Contents

1 Surveil ......................................................... 3
  1.1 Project Info .................................................. 3
  1.2 Getting started ............................................. 3

2 Surveil project architecture .................................. 5
  2.1 Global project architecture .............................. 5
  2.2 OpenStack Integration ..................................... 5
  2.3 Main components .......................................... 5

3 Tutorials ....................................................... 7
  3.1 Using Surveil ............................................... 7
  3.2 Contributing ............................................... 12

4 Web API .......................................................... 17
  4.1 V1 Web API .................................................... 17
  4.2 V2 Web API .................................................... 22

5 Administration .................................................... 39
  5.1 Surveil API .................................................... 39
  5.2 Surveil Openstack Interface ............................ 40
  5.3 Surveil Web UI ................................................. 41

6 Indices and tables ............................................... 43

HTTP Routing Table ............................................. 45
Table of Contents:
Monitoring as a Service
An OpenStack related project designed to provide highly available, scalable and flexible monitoring for OpenStack.

1.1 Project Info

- Documentation: https://surveil.readthedocs.org/
- IRC: #surveil at Freenode
- Issue tracker: https://waffle.io/surveil/surveil-meta (Also on GitHub)
- Open Gerrit Changesets: https://review.openstack.org/#/q/status:open+surveil,n,z

1.1.1 Related projects

- Bansho (Surveil Web UI): https://github.com/stackforge/bansho
- Puppet module: https://github.com/stackforge/puppet-surveil

1.2 Getting started

There is a getting started guide available here.
Surveil project architecture

2.1 Global project architecture

2.2 OpenStack Integration

2.3 Main components

- **Surveil**: REST API
- **python-surveileclient**: command line interface and Python library
- **Alignak**: Core monitoring framework
- **Bansho**: Surveil web interface
- **InfluxDB**: Storing metrics
- **Redis**: API caching
- **Grafana**: Data visualization
Chapter 2. Surveil project architecture
3.1 Using Surveil

3.1.1 Installing Surveil

Surveil is currently packaged for Centos 7. You can install it via our custom repositories.

0. Installing the repositories

Install the RDO repositories with the following command:

```bash
yum install -y https://rdoproject.org/repos/rdo-release.rpm
```

Install the Surveil repositories with the following command:

```bash
yum install -y yum-utils
yum-config-manager --add-repo http://yum.surveil.io/centos_7/
```

1. Installing Surveil

All-in-One installation: surveil-full

Surveil does not work with SELinux yet. To disable it, use the following commands:

```bash
echo 0 > /sys/fs/selinux/enforce
sed -i 's/SELINUX=.*/SELINUX=disabled/g' /etc/selinux/config
```

Install surveil-full with the following command:

```bash
yum install -y surveil-full --nogpgcheck
```

Due to an issue with MongoDB presenting itself as running before it is ready, start it 20 seconds before the other services:

```
systemctl start mongod.service
```

Launch all surveil services with the following command:

```
systemctl start surveil-full.target
```
The surveil-init command will flush existing MongoDB Alignak config, create an InfluxDB database and upload configuration templates to Alignak:

```bash
surveil-init --mongodb --influxdb --packs
```

The surveil-webui-init command will pre-create data sources in Grafana:

```bash
surveil-webui-init -H localhost -U root -P root -p 8086 -N db -g "http://localhost/surveil/grafana"
```

2. Testing the API

You should now be able to use the API:

```bash
surveil status-host-list
surveil config-host-list
```

3. Surveil Web UI

Access the Surveil Web UI at http://localhost:80/surveil

3.1.2 Monitoring a host with passive checks

Surveil allows for both passive monitoring and polling. In this guide, we will be creating a host and send passive check results.

0. Creating the host and service

With the Surveil CLI:

```bash
surveil config-host-create --host_name passive_check_host --address 127.0.0.1
surveil config-service-create --host_name passive_check_host --service_description passive_check_service ... 3 --check_period "24x7" --notification_interval 30 --notification_period "24x7" --contacts admin --contact_groups admins
surveil config-reload
```

1. Sending check results

With the Surveil CLI:

```bash
surveil status-submit-check-result --host_name passive_check_host --service_description passive_check_service --output "Hello!" --return_code 0
```

2. Consulting the status of your host

With the Surveil CLI:

```bash
surveil status-service-list
```

3.1.3 Monitoring with your custom plugin

Surveil is compatible with Nagios plugins. It is trivial to write a custom plugin to monitor your application. In this guide, we will create a new plugin and configure a new Host that uses it in Surveil.
0. Install the plugin

Surveil support Nagios plugins. For more information about Nagios plugins, please refer to the Nagios plugin API documentation for more information.

There are many plugins available on the web. For example, the nagios-plugins project contains many plugins written in C and the monitoring-tools project contains many plugins written in Python.

Surveil loads plugins from /usr/lib/monitoring/plugins/. In this example, we will be installing a simple fake plugin written in Bash:

```
#!/bin/bash
echo -e '#!/bin/bash
echo "DISK $1 OK - free space: / 3326 MB (56%); | /=2643MB;5948;5958;0;5968"'
chmod +x /usr/lib/monitoring/plugins/custom/check_example
```

1. Create a host using this plugin

Now that you are done developing your plugin, it is time to use it in Surveil.

Creating a command

Before you can use your plugin in a host/service configuration, you need to create an Alignak command:

```
surveil config-command-create --command_name check_example --command_line "'$CUSTOMPLUGINSDIR$/check_example $HOSTADDRESS$'"
```

Creating a host

Create a host with the following command:

```
surveil config-host-create --host_name check_example_host --address savoirfairelinux.com --use generic-host
```

Creating a Service

Create a service with the following command:

```
surveil config-service-create --host_name check_example_host --service_description check_example_service --check_command ... 3 --check_period "24x7" --notification_interval 30 --notification_period "24x7" --contacts admin --contact_groups admins
```

Reload the config

Reload the config this will tell Alignak to reload the new config with the new host

```
surveil config-reload
```

Show the new service

Show the service list with this command:

```
surveil status-service-list
```

You should see the service you just add in the list with the correct status (this could take a minute a two for the result to show)
3.1.4 Heat AutoScaling with Surveil

When used with OpenStack integration, Surveil export metrics to Ceilometer. This allows for auto scaling based on application metrics with Heat.

For example, the autoscaling.yaml template below allows for scaling when there is an average of more than four users connected to the machines in the stack (via ssh).

```yaml
autoscaling.yml

heat_template_version: 2013-05-23
description: Creates an autoscaling group based on Surveil's metrics
parameters:
  image:
    type: string
    default: rhel7-updated
    description: Image used for servers
  key:
    type: string
    default: < USER KEY HERE >
    description: SSH key to connect to the servers
  flavor:
    type: string
    default: c1.small
    description: flavor used by the web servers
  network_public:
    type: string
    default: public-01
    description: Public network used by the server
  network_private:
    type: string
    default: private-01
    description: Private network used by the server
  monitoring_server:
    type: string
    default: < SURVEIL SERVER IP HERE >
    description: Monitoring server address to allow connections from
resources:
  asg:
    type: OS::Heat::AutoScalingGroup
    properties:
      min_size: 1
      max_size: 6
      resource:
        type: OS::Nova::Server
        properties:
          flavor: {get_param: flavor}
          image: {get_param: image}
          key_name: {get_param: key}
          networks:
            - network: {get_param: network_public}
            - network: {get_param: network_private}
          security_groups:
            - default
            - sysadmin
            - insecure
        metadata:
```
metering.stack: {get_param: "OS::stack_id"}
surveil_tags: linux-system-nrpe
user_data_format: RAW
user_data:
str_replace:
template: |
#!/bin/bash
# These commands install and configure Nagios plugins and NRPE.
rpm -Uvh http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
yum install -y nrpe wget bc
yum install -y nagios-plugins-users nagios-plugins-disk nagios-plugins-load --disabledep
mkdir -p /usr/lib64/nagios/plugins/sfl-monitoring-tools/check_users
svn checkout https://github.com/savoirfairelinux/monitoring-tools/tags/0.3.2/plugins
svn checkout https://github.com/savoirfairelinux/monitoring-tools/tags/0.3.2/plugins
sed -i 's/^allowed_hosts=.*/allowed_hosts=$monitoring_server/' /etc/nagios/nrpe.cfg
echo "command[check_disk]=/usr/lib64/nagios/plugins/check_disk -w 85 -c 90 " >> /etc/nagios/nrpe.cfg
echo "command[check_cpu]=/usr/lib64/nagios/plugins/sfl-monitoring-tools/check_cpu/check_cpu -w 80 -c 90 " >> /etc/nagios/nrpe.cfg
echo "command[check_memory]=/usr/lib64/nagios/plugins/sfl-monitoring-tools/check_mem/check_mem -u -w 80.0 -c 90.0 " >> /etc/nagios/nrpe.cfg
echo "command[check_swap]=/usr/lib64/nagios/plugins/sfl-monitoring-tools/check_swap/check_swap 20 10 " >> /etc/nagios/nrpe.cfg
systemctl enable nrpe
systemctl start nrpe

server_scaleup_policy:
type: OS::Heat::ScalingPolicy
properties:
  adjustment_type: change_in_capacity
  auto_scaling_group_id: {get_resource: asg}
  cooldown: 30
  scaling_adjustment: 1

server_scaledown_policy:
type: OS::Heat::ScalingPolicy
properties:
  adjustment_type: change_in_capacity
  auto_scaling_group_id: {get_resource: asg}
  cooldown: 30
  scaling_adjustment: -1

users_alarm_high:
type: OS::Ceilometer::Alarm
properties:
  description: Scale-up if the average connected users is > 3 for 1 minute
  meter_name: SURVEIL_users
  statistic: avg
  period: 60
  evaluation_periods: 1
  threshold: 3
  alarm_actions:
    - {get_attr: [server_scaleup_policy, alarm_url]}
  matching_metadata: {'stack': {get_param: "OS::stack_id"}}
  comparison_operator: gt

users_alarm_low:
type: OS::Ceilometer::Alarm
properties:
  description: Scale-down if the average connected users is < 1 for 1 minute
  meter_name: SURVEIL_users
statistic: avg
period: 60
evaluation_periods: 1
threshold: 1
alarm_actions:
  - {get_attr: [server_scaledown_policy, alarm_url]}
matching_metadata: {'stack': {get_param: "OS::stack_id"}}
comparison_operator: lt

outputs:
scale_up_url:
description: >
  This URL is the webhook to scale up the autoscaling group. You
  can invoke the scale-up operation by doing an HTTP POST to this
  URL; no body nor extra headers are needed.
  value: {get_attr: [server_scaleup_policy, alarm_url]}
scale_dn_url:
description: >
  This URL is the webhook to scale down the autoscaling group.
  You can invoke the scale-down operation by doing an HTTP POST to
  this URL; no body nor extra headers are needed.
  value: {get_attr: [server_scaledown_policy, alarm_url]}
ceilometer_query:
value:
  str_replace:
    template: >
      ceilometer statistics -m SURVEIL_users
      -q metadata.user_metadata.stack=$stackval -p 600 -a avg
    params:
      $stackval: { get_param: "OS::stack_id" }
description: >
  This is a Ceilometer query for statistics on the SURVEIL_users meter
  Samples about OS::Nova::Server instances in this stack. The -q
  parameter selects Samples according to the subject's metadata.
  When a VM's metadata includes an item of the form metering.X=Y,
  the corresponding Ceilometer resource has a metadata item of the
  form user_metadata.X=Y and samples about resources so tagged can
  be queried with a Ceilometer query term of the form
  metadata.user_metadata.X=Y. In this case the nested stacks give
  their VMs metadata that is passed as a nested stack parameter,
  and this stack passes a metadata of the form metering.stack=Y,
  where Y is this stack's ID.

3.2 Contributing

3.2.1 Getting started with Surveil

0. Prerequisite

Surveil's development environment is based on Docker and docker-compose.

First you need to install Docker. Refer to the project installation documentation.

You can install docker-compose with the following command:

  sudo pip install -U docker-compose
1. Starting the containers

You will then be able to use the environment with the following commands:

- `sudo docker-compose up`: Launch Surveil and its dependencies in containers.
- `sudo docker-compose down`: Kill the active docker containers, if any.
- `sudo docker-compose rm`: Remove all containers, if any.
- `sudo docker-compose build`: Build the docker images.

Configuration for the different services running in the Docker containers are stored in `tools/docker`.

After running `sudo docker-compose up`, you should be able to access all services at the ports configured in the `docker-compose.yml` file.

- Surveil API: `http://localhost:5311/v1/hello`
- Bansho (surveil web interface): `http://localhost:8888` (any login info is fine)
- Grafana: `http://localhost:80` (user:admin pw:admin)
- Shinken WebUI: `http://localhost:7767/all` (user:admin pw:admin)

After about 40 seconds, a script will be executed to create fake hosts in the Surveil configuration. You should see it in the docker-compose logs.

The Surveil container mounts your local project folder and pecan reloads every time the project files change thus providing a proper development environment.

**Note:** Fedora users might want to uncomment the `privileged: true` line in `docker-compose.yml` if they face permissions issues.

2. Interacting with the API

You can use the `python-surveilclient` CLI to interact with the API.

Install it with the following command:

```
sudo pip install -U python-surveilclient
```

You’ll need to provide the Surveil API URL. You can do this with the `--surveil-api-url` parameter, but it’s easier to just set it as environment variable:

```
export SURVEIL_API_URL=http://localhost:5311/v2
export SURVEIL_AUTH_URL=http://localhost:5311/v2/auth
```

**Viewing host status**

You can use the CLI to view the status of the currently monitored hosts and services with `surveil status-host-list` and `surveil status-service-list`

Example output:

```
+-------------------------------+---------------+-------+------------+-----------------------------------+
| host_name        | address   | state | last_check | plugin_output                     |
|-------------------------------+---------------+-------+------------+-----------------------------------+
| srv-ldap-01        | 127.0.0.1   | UP    | 1431712968 | OK - 127.0.0.1: rta 0.036ms, l... |
| sw-iwebcore-01     | 127.0.0.1   | UP    | 1431712971 | OK - 127.0.0.1: rta 0.041ms, l... |
```
You can also use the CLI to view the configured hosts in the API with `surveil config-host-list` and `surveil config-service-list`.

**Adding a new host**

The Surveil CLI provides function to add hosts:

```
surveil config-host-create --host_name openstackwebsite --address openstack.org
```

This will configure a new host in Surveil. However, it won’t be monitored until Surveil’s config is reloaded. You can do this with the CLI:

```
surveil config-reload
```

It will take from 5 to 10 seconds for Surveil to start monitoring the host. After this delay, you will be able to consult the host status with the CLI:

```
surveil status-host-list
```

**Using Bansho the web interface**

The Surveil client uses the Surveil API to query information concerning hosts and services. Bansho (Surveil’s web interface) also uses this API. To use Bansho simply open a browser at [http://localhost:8888](http://localhost:8888) and press login.

**3.2.2 Developping the API**

**Launching the stack**

If you have completed the *Getting started with Surveil* tutorial, you should know how to launch the stack:

```
sudo docker-compose up
```

**Editing the code**

The Surveil container mounts your local project folder and pecan reloads every time the project files change thus providing a proper development environment.

For example, edit the `surveil/api/controllers/v2/hello.py` file and change **Hello World!** by **Hello Devs!**.

After you save the file, the following logs will appear in Surveil’s output:

```
surveil_1 | Some source files have been modified
surveil_1 | Restarting server...
```

You should be able to test your modification by accessing [http://localhost:5311/v2/hello](http://localhost:5311/v2/hello) with your browser.
Disabling permissions

Depending on what you are working on, it might be practical to disable permissions. This can be done by editing the `policy.json` file found at `etc/surveil/policy.json`.

For example, you could modify the following lines:

```
"admin_required": "role:admin or is_admin:1",
"surveil_required": "role:surveil or rule:admin_required",

"surveil:admin": "rule:admin_required",
"surveil:authenticated": "rule:surveil_required",
```

by:

```
"admin_required": "@",
"surveil_required": "@",

"surveil:admin": "@",
"surveil:authenticated": "@",
```

This will modify permissions so that all API calls that require the `admin` rule now pass without any verification.

### 3.2.3 Developing the API without docker

You can get development environment without docker:

```
git clone https://review.openstack.org/stackforge/surveil
cd surveil
virtualenv env
source env/bin/activate
pip install -r requirements.txt
python setup.py develop
python setup.py install_data
surveil-api -p env/etc/surveil/config.py -a env/etc/surveil/api_paste.ini -c env/etc/surveil/surveil.cfg
```

Edit your config files:

```
vim env/etc/surveil/config.py
vim env/etc/surveil/surveil.cfg
vim env/etc/surveil/policy.json
vim env/etc/surveil/api_paste.ini
```

Don’t forget to start your databases (MongoDB and InfluxDB)

### 3.2.4 Running the tests

#### Using tox

Surveil is tested and supported on Python 2.7 and Python 3.4. The project uses tox to manage tests.

The following command will run the tests for Python 3.4, Python 2.7, Flake8 and Docs:

```
tox
```

You can also run only one set of tests by specifying the tox environment to run (see tox.ini for more details):
Building the docs

To build the docs, simply run `tox -edocs`. The docs will be available in the `doc/build/html` folder. After every commit, docs are automatically built on readthedocs and hosted on surveil.readthedocs.org.

Integration tests

Integration tests are ran nightly on test.savoirfairelinux.net. You can run them on your machine with `tox -eintegration`. Before you launch the command, make sure that you don’t have any other Surveil containers running as they may interfere with the integration tests. Integration tests will create multiple containers on your machine.
4.1 V1 Web API

4.1.1 Hello

GET /v1/hello
Says hello.

4.1.2 Hosts

GET /v1/hosts
Returns all hosts.

Parameters

- **data** ([Host]) – a host within the request body.

   Return type [Host]

POST /v1/hosts
Create a new host.

Parameters

- **data** ([Host]) – a host within the request body.

   Return type [Host]

GET /v1/hosts/ (host_name)
Returns a specific host.

Return type [Host]

PUT /v1/hosts/ (host_name)
Modify this host.

Parameters

- **data** ([Host]) – a host within the request body.

DELETE /v1/hosts/ (host_name)
Delete this host.

GET /v1/hosts/ (host_name)/services
Returns all services associated with this host.

Return type list([Service])
**GET /v1/hosts/ (host_name) /services/**

`service_name/service_description` Returns a specific service.

Return type `Service`

**POST /v1/hosts/ (host_name) /results**

Submit a new check result.

Parameters

- `data (CheckResult)` – a check result within the request body.

**POST /v1/hosts/ (host_name) /services/**

`service_description/results` Submit a new check result.

Parameters

- `data (CheckResult)` – a check result within the request body.

**type CheckResult**

Data samples:

**Json**

```json
{
  "output": "CPU Usage 98%|c[cpu]=98%;80;95;0;100",
  "return_code": 0,
  "time_stamp": "1409087486"
}
```

**XML**

```
<value>
  <time_stamp>1409087486</time_stamp>
  <return_code>0</return_code>
  <output>CPU Usage 98%|c[cpu]=98%;80;95;0;100</output>
</value>
```

**output**

Type `unicode`

The output of the check.

**return_code**

Type `int`

The return code of the check.

**time_stamp**

Type `unicode`

The time the check was executed. Defaults to now.

**type Host**

Data samples:

**Json**

```json
{
  "address": "192.168.1.254",
  "check_period": "24x7",
  "contact_groups": "router-admins",
  "contacts": "admin,carl",
}
"custom_fields": {
    "OS_AUTH_URL": "http://localhost:8080/v2"
},
"host_name": "bogus-router",
"max_check_attempts": 5,
"notification_interval": 30,
"notification_period": "24x7",
"use": "generic-host"

XML

<value>
  <host_name>bogus-router</host_name>
  <address>192.168.1.254</address>
  <max_check_attempts>5</max_check_attempts>
  <check_period>24x7</check_period>
  <contacts>admin, carl</contacts>
  <contact_groups>router-admins</contact_groups>
  <notification_interval>30</notification_interval>
  <notification_period>24x7</notification_period>
  <use>generic-host</use>
  <custom_fields>
    <item>
      <key>OS_AUTH_URL</key>
      <value>http://localhost:8080/v2</value>
    </item>
  </custom_fields>
</value>

address

Type unicode

The address of the host. Normally, this is an IP address.

check_period

Type unicode

The time period during which active checks of this host can be made.

contact_groups

Type unicode

List of the short names of the contact groups that should be notified.

contacts

Type unicode

A list of the short names of the contacts that should be notified.

custom_fields

Type dict(unicode: unicode)

Custom fields for the host.

host_name

Type unicode

The name of the host.
The template to use for this host

4.1.3 Services

GET /v1/services
Returns all services.

Return type list(Service)

POST /v1/services
Create a new service.

Parameters

• data (Service) – a service within the request body.

Return type Service

type Service

Data samples:

Json

{
    "check_command": "check-disk!/dev/sdb1",
    "check_interval": 5,
    "check_period": "24x7",
    "contact_groups": "linux-admins",
    "contacts": "surveil-ptl,surveil-bob",
    "host_name": "sample-server",
    "max_check_attempts": 5,
    "notification_interval": 3,
    "notification_period": "24x7",
    "retry_interval": 3,
    "service_description": "check-disk-sdb"
}

XML

<value>
    <host_name>sample-server</host_name>
    <service_description>check-disk-sdb</service_description>
    <check_command>check-disk!/dev/sdb1</check_command>
    <max_check_attempts>5</max_check_attempts>
    <check_interval>5</check_interval>
    <retry_interval>3</retry_interval>
    <check_period>24x7</check_period>
    <notification_interval>3</notification_interval>
    <notification_period>24x7</notification_period>
    <contacts>surveil-ptl,surveil-bob</contacts>
    <contact_groups>linux-admins</contact_groups>
</value>
4.1.4 Commands

GET /v1/commands
Returns all commands.

Return type  list(Command)

POST /v1/commands
Create a new command.

Parameters

• data (Command) – a command within the request body.

Return type  Command

GET /v1/commands/(command_name)
Returns a specific command.

Return type  Command

PUT /v1/commands/(command_name)
Modify this command.

Parameters

• data (Command) – a command within the request body.

DELETE /v1/commands/(command_name)
Delete this command.

type Command
Data samples:

Json

{ 
  "command_line": "/bin/check_http",
  "command_name": "check_http"
}

XML

<value>
  <command_name>check_http</command_name>
  <command_line>/bin/check_http</command_line>
</value>

command_line

Type  unicode

This directive is used to define what is actually executed by Shinken

command_name

Type  unicode

The name of the command
4.2 V2 Web API

4.2.1 Config

Hosts

POST /v2/config/hosts
Returns all hosts. :type data: LiveQuery
   Return type list(Host)

PUT /v2/config/hosts
Create a new host.
   Parameters
      • data (Host) – a host within the request body.
   Return type Host

GET /v2/config/hosts/(host_name)
Returns a specific host.
   Return type Host

PUT /v2/config/hosts/(host_name)
Modify this host.
   Parameters
      • data (Host) – a host within the request body.

DELETE /v2/config/hosts/(host_name)
Delete this host.

GET /v2/config/hosts/(host_name)/services
Returns all services assocaited with this host.
   Return type list(Service)

GET /v2/config/hosts/(host_name)/services/
    service_name/service_description Returns a specific service.
   Return type Service

DELETE /v2/config/hosts/(host_name)/services/
    service_name/service_description Delete a specific service.

Services

POST /v2/config/services
Returns all services. :type data: LiveQuery
   Return type list(Service)

PUT /v2/config/services
Create a new service.
   Parameters
      • data (Service) – a service within the request body.
   Return type Service
**Type Service**

Data samples:

**Json**

```json
{
   "check_command": "check-disk!/dev/sdb1",
   "check_interval": 5,
   "check_period": "24x7",
   "contact_groups": [
      "linux-admins"
   ],
   "contacts": [
      "surveil-ptl",
      "surveil-bob"
   ],
   "host_name": [
      "sample-server"
   ],
   "max_check_attempts": 5,
   "notification_interval": 3,
   "notification_period": "24x7",
   "passive_checks_enabled": "1",
   "retry_interval": 3,
   "service_description": "check-disk-sdb"
}
```

**XML**

```xml
<value>
   <host_name>
      <item>sample-server</item>
   </host_name>
   <service_description>check-disk-sdb</service_description>
   <check_command>check-disk!/dev/sdb1</check_command>
   <max_check_attempts>5</max_check_attempts>
   <check_interval>5</check_interval>
   <retry_interval>3</retry_interval>
   <check_period>24x7</check_period>
   <notification_interval>3</notification_interval>
   <notification_period>24x7</notification_period>
   <contacts>
      <item>surveil-ptl</item>
      <item>surveil-bob</item>
   </contacts>
   <contact_groups>
      <item>linux-admins</item>
   </contact_groups>
   <passive_checks_enabled>1</passive_checks_enabled>
</value>
```

**Commands**

**POST /v2/config/commands**

Returns all commands. :type data: LiveQuery

**Return type** list(Command)
PUT /v2/config/commands
Create a new command.

Parameters

• **data** (*Command*) – a command within the request body.

Return type *Command*

GET /v2/config/commands/(command_name)
Returns a specific command.

Return type *Command*

PUT /v2/config/commands/(command_name)
Modify this command.

Parameters

• **data** (*Command*) – a command within the request body.

DELETE /v2/config/commands/(command_name)
Delete this command.

Business impact modulations

POST /v2/config/businessimpactmodulations
Returns all business impact modulations. :type data: LiveQuery

Return type list(*BusinessImpactModulation*)

PUT /v2/config/businessimpactmodulations
Create a new business impact modulation.

Parameters

• **data** (*BusinessImpactModulation*) – a business impact modulation within the request body.

Return type *BusinessImpactModulation*

Check modulations

POST /v2/config/checkmodulations
Returns all check modulations. :type data: LiveQuery

Return type list(*CheckModulation*)

PUT /v2/config/checkmodulations
Create a new check modulation.

Parameters

• **data** (*CheckModulation*) – a check modulation within the request body.

Notification ways

POST /v2/config/notificationways
Returns all notification ways. :type data: LiveQuery

Return type list(*NotificationWay*)
PUT /v2/config/notificationways
Create a new notification way.

Parameters

- **data** (*NotificationWay*) – a notification way within the request body.

types documentation

type Command

Data samples:

**Json**

```json
{
    "command_line": "/bin/check_http",
    "command_name": "check_http"
}
```

**XML**

```xml
<value>
    <command_name>check_http</command_name>
    <command_line>/bin/check_http</command_line>
</value>
```

command_line

Type `unicode`

This directive is used to define what is actually executed by Shinken

command_name

Type `unicode`

The name of the command

type Host

Data samples:

**Json**

```json
{
    "address": "192.168.1.254",
    "check_period": "24x7",
    "contact_groups": [
        "router-admins"
    ],
    "contacts": [
        "admin",
        "carl"
    ],
    "custom_fields": {
        "OS_AUTH_URL": "http://localhost:8080/v2"
    },
    "host_name": "bogus-router",
    "max_check_attempts": 5,
    "notification_interval": 30,
    "notification_period": "24x7",
    "use": [
        "generic-host"
    ]
}
```
XML

```xml
<value>
  <host_name>bogus-router</host_name>
  <address>192.168.1.254</address>
  <max_check_attempts>5</max_check_attempts>
  <check_period>24x7</check_period>
  <contacts>
    <item>admin</item>
    <item>carl</item>
  </contacts>
  <contact_groups>
    <item>router-admins</item>
  </contact_groups>
  <notification_interval>30</notification_interval>
  <notification_period>24x7</notification_period>
  <use>
    <item>generic-host</item>
  </use>
  <custom_fields>
    <item>
      <key>OS_AUTH_URL</key>
      <value>http://localhost:8080/v2</value>
    </item>
  </custom_fields>
</value>
```

**address**

Type `unicode`

The address of the host. Normally, this is an IP address.

**check_period**

Type `unicode`

The time period during which active checks of this host can be made.

**contact_groups**

Type `list(unicode)`

List of the short names of contact groups that should be notified.

**contacts**

Type `list(unicode)`

A list of the short names of the contacts that should be notified.

**custom_fields**

Type `dict(unicode: unicode)`

Custom fields for the host.

**host_name**

Type `unicode`

The name of the host.
use

Type list(unicode)

The template to use for this host.

type CheckResult

Data samples:

Json

{
    "output": "CPU Usage 98%|c[cpu]=98%;80;95;0;100",
    "return_code": 0,
    "time_stamp": "1409087486"
}

XML

<value>
    <time_stamp>1409087486</time_stamp>
    <return_code>0</return_code>
    <output>CPU Usage 98%|c[cpu]=98%;80;95;0;100</output>
</value>

output

Type unicode

The output of the check.

return_code

Type int

The return code of the check.

time_stamp

Type unicode

The time the check was executed. Defaults to now.

type CheckModulation

Data samples:

Json

{
    "check_command": "check_ping_night",
    "check_period": "night",
    "check_modulation_name": "ping_night"
}

XML

<value>
    <check_modulation_name>ping_night</check_modulation_name>
    <check_command>check_ping_night</check_command>
    <check_period>night</check_period>
</value>

type NotificationWay

Data samples:

4.2. V2 Web API
Json

```json
{
    "host_notification_commands": [
        "notify-host"
    ],
    "host_notification_options": [
        "d",
        "u",
        "r",
        "f",
        "s"
    ],
    "host_notification_period": "24x7",
    "notificationway_name": "email_in_day",
    "service_notification_commands": [
        "notify-service"
    ],
    "service_notification_options": [
        "w",
        "u",
        "c",
        "r",
        "f"
    ],
    "service_notification_period": "24x7"
}
```

XML

```xml
<value>
    <notificationway_name>email_in_day</notificationway_name>
    <host_notification_period>24x7</host_notification_period>
    <service_notification_period>24x7</service_notification_period>
    <host_notification_options>
        <item>d</item>
        <item>u</item>
        <item>r</item>
        <item>f</item>
    </host_notification_options>
    <service_notification_options>
        <item>w</item>
        <item>u</item>
        <item>c</item>
        <item>r</item>
        <item>f</item>
    </service_notification_options>
    <host_notification_commands>
        <item>notify-host</item>
    </host_notification_commands>
    <service_notification_commands>
        <item>notify-service</item>
    </service_notification_commands>
</value>
```
4.2.2 Status

Events

webprefix /v2/status/events/

Hosts

GET /v2/status/hosts
Returns all hosts.

Return type list(LiveHost)

POST /v2/status/hosts
Given a LiveQuery, returns all matching hosts. :type query: LiveQuery

Return type list(LiveHost)

GET /v2/status/hosts/ (host_name)
Returns a specific host.

Return type LiveHost

GET /v2/status/hosts/ (host_name)/config
Returns config from a specific host.

POST /v2/status/hosts/ (host_name)/results
Submit a new check result.

Parameters

• data (CheckResult) – a check result within the request body.

GET /v2/status/hosts/ (host_name)/metrics
Returns all metrics name for a host.

Return type list(Metric)

GET /v2/status/hosts/ (host_name)/metrics/ metric_name
Return the last measure for the metric name on the host.

Return type Metric

POST /v2/status/hosts/ (host_name)/metrics/ metric_name
Given a live query, returns all matching metrics.

Parameters

• live_query – a live query within the request body.

Return type list(Metric)

POST /v2/status/hosts/ (host_name)/services/ service_description/results
Submit a new check result.

Parameters

• data (CheckResult) – a check result within the request body.

GET /v2/status/hosts/ (host_name)/services/ service_description/metrics
Returns all metrics name for a host with a service.

Return type list(Metric)
**Services**

**GET /v2/status/services**
Returns all services.

*Return type* list(*LiveService*)

**POST /v2/status/services**
Given a LiveQuery, returns all matching services. *type query: LiveQuery*

*Return type* list(*LiveService*)

**types documentation**

**type LiveService**

Data samples:

**Json**

```json
{
  "acknowledged": true,
  "description": "Serves Stuff",
  "host_name": "Webserver",
  "last_check": 1429220785,
  "last_state_change": 1429220785.481679,
  "long_output": "Serves /var/www/
Serves /home/webserver/www/",
  "plugin_output": "HTTP OK - GOT NICE RESPONSE",
  "service_description": "Apache",
  "state": "OK"
}
```

**XML**

```xml
<value>
  <host_name>Webserver</host_name>
  <service_description>Apache</service_description>
  <description>Serves Stuff</description>
  <state>OK</state>
  <acknowledged>true</acknowledged>
  <last_check>1429220785</last_check>
  <last_state_change>1429220785.48</last_state_change>
  <plugin_output>HTTP OK - GOT NICE RESPONSE</plugin_output>
  <long_output>Serves /var/www/
Serves /home/webserver/www/</long_output>
</value>
```

**acknowledged**

*Type* bool

Wether or not the problem, if any, has been acknowledged

**description**

*Type* unicode

The description of the service

**host_name**

*Type* unicode
The host for the service

last_check
  Type int
  The last time the service was checked

last_state_change
  Type float
  The last time the state has changed

long_output
  Type unicode
  Plugin long output of the last check

plugin_output
  Type unicode
  Plugin output of the last check

service_description
  Type unicode
  The name of the service

state
  Type unicode
  The current state of the service

**type LiveHost**

Data samples:

**Json**

```json
{
  "acknowledged": true,
  "address": "127.0.0.1",
  "childs": [
    "surveil.com"
  ],
  "description": "Very Nice Host",
  "host_name": "CoolHost",
  "last_check": 1429220785,
  "last_state_change": 1429220785,
  "long_output": "The ping was great\nI love epic ping-pong games",
  "parents": [
    "parent.com"
  ],
  "plugin_output": "PING OK - Packet loss = 0%, RTA = 0.02 ms",
  "services": ["load",
               "cpu",
               "disk_usage"
  ],
  "state": "OK"
}
```

4.2. V2 Web API
XML

```xml
<value>
  <host_name>CoolHost</host_name>
  <address>127.0.0.1</address>
  <childs>
    <item>surveil.com</item>
  </childs>
  <parents>
    <item>parent.com</item>
  </parents>
  <description>Very Nice Host</description>
  <state>OK</state>
  <last_check>1429220785</last_check>
  <last_state_change>1429220785</last_state_change>
  <plugin_output>PING OK - Packet loss = 0%, RTA = 0.02 ms</plugin_output>
  <long_output>The ping was great
I love epic ping-pong games</long_output>
  <services>
    <item>load</item>
    <item>cpu</item>
    <item>disk_usage</item>
  </services>
</value>
```

**acknowledged**

Type `bool`  
Wether or not the problem, if any, has been acknowledged

**address**

Type `unicode`  
The address of the host

**childs**

Type `list(unicode)`  
The childs of the host

**description**

Type `unicode`  
The description of the host

**host_name**

Type `unicode`  
The name of the host

**last_check**

Type `int`  
The last time the host was checked

**last_state_change**

Type `int`
The last time the state has changed

**long_output**

Type: unicode

Plugin long output of the last check

**parents**

Type: list(unicode)

The parents of the host

**plugin_output**

Type: unicode

Plugin output of the last check

**services**

Type: list(unicode)

The services of the host

**state**

Type: unicode

The current state of the host

**type Metric**

Data samples:

**Json**

```
{
    "critical": "100",
    "max": "100",
    "metric_name": "pl",
    "min": "0",
    "unit": "s",
    "value": "0",
    "warning": "100"
}
```

**XML**

```
<value>
    <metric_name>pl</metric_name>
    <max>100</max>
    <min>0</min>
    <critical>100</critical>
    <warning>100</warning>
    <value>0</value>
    <unit>s</unit>
</value>
```

**critical**

Type: unicode

Critical value for the metric

**max**
**Surveil, Release**

- **Type** \(\text{unicode}\)
  
  Maximum value for the metric

- **metric_name**
  
  **Type** \(\text{unicode}\)
  
  Name of the metric

- **min**
  
  **Type** \(\text{unicode}\)
  
  Minimal value for the metric

- **unit**
  
  **Type** \(\text{unicode}\)
  
  Unit of the metric

- **value**
  
  **Type** \(\text{unicode}\)
  
  Current value of the metric

- **warning**
  
  **Type** \(\text{unicode}\)
  
  Warning value for the metric

**type TimeInterval**

- **Hold a time.**

  Data samples:

  **Json**

  ```json
  {
    "end_time": "2015-01-29T22:50:44Z",
    "start_time": "2015-01-29T21:50:44Z"
  }
  ```

  **XML**

  ```xml
  <value>
  <start_time>2015-01-29T21:50:44Z</start_time>
  <end_time>2015-01-29T22:50:44Z</end_time>
  </value>
  ```

- **end_time**
  
  **Type** \(\text{unicode}\)
  
  The ending time.

- **start_time**
  
  **Type** \(\text{unicode}\)
  
  The starting time.

**type Event**

- **Data samples:**
Json

```json
{
  "alert_type": "SERVICE",
  "attempts": 4,
  "event_type": "ALERT",
  "host_name": "CoolHost",
  "notification_method": "notify-service-by-email",
  "notification_type": "",
  "output": "WARNING - load average: 5.04, 4.67, 5.04",
  "service_description": "Apache Service",
  "state": "CRITICAL",
  "state_type": "HARD",
  "time": "2015-06-04T18:55:12Z"
}
```

XML

```xml
<value>
  <time>2015-06-04T18:55:12Z</time>
  <event_type>ALERT</event_type>
  <host_name>CoolHost</host_name>
  <service_description>Apache Service</service_description>
  <state>CRITICAL</state>
  <state_type>HARD</state_type>
  <attempts>4</attempts>
  <notification_type />
  <notification_method>notify-service-by-email</notification_method>
  <alert_type>SERVICE</alert_type>
  <output>WARNING - load average: 5.04, 4.67, 5.04</output>
</value>
```

**alert_type**

Type: unicode

Type of alert. This is only HOST or SERVICE

**attempts**

Type: int

Number of attempts to confirm state

**downtime_type**

Type: unicode

Type of alert. This is only HOST or SERVICE

**event_type**

Type: unicode

Type of event. This is only ALERT

**host_name**

Type: unicode

Host which the alert is from.

**output**

Type: unicode
Additional output of the alert.

**service_description**
- Type: unicode
  - Service which raised the alert

**state**
- Type: unicode
  - State of the service or host who raised the alert

**state_type**
- Type: unicode
  - Confirmness level of the state [SOFT|HARD]

**time**
- Type: unicode
  - Timestamp of the alert

### 4.2.3 Actions

**acknowledge**

**POST /v2/actions/acknowledge**
  - Acknowledge a host/service. :type ack: Acknowledgement
  - Return type: Info

**DELETE /v2/actions/acknowledge**
  - Delete a host/service acknowledgement. :type ack: Acknowledgement
  - Return type: Info

**downtime**

**POST /v2/actions/downtime**
  - Put a host/service in downtime. :type dt: Downtime
  - Return type: Info

**DELETE /v2/actions/downtime**
  - Delete a host/service downtime. :type dt: Downtime
  - Return type: Info

**types documentation**

**type Acknowledgement**
  - Data samples:
    - Json
{ "author": "aviau", "comment": "Working on it.", "host_name": "localhost", "notify": 0, "persistent": 1, "service_description": "ws-arbiter", "sticky": 1, "time_stamp": ""}

XML
<value>
  <host_name>localhost</host_name>
  <service_description>ws-arbiter</service_description>
  <time_stamp />
  <sticky>1</sticky>
  <notify>0</notify>
  <persistent>1</persistent>
  <author>aviau</author>
  <comment>Working on it.</comment>
</value>

host_name
  Type unicode
  The name of the host

type Downtime
  Data samples:

Json
{
  "author": "aviau",
  "comment": "No comment.",
  "duration": 86400,
  "end_time": 1430150469,
  "fixed": 1,
  "host_name": "localhost",
  "service_description": "ws-arbiter",
  "start_time": 1430150469,
  "time_stamp": 1430150469,
  "trigger_id": 0
}

XML
<value>
  <host_name>localhost</host_name>
  <service_description>ws-arbiter</service_description>
  <time_stamp>1430150469</time_stamp>
  <start_time>1430150469</start_time>
  <end_time>1430150469</end_time>
  <fixed>1</fixed>
  <duration>86400</duration>
  <trigger_id>0</trigger_id>
  <author>aviau</author>

4.2. V2 Web API
4.2.4 Bansho

Config

GET /v2/bansho/config
Retrieve user config, empty dict if no config exists.

Return type unicode

POST /v2/bansho/config
Save user config.

Parameters
  • config (unicode) – JSON config object
This section will cover the administration and configuration of the Surveil services.

## 5.1 Surveil API

The Surveil API provides Surveil’s REST API.

<table>
<thead>
<tr>
<th>package name (RPM)</th>
<th>surveil</th>
</tr>
</thead>
<tbody>
<tr>
<td>services</td>
<td>surveil-api.service</td>
</tr>
<tr>
<td>Default port</td>
<td>5311</td>
</tr>
<tr>
<td>configuration (API)</td>
<td>/etc/surveil/surveil.cfg</td>
</tr>
<tr>
<td>configuration (permissions)</td>
<td>/etc/surveil/policy.json</td>
</tr>
<tr>
<td>configuration (API - pipeline)</td>
<td>/etc/surveil/api_paste.ini</td>
</tr>
</tbody>
</table>

The Surveil API needs access to InfluxDB, Alignak and MongoDB. If Keystone authentication is enabled, it needs access to Keystone (see api_paste.ini).

### 5.1.1 Configuration samples

**/etc/surveil/surveil.cfg**

```ini
[surveil]

# mongodb_uri is used to connect to MongoDB. Uses the MongoDB Connection
# String URI Format
mongodb_uri= mongodb://mongo:27017

# ws_arbiter_url is the endpoint of the ws-arbiter module of Alignak it is
# used to send commands to Alignak
ws_arbiter_url= http://alignak:7760

# influxdb_uri is used to connect to InfluxDB. Uses the python-influxdb
# connection string format
influxdb_uri= influxdb://root:root@influxdb:8086/db
```

**/etc/surveil/policy.json**

For documentation on this configuration file, refer to the OpenStack documentation.
surveil, Release

```
{
    "admin_required": "role:admin or is_admin:1",
    "surveil_required": "role:surveil or rule:admin_required",

    "surveil:admin": "rule:admin_required",
    "surveil:authenticated": "rule:surveil_required",

    "surveil:break": "!",
    "surveil:pass": "@"
}
```

/etc/surveil/api_paste.ini

```
# Surveil API WSGI Pipeline
# Define the filters that make up the pipeline for processing WSGI requests

# Replace `surveil-auth` by `authtoken` to enable Keystone authentication.
[pipeline:main]
pipeline = surveil-auth api-server

[app:api-server]
paste.app_factory = surveil.api.app:app_factory

[filter:surveil-auth]
paste.filter_factory = surveil.api.authmiddleware.auth:filter_factory

[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory

# Keystone auth settings
auth_host=198.72.123.131
auth_protocol=http
admin_user=admin
admin_password=password
admin_tenant_name=admin
```

5.2 Surveil Openstack Interface

surveil-os-interface is a daemon that connects to the OpenStack message queue. It reacts to various events and automatically configures Surveil monitoring. For example, instances created in Nova will automatically be monitored by Surveil.

<table>
<thead>
<tr>
<th>package name (RPM)</th>
<th>surveil</th>
</tr>
</thead>
<tbody>
<tr>
<td>services</td>
<td>surveil-os-interface.service</td>
</tr>
<tr>
<td>configuration</td>
<td>/etc/surveil/surveil_os_interface.cfg</td>
</tr>
</tbody>
</table>

Surveil-os-interface needs access to OpenStack’s message queue. The following options must be set in /etc/nova/nova.conf:

```
notification_driver=nova.openstack.common.notifier.rpc_notifier
notification_topics=notifications,surveil
notify_on_state_change=vm_and_task_state
notify_on_any_change=True
```
5.2.1 Configuration samples

/etc/surveil/surveil_os_interface.cfg

```
[surveil-os-interface]

# Surveil API URL
SURVEIL_API_URL=http://surveil:8080/v2

# Surveil Auth URL
SURVEIL_AUTH_URL=http://surveil:8080/v2/auth

# Surveil version
SURVEIL_VERSION=2_0

# OpenStack Credentials. Used for creating hosts in Surveil.
SURVEIL_OS_AUTH_URL=http://localhost/v2.0
SURVEIL_OS_USERNAME=admin
SURVEIL_OS_PASSWORD=password
SURVEIL_OS_TENANT_NAME=admin

# Default monitoring pack to use with all OpenStack instances
SURVEIL_DEFAULT_TAGS=openstack-host

# Network used to monitor hosts. Surveil must have access to this network.
SURVEIL_NETWORK_LABEL=surveil

# AMQP credentials
RABBIT_HOST=192.168.49.239
RABBIT_PORT=5672
QUEUE=surveil
RABBIT_USER=admin
RABBIT_PASSWORD=admin
```

5.3 Surveil Web UI

The Surveil Web UI is a web interface for Surveil.

<table>
<thead>
<tr>
<th>package name (RPM)</th>
<th>surveil-webui</th>
</tr>
</thead>
<tbody>
<tr>
<td>required services</td>
<td>httpd.service</td>
</tr>
<tr>
<td>Default port</td>
<td>80</td>
</tr>
<tr>
<td>configuration (global)</td>
<td>/etc/surveil-webui/config.json</td>
</tr>
<tr>
<td>configuration (user config)</td>
<td>/etc/surveil-webui/default_user_config.json</td>
</tr>
</tbody>
</table>

surveil-webui implements the Surveil API. It needs access to the Surveil API endpoint and Grafana. By default, it is packaged with a reverse proxy in `/etc/http/conf.d/surveil`:

```

ProxyPass /surveil/surveil/ http://localhost:5311/
ProxyPassReverse /surveil/surveil/ http://localhost:5311/

RequestHeader set GRAFANA-USER "admin"
```
Indices and tables

- genindex
- modindex
- search
## HTTP Routing Table

### /v1

- GET /v1/commands, 21
- GET /v1/commands/(command_name), 21
- GET /v1/hello, 17
- GET /v1/hosts, 17
- GET /v1/hosts/(host_name), 17
- GET /v1/hosts/(host_name)/services, 17
- GET /v1/hosts/(host_name)/services/(service_name)/(service_description), 17
- GET /v1/services, 20
- POST /v1/commands, 21
- POST /v1/hosts, 17
- POST /v1/hosts/(host_name)/results, 18
- POST /v1/hosts/(host_name)/services/(service_description)/results, 18
- PUT /v1/commands/(command_name), 21
- DELETE /v1/commands/(command_name), 21
- DELETE /v1/hosts/(host_name), 17

### /v2

- GET /v2/bansho/config, 38
- GET /v2/config/commands/(command_name), 24
- GET /v2/config/hosts/(host_name), 22
- GET /v2/config/hosts/(host_name)/services, 22
- GET /v2/config/hosts/(host_name)/services/(service_name)/(service_description), 22
- GET /v2/status/hosts, 29
- GET /v2/status/hosts/(host_name), 29
- GET /v2/status/hosts/(host_name)/config, 29
- GET /v2/status/hosts/(host_name)/metrics, 29
- POST /v2/status/hosts/(host_name)/metrics/(metric_name), 29
- POST /v2/status/hosts/(host_name)/services/(service_description)/metrics, 29
- POST /v2/actions/acknowledge, 30
- POST /v2/actions/downtime, 36
- POST /v2/bansho/config, 38
- POST /v2/config/businessimpactmodulations, 24
- POST /v2/config/checkmodulations, 24
- POST /v2/config/commands, 23
- POST /v2/config/commands/(command_name), 24
- POST /v2/config/hosts, 22
- POST /v2/config/hosts/(host_name), 22
- POST /v2/config/notificationways, 24
- POST /v2/config/services, 22
- POST /v2/status/hosts, 29
- POST /v2/status/hosts/(host_name)/metrics/(metric_name), 29
- PUT /v2/config/businessimpactmodulations, 24
- PUT /v2/config/checkmodulations, 24
- PUT /v2/config/commands, 23
- PUT /v2/config/commands/(command_name), 24
- PUT /v2/config/hosts, 22
- PUT /v2/config/hosts/(host_name), 22
- PUT /v2/config/notificationways, 24
- PUT /v2/config/services, 22
- DELETE /v2/actions/acknowledge, 30
- DELETE /v2/actions/downtime, 36
- DELETE /v2/config/commands/(command_name), 24
- DELETE /v2/config/hosts/(host_name), 22
- DELETE /v2/config/hosts/(host_name)/services/(service_description), 22
- DELETE /v2/status/hosts/(host_name)/services/(service_description)/results, 29
- DELETE /v2/status/hosts/(host_name)/services/(service_description)/metrics, 29
Acknowledged (surveil.api.datamodel.status.live_host.LiveHost attribute), 32
Acknowledged (surveil.api.datamodel.status.live_service.LiveService attribute), 30
Acknowledgement (webservice type), 36
address (surveil.api.controllers.v1.datamodel.host.Host attribute), 19
address (surveil.api.datamodel.config.host.Host attribute), 26
address (surveil.api.datamodel.status.live_host.LiveHost attribute), 32
alert_type (surveil.api.datamodel.status.event.Event attribute), 35
attempts (surveil.api.datamodel.status.event.Event attribute), 35
author (surveil.api.datamodel.actions.downtime.Downtime attribute), 38

check_period (surveil.api.controllers.v1.datamodel.host.Host attribute), 19
check_period (surveil.api.datamodel.config.host.Host attribute), 26
CheckModulation (webservice type), 27
CheckResult (webservice type), 18, 27
child (surveil.api.datamodel.status.live_host.LiveHost attribute), 32
Command (webservice type), 21, 25
command_line (surveil.api.controllers.v1.datamodel.command.Command attribute), 21
command_line (surveil.api.datamodel.config.command.Command attribute), 25
custom_fields (surveil.api.controllers.v1.datamodel.host.Host attribute), 19
custom_fields (surveil.api.datamodel.config.host.Host attribute), 26
critical (surveil.api.datamodel.status.metrics.metric.Metric attribute), 33
custom_fields (surveil.api.controllers.v1.datamodel.host.Host attribute), 19
custom_fields (surveil.api.datamodel.config.host.Host attribute), 26
descrition (surveil.api.datamodel.status.live_host.LiveHost attribute), 32
description (surveil.api.datamodel.status.live_service.LiveService attribute), 30
Downtime (webservice type), 37
downtime_type (surveil.api.datamodel.status.event.Event attribute), 35
duration (surveil.api.datamodel.actions.downtime.Downtime attribute), 38

end_time (surveil.api.datamodel.actions.downtime.Downtime attribute), 38
end_time (surveil.api.datamodel.status.metrics.time_interval TimeInterval attribute), 34
Event (webservice type), 34
downtime_type (surveil.api.datamodel.status.event.Event attribute), 35

host_name (surveil.api.controllers.v1.datamodel.host.Host attribute), 19
host_name (surveil.api.datamodel.actions.acknowledgement.Acknowledgement attribute), 37
Index