": "OK",
    "items": [
        {
            "service_name": "Zombies",
            "host_name": "chazay",
            "user_name": "Alignak",
            "_created": "Sun, 12 Mar 2017 19:14:48 GMT",
            "message": "",
            "type": "check.result"
        },
        {
            "service_name": "Users",
            "host_name": "denice",
            "user_name": "Alignak",
            "_created": "Sun, 12 Mar 2017 19:14:40 GMT",
            "message": "",
            "type": "check.result"
        },
        {
            "service_name": "Zombies",
            "host_name": "alignak_glpi",
            "user_name": "Alignak",
            "_created": "Sun, 12 Mar 2017 19:14:37 GMT",
            "message": "",
            "type": "check.result"
        },
        {
            "service_name": "Processus",
            "host_name": "lachassagne",
            "user_name": "Alignak",
            "_created": "Sun, 12 Mar 2017 19:14:18 GMT",
            "message": "",
            "type": "check.result"
        },
        .../...
    ]
}
```

The result is a JSON object containing a *_status* property that should be 'OK' and an *items* array property that contain the 25 most recent history events stored in the backend. Each item in this array has the properties:

- _created: GMT date of the event creation in the backend
- host_name / service_name
- user_name: Alignak for Alignak self-generated events, else web UI user that provoked the event
- message: for an Alignak check result, this will contain the main check result information: state[state_type] (acknowledged/downtimed): output (eg. UP[HARD] (False/False): Check output)
- **type is the event type:** # WebUI user comment "webui.comment",

Check result “check.result”,

Request to force a check (from WebUI) “check.request”, “check.requested”,

Add acknowledge (from WebUI) “ack.add”, # Set acknowledge “ack.processed”, # Delete acknowledge “ack.delete”,

Add downtime (from WebUI) “downtime.add”, # Set downtime “downtime.processed”, # Delete downtime “downtime.delete”

timeperiod transition “monitoring.timeperiod_transition”, # alert “monitoring.alert”, # event handler “monitoring.event_handler”, # flapping start / stop “monitoring.flapping_start”, “monitoring.flapping_stop”, # downtime start / cancel / end “monitoring.downtime_start”, “monitoring.downtime_cancelled”, “monitoring.downtime_end”, # acknowledge “monitoring.acknowledge”, # notification “monitoring.notification”,

Some parameters can be used to refine the results:

- count: number of elements to get (default=25). According to the Alignak backend pagination, the maximu number of elements taht can be returned is 50.
- page: page number (default=0). With the default count (25 items), page=0 returns the items from 0 to 24, page=1 returns the items from 25 to 49, . . .
- **search: search criteria in the items fields. The search criteria is using the same search engin as the one implemented in the**
  host_name:pattern, search for pattern in the host_name field (pattern can be a regex) *service_name:pattern*, search for pattern in the host_name field (pattern can be a regex) *user_name:pattern*, search for pattern in the host_name field (pattern can be a regex)

  type:monitoring-alert, search for all events that have the *monitoring.alert* type

  several search criterias can be used simultaneously. Simply separate them with a space character:
  host_name:pattern type:monitoring-alert‘

  (To be completed. . .)

Note that the returned items are always sorted to get the most recent first

4.4 Get host data

To get an Alignak host data, GET on the *host* endpoint:

```
 $ curl --request GET \
   --url http://demo.alignak.net:8888/host \
   --header 'authorization: Basic␣
↪MTQ4NDU1ODM2NjkyMi1iY2Y3Y2NmMS03MjM4LTQ4N2ItYWJkOS0zMGNlZDdlNDI2ZmI6' \
   --header 'cache-control: no-cache' \
   --header 'content-type: application/json' \
   --data '
   {
     "name": "passive-01",
   }'

 OR:
 $ curl --request GET \
   --url http://demo.alignak.net:8888/host/passive-01 \
   --header 'authorization: Basic␣
↪MTQ4NDU1ODM2NjkyMi1iY2Y3Y2NmMS03MjM4LTQ4N2ItYWJkOS0zMGNlZDdlNDI2ZmI6' \
   --header 'cache-control: no-cache' \
   --header 'content-type: application/json'


# JSON result
```

```
{"_status": "OK", "_result": [{"ls_grafana": true, "business_impact_modulations": [],
↪"_template_fields": [], "action_url": "", "low_flap_threshold": 25, "process_perf_
↪data": true, "icon_image": "", "ls_last_time_down": 1506319207, "_realm": {"_updated
↪": "Fri, 22 Sep 2017 10:20:52 GMT", "_level": 1, "_id": "59c4e40435d17b8e0c6acc61",
↪"global_warning_threshold": 3, "name": "North", "definition_order": 100, "_children
↪": [], "default": false, "notes": "", "hosts_critical_threshold": 5, "_all_children
↪": [], "_parent": "59c4e38535d17b8dcb0bed42", "alias": "North country", "services_
↪warning_threshold": 3, "global_critical_threshold": 5, "_created": "Fri, 22 Sep␣
↪2017 10:20:52 GMT", "hosts_warning_threshold": 3, "_tree_parents": [
↪"59c4e38535d17b8dcb0bed42"], "services_critical_threshold": 5, "_etag":
↪"8d97a350faef5d2da957c3adf00ff7bb04d96e57", "imported_from": "alignak-backend-import
↪"}, "display_name": "", "notification_interval": 1440, "ls_execution_time": 0.0,
↪"retry_interval": 0, "snapshot_enabled": false, "event_handler_enabled": false, "3d_
↪coords": "", "parents": [], "location": {"type": "Point", "coordinates": [48.858293,
↪ 2.294601]}, "labels": [], "snapshot_period": {"_updated": "Fri, 22 Sep 2017␣
↪10:18:45 GMT", "name": "Never", "definition_order": 100, "_sub_realm": true, "notes
↪": "", "is_active": true, "dateranges": [], "alias": "No time is a good time",
↪"imported_from": "unknown", "exclude": [], "_created": "Fri, 22 Sep 2017 10:18:45␣
↪GMT", "_id": "59c4e38535d17b8dcb0bed47", "_etag":
↪"6dad1d7c31fbaaa87188dc08ed3b2cb4c63c0077", "_realm": "59c4e38535d17b8dcb0bed42"},
↪"notifications_enabled": true, "address6": "", "freshness_threshold": 14400, "alias
↪": "Passive host 1", "time_to_orphanage": 300, "name": "passive-01", "notes": "",
↪"ls_last_notification": 0, "custom_views": [], "active_checks_enabled": false, "_
↪templates": [{"ls_grafana": false, "business_impact_modulations": [], "labels": [],
↪"action_url": "", "low_flap_threshold": 25, "process_perf_data": true, "business_
↪rule_downtime_as_ack": false, "ls_last_time_down": 0, "_realm":
↪"59c4e40435d17b8e0c6acc61", "display_name": "", "notification_interval": 1440, "ls_
↪execution_time": 0.0, "retry_interval": 0, "snapshot_enabled": false, "event_
↪handler_enabled": false, "3d_coords": "", "parents": [], "ls_acknowledged": false,
↪"_template_fields": [], "snapshot_period": "59c4e38535d17b8dcb0bed47",
↪"notifications_enabled": true, "address6": "", "freshness_threshold": 14400, "time_
↪to_orphanage": 300, "name": "north-host", "notes": "", "ls_last_notification": 0,
↪"high_flap_threshold": 50, "custom_views": [], "active_checks_enabled": false, "_
↪templates": ["59c4e40535d17b8e0c6acca9"], "service_includes": [], "reactionner_tag
↪": "", "notes_url": "", "ls_last_state": "UNREACHABLE", "ls_last_time_down": 0,
↪"usergroups": ["59c4e40535d17b8e0c6acca4"], "notification_period":
↪"59c4e40435d17b8e096acc62", "resultmodulations": [], "icon_image": "", "stalking_
↪options": [], "_sub_realm": true, "ls_long_output": "", "macromodulations": [], "ls_
↪state_id": 3, "business_rule_host_notification_options": ["d", "u", "r", "f", "s"],
```

(continues on next page)

(continued from previous page)

The result is a JSON object containing a *_status* property that should be 'OK' and a *_result* property that contain the list of the hosts fetched from the backend. Each item in this array has the properties defined in the Alignak backend for an host and it also contains the related objects (e.g. *check_command*, ...) as thery are also defined in the backend.

4.5 Hosts / services state report

4.5.1 Host/service livestate

To send an host/service live state, PATCH on the *host* endpoint providing the host name and its state:

```
$ curl --request PATCH \
  --url http://demo.alignak.net:8888/host \
  --header 'authorization: Basic␣
↪MTQ4NDU1ODM2NjkyMi1iY2Y3Y2NmMS03MjM4LTQ4N2ItYWJkOS0zMGNlZDdlNDI2ZmI6' \
  --header 'cache-control: no-cache' \
  --header 'content-type: application/json' \
  --data '
  {
    "name": "passive-01",
    "variables": {
        "test": "test"
    },
    "active_checks_enabled": false,
    "passive_checks_enabled": true,
    "livestate": {
        "state": "UP",
        "output": "WS output - active checks disabled"
    },
    "services": {
        "first": {
            "name": "dev_BarcodeReader",
            "active_checks_enabled": false,
            "passive_checks_enabled": true,
            "livestate": {
                "state": "OK",
                "output": "WS output - I am ok!"
            }
        }
    }
}'

# JSON result
{
  "_status": "OK",
  "_result": [
    "passive-01 is alive :)",
    "[1491368659] PROCESS_HOST_CHECK_RESULT;passive-01;0;WS output - active checks␣
↪disabled",
    "[1491368659] PROCESS_SERVICE_CHECK_RESULT;passive-01;dev_BarcodeReader;0;WS␣
↪output - I am ok!",
    "Service 'passive-01/dev_BarcodeReader' unchanged.",
    "Host 'passive-01' unchanged."
  ],
  "_feedback": {
    "passive_checks_enabled": true,
    "active_checks_enabled": false,
    "alias": "Passive host 1",
    "freshness_state": "d",
    "notes": "",
    "retry_interval": 0,
    "_overall_state_id": 4,
```

(continues on next page)

(continued from previous page)

```
    "freshness_threshold": 14400,
    "location": {
      "type": "Point",
      "coordinates": [
        46.60611,
        1.87528
      ]
    },
    "check_interval": 5,
    "services": {
      "first": {
        "active_checks_enabled": false,
        "freshness_threshold": 43200,
        "_overall_state_id": 1,
        "freshness_state": "x",
        "notes": "",
        "retry_interval": 0,
        "alias": "Barcode reader",
        "passive_checks_enabled": true,
        "check_interval": 0,
        "max_check_attempts": 1,
        "check_freshness": true
      }
    },
    "max_check_attempts": 1,
    "check_freshness": true
  }
}
```

The result is a JSON object containing a *_status* property that should be 'OK' and a *_result* array property that contains information about the actions that were executed. A *_feedback* dictionary property provides some informatyion about the host/service.

If an error is detected, the *_status* property is not 'OK' and a *_issues* array property will report the detected error(s).

The */host/host_name* can be used to target the host. If a *name* property is present in the JSON data then this property will take precedence over the *host_name* in the endpoint.

For the host services states, use the same syntax as for an host:

```
$ curl -X PATCH -H "Content-Type: application/json" -d '{
    "name": "test_host",
    "livestate": {
        "state": "up",
        "output": "Output...",
        "long_output": "Long output...",
        "perf_data": "'counter'=1"
    },
    "services": {
        "test_service": {
            "name": "test_service",
            "livestate": {
                "state": "ok",
                "output": "Output...",
                "long_output": "Long output...",
                "perf_data": "'counter'=1"
            }
```

(continues on next page)

(continued from previous page)

```
        },
        "test_service2": {
            "name": "test_service2",
            "livestate": {
                "state": "warning",
                "output": "Output...",
                "long_output": "Long output...",
                "perf_data": "'counter'=2"
            }
        },
        "test_service3": {
            "name": "test_service3",
            "livestate": {
                "state": "critical",
                "output": "Output...",
                "long_output": "Long output...",
                "perf_data": "'counter'=3"
            }
        },
    }
}' "http://demo.alignak.net:8888/host"
```

The livestate data for an host or service may contain: - *state*: “ok”,”warning”,”critical”,”unknown”,”unreachable” for a service. “up”,”down”,”unreachable” for an host. - *output*: the host/service check output - *long_output*: the host/service long output (second part of the output) - *perf_data*: the host/service check performance data - *timestamp*: timestamp for the host/service check

Note that the *livestate* for the host or for any service may be an array if more than one result is to be reported to the Web Service.

4.5.2 Host custom variables

To create/update host custom variables, PATCH on the *host* endpoint providing the host name and its variables:

```
$ curl -X PATCH -H "Content-Type: application/json" -d '{
    "name": "test_host",
    "variables": {
        "test1": "string",
        "test2": 12,
        "test3": 15055.0,
        "test4": "new!"
    }
}' "http://demo.alignak.net:8888/host"
```

The result is a JSON object containing a *_status* property that should be ‘OK’ and an *_result* array property that contains information about the actions that were executed.

If an error is detected, the *_status* property is not ‘OK’ and a *_issues* array property will report the detected error(s).

The */host/host_name* can be used to target the host. If a *name* property is present in the JSON data then this property will take precedence over the *host_name* in the endpoint.

4.5.3 Host enable/disable checks

To enable/disable hosts/services checks, PATCH on the *host* endpoint providing the host (service) name and its checks configuration:

```
$ curl -X PATCH -H "Content-Type: application/json" -d '{
    "name": "test_host",
    "active_checks_enabled": True,
    "passive_checks_enabled": True,
    "services": {
        "test_service": {
            "name": "test_ok_0",
            "active_checks_enabled": True,
            "passive_checks_enabled": True,
        },
        "test_service2": {
            "name": "test_ok_1",
            "active_checks_enabled": False,
            "passive_checks_enabled": False,
        },
        "test_service3": {
            "name": "test_ok_2",
            "active_checks_enabled": True,
            "passive_checks_enabled": False,
        },
    }
}' "http://demo.alignak.net:8888/host"
```

The result is a JSON object containing a *_status* property that should be 'OK' and an *_result* array property that contains information about the actions that were executed.

If an error is detected, the *_status* property is not 'OK' and a *_issues* array property will report the detected error(s).

The */host/host_name* can be used to target the host. If a *name* property is present in the JSON data then this property will take precedence over the *host_name* in the endpoint.

4.5.4 Host/service creation

If the configuration parameters *allow_host_creation* and *allow_service_creation* are set in the module configuration file, hosts and services may be created when patching the */host* endpoint.

Each time that the */host* endpoint is patched, the module will check if the concerned host/services exist in the Alignak backend. If they do not exist, they will be created.

Some data may be provided for the creation in the *template* property. If no template data are provided, the host/service will be created with the default values defined in the backend. The host/service properties managed in the backend are described in the **'backend documentation<http://docs.alignak.net/projects/alignak-backend/en/develop/resources/confighost.html>'_**.

To create hosts/services, PATCH on the *host* endpoint providing the host (service) data in the *template* property:

```
$ curl -X PATCH -H "Content-Type: application/json" -d '{
    "name": "test_host",
    "template": {
        "alias": "My host...",
        "_templates": ["generic-host", "important"]
    },
```

(continues on next page)

(continued from previous page)

```
    "services": {
        "test_service": {
            "name": "test_ok_0",
            "template": {
                "alias": "My service...",
                "_templates": ["generic-service", "normal"]
            },
        }
    }
}' "http://demo.alignak.net:8888/host"
```

4.6 Hosts / services creation

If the configuration parameters *allow_host_creation* and *allow_service_creation* are set in the module configuration file, hosts and services may be created when patching (or posting) the */host* endpoint.

Each time that the */host* endpoint is patched (or posted), the module will check if the concerned host/services exist in the Alignak backend. If they do not exist, they will be created.

Note that it is recommended to use the HTTP PATCH verb. In a REST API, PATCH is used to update only some information of an object whereas POST is used to create a new object. However, some firewall/gateway equipments do not easily allow using the HTTP verb PATCH. This is why, the Alignak Web Service listener allows to POST on the /hot endpoint.

Some data may be provided for the host/service creation in the *template* property. If no template data are provided, the host/service will be created with the default values defined in the backend. The host/service properties managed in the backend and their default values are described in the **'backend documentation<http://docs.alignak.net/projects/alignak-backend/en/develop/resources/confighost.html>'_**.

The main interesting data that may be used in the *template* property are the *alias*, *_realm* and *_templates* to use for the item creation.

The *alias* is a friendly name tht can be used in the Alignak Web UI instead of the host/service name.

The *_realm* is the name of the realm in which the host will be referenced. If the provided *_realm* is not found in the Alignak backend, the host will be attached to the default *All* realm. Note that the provided realm name can be uppercased, lowercased or capitalized by the Web Service module according to its configuration (see the configuration parameter *realm_case*).

The *_templates* property is the list of the hosts templates to get used for the host creation. The backend templates mechanism is explained in the Alignak backend documentation. The main idea is that a template allows to pre-define all the host properties and services that will be used for a host creation... as suche it make sit easy and simple to create a new host with only one property: its template(s) list!

To create hosts/services, PATCH (or POST) on the *host* endpoint providing the host (service) creation data in the *template* property:

```
$ curl -X PATCH -H "Content-Type: application/json" -d '{
    "name": "test_host",
    "template": {
        "alias": "My host...",
        "_realm": "My realm",
        "_templates": ["generic-host", "important"]
    },
    "services": {
        "test_service": {
            "name": "test_ok_0",
            "template": {
                "alias": "My service...",
                "_templates": ["generic-service", "normal"]
            },
        }
    }
}' "http://demo.alignak.net:8888/host"
```

4.7 External commands

To send an external command, JSON post on the *command* endpoint.

For a global Alignak command:

```
# Disable all notifications from Alignak
$ curl -X POST -H "Content-Type: application/json" -d '{
    "command": "disable_notifications"
}' "http://demo.alignak.net:8888/command"

{"_status": "ok", "_result": "DISABLE_NOTIFICATIONS"}

# Enable all notifications from Alignak
$ curl -X POST -H "Content-Type: application/json" -d '{
    "command": "enable_notifications"
}' "http://demo.alignak.net:8888/command"

{"_status": "ok", "_result": "ENABLE_NOTIFICATIONS"}
```

If your command requires to target a specific element:

```
# Notify a host check result for `always_down` host
$ curl -X POST -H "Content-Type: application/json" -d '{
    "command": "PROCESS_HOST_CHECK_RESULT",
    "element": "always_down",
    "parameters": "0;Host is UP and running"
}' "http://demo.alignak.net:8888/command"

{"_status": "ok", "_result": "PROCESS_HOST_CHECK_RESULT;always_down;0;Host is UP and␣
↪running"}

# Notify a service check result for `always_down/Load` host
$ curl -X POST -H "Content-Type: application/json" -d '{
    "command": "PROCESS_SERVICE_CHECK_RESULT",
    "element": "always_down/Load",
    "parameters": "0;Service is OK|'My metric=12%:80:90:0:100"
}' "http://demo.alignak.net:8888/command"

{"_status": "ok", "_result": "PROCESS_SERVICE_CHECK_RESULT;always_down/Load;0;Service␣
↪is OK"}

# Notify a service check result for `always_down/Load` host (Alignak syntax)
$ curl -X POST -H "Content-Type: application/json" -d '{
    "command": "PROCESS_SERVICE_CHECK_RESULT",
    "host": "always_down",
    "service": "Load",
    "parameters": "0;Service is OK|'My metric=12%:80:90:0:100"
}' "http://demo.alignak.net:8888/command"

{"_status": "ok", "_result": "PROCESS_SERVICE_CHECK_RESULT;always_down/Load;0;Service␣
↪is OK"}
```

Note: the *element* parameter is the old fashioned Nagios way to target an element and you can target a service with *host;service* syntax or with *host/service* syntax. Alignak recommands to use the *host*, *service* or *user* parameters in place of *element* !

Note: a timestamp (integer or float) can also be provided. If it does not exist, Alignak will use the time it receives the

command as a timestamp. Specify a *timestamp* parameter if you want to set a specific one for the command

```
# Notify a host check result for `always_down` host at a specific time stamp
$ curl -X POST -H "Content-Type: application/json" -d '{
    "timestamp": "1484992154",
    "command": "PROCESS_HOST_CHECK_RESULT",
    "element": "always_down",
    "parameters": "0;Host is UP and running"
}' "http://demo.alignak.net:8888/command"

{"_status": "ok", "_result": "PROCESS_HOST_CHECK_RESULT;always_down;0;Host is UP and␣
↪running"}
```

Note: for the available external commands, see the Alignak documentation chapter on the external commands.

4.8 Host event

4.8.1 Host event

To send an host/service comment, POST on the *event* endpoint providing the host/service name and the required comment.

The result is a JSON object containing a *_status* property that should be 'OK' and a *_result* array property that contains information about the actions that were executed.

If an error is detected, the *_status* property is not 'OK' and a *_issues* array property will report the detected error(s).

Adding a comment for an host

```
$ curl --request POST \
  --url http://demo.alignak.net:8888/event \
  --header 'authorization: Basic␣
↪MTQ4NDU1ODM2NjkyMi1iY2Y3Y2NmMS03MjM4LTQ4N2ItYWJkOS0zMGNlZDdlNDI2ZmI6' \
  --header 'cache-control: no-cache' \
  --header 'content-type: application/json' \
  --data '
  {
     "host": "test_host",
     "comment": "My host comment"
  }'

# JSON result
{"_status": "OK", "_result": ["ADD_HOST_COMMENT;test_host;1;Alignak WS;My host comment
↪"]}
```

Adding a comment for a service

```
$ curl --request POST \
  --url http://demo.alignak.net:8888/event \
  --header 'authorization: Basic␣
↪MTQ4NDU1ODM2NjkyMi1iY2Y3Y2NmMS03MjM4LTQ4N2ItYWJkOS0zMGNlZDdlNDI2ZmI6' \
  --header 'cache-control: no-cache' \
  --header 'content-type: application/json' \
  --data '
  {
     "host": "test_host",
     "service": "test_service",
     "comment": "My comment"
  }'

# JSON result
{"_status": "OK", "_result": ["ADD_SVC_COMMENT;test_host;test_service;1;Alignak WS;My␣
↪comment"]}
```

If *author* is not specified in the posted data, "Alignak WS" will be used as the author of the comment. If *timestamp* is not specified in the posted data, the comment will be timestamped with the current date/time.