Zabbix

task

- ~~dashboard con conteggi aggregati (e altri grafici interessanti)~~
- ~~trigger di allarme: * luce accesa da troppo tempo~~
- ~~export dati csv con API~~
- ~~gestire utente non privilegiato su zabbix~~
- ~~VPN robusta che riparte quando il cloud non è raggiungibile~~
- gestire il mancato boot della componente zabbix di hass
  - /etc/openvpn/iotaiuto.conf ?
  - come far ripartire la connessione zabbix all'interno di hass ?
- esphome:
  - costruire in power meter con allarme su alexa
  - costruire un citofono con ESP32 CAM

API

```
#!/bin/sh


# 1. set connection details
url=http://monitor.iotaiuto.it/api_jsonrpc.php
user=xxx
password=xxx

# 2. get authorization token
auth=$(curl -s -X POST \
-H 'Content-Type: application/json-rpc' \
-d " \
{
 \"jsonrpc\": \"2.0\",
 \"method\": \"user.login\",
 \"params\": {
  \"user\": \"$user\",
  \"password\": \"$password\"
 },
 \"id\": 1,
 \"auth\": null
}
" $url | \
jq -r '.result'
)

# 3. show triggers in problem state
curl -s -X POST \
-H 'Content-Type: application/json-rpc' \
```

```
-d " \
{
 \"jsonrpc\": \"2.0\",
    \"method\": \"trigger.get\",
    \"params\": {
        \"output\": \"extend\",
        \"selectHosts\": \"extend\",
        \"filter\": {
            \"value\": 1
        },
        \"sortfield\": \"priority\",
        \"sortorder\": \"DESC\"
    },
    \"auth\": \"$auth\",
    \"id\": 1
}
" $url | \
jq -r '.result'

# 4. logout user
curl -s -X POST \
-H 'Content-Type: application/json-rpc' \
-d " \
{
    \"jsonrpc\": \"2.0\",
    \"method\": \"user.logout\",
    \"params\": [],
    \"id\": 1,
    \"auth\": \"$auth\"
}
" $url
 ~  Scaricati 
 →  # 1. set connection details
    url=http://127.0.0.1/api_jsonrpc.php
    user=api
    password=zabbix

    # 2. get authorization token
    auth=$(curl -s -X POST \
    -H 'Content-Type: application/json-rpc' \
    -d " \
    {
     \"jsonrpc\": \"2.0\",
     \"method\": \"user.login\",
     \"params\": {
      \"user\": \"$user\",
      \"password\": \"$password\"
     },
     \"id\": 1,
    # 1. set connection details
    url=http://127.0.0.1/api_jsonrpc.php
```

```
user=api \
password=zabbix
)
# 2. get authorization token
auth=$(curl -s -X POST \oblem state
-H 'Content-Type: application/json-rpc' \
-d " \ 'Content-Type: application/json-rpc' \
{   -d " \
 \"jsonrpc\": \"2.0\",
 \"method\": \"user.login\",
 \"params\": {: \"trigger.get\",
  \"user\": \"$user\",
  \"password\": \"$password\"",
 },     \"selectHosts\": \"extend\",
 \"id\": 1,ilter\": {
 \"auth\": nullalue\": 1
}       },
" $url | \sortfield\": \"priority\",
jq -r '.result'rder\": \"DESC\"
)   },
    \"auth\": \"$auth\",
# 3. show triggers in problem state
curl -s -X POST \
" $u-H 'Content-Type: application/json-rpc' \
    -d " \'.result'
{
 \"jsonrpc\": \"2.0\",
    \"method\": \"trigger.get\",
    \"params\": {ype: application/json-rpc' \
        \"output\": \"extend\",
        \"selectHosts\": \"extend\",
        \"filter\": {0\",
            \"value\": 1ogout\",
        },ms\": [],
        \"sortfield\": \"priority\",
        \"sortorder\": \"DESC\"
    },
    \"auth\": \"$auth\",
    \"id\": 1
}
" $url | \
    jq -r '.result'

# 4. logout user
curl -s -X POST \
    -H 'Content-Type: application/json-rpc' \
    -d " \
{
    \"jsonrpc\": \"2.0\",
    \"method\": \"user.logout\",
    \"params\": [],
```

```
        \"id\": 1,
        \"auth\": \"$auth\"
    }
    " $url
```

```
#!/bin/bash

# 1. set connection details
url=http://monitor.iotaiuto.it/api_jsonrpc.php
user=admin
password=Galileo2018.

# 2. get authorization token
auth=$(curl -s -X POST \
-H 'Content-Type: application/json-rpc' \
-d " \
{
 \"jsonrpc\": \"2.0\",
 \"method\": \"user.login\",
 \"params\": {
  \"user\": \"$user\",
  \"password\": \"$password\"
 },
 \"id\": 1,
 \"auth\": null
}
" $url | \
jq -r '.result'
)

# 3. show triggers in problem state
curl -s -X POST \
-H 'Content-Type: application/json-rpc' \
-d " \
{
 \"jsonrpc\": \"2.0\",
    \"method\": \"trigger.get\",
    \"params\": {
        \"output\": \"extend\",
        \"selectHosts\": \"extend\",
        \"filter\": {
            \"value\": 1
        },
        \"sortfield\": \"priority\",
        \"sortorder\": \"DESC\"
    },
    \"auth\": \"$auth\",
    \"id\": 1
}
" $url | \
jq -r '.result' > ../flussi/dati.json
```

```
jq -r 'del(.[].hosts) | (map(keys) | add | unique) as $cols | map(. as $row
| $cols | map($row[.])) as $rows | $cols, $rows[] | @csv'
../flussi/dati.json > ../flussi/dati_trigger.csv
jq -r '(map(.hosts[] + {"triggerid": .triggerid} | keys) | add | unique) as
$cols | map(. as $row | $cols | map($row[.])) as $rows | $cols, $rows[] |
@csv' ../flussi/dati.json > ../flussi/dati_host.csv
rm ../flussi/dati.json
cat ../flussi/dati_trigger.csv & ../flussi/dati_host.csv

# 4. logout user
curl -s -X POST \
-H 'Content-Type: application/json-rpc' \
-d " \
{
    \"jsonrpc\": \"2.0\",
    \"method\": \"user.logout\",
    \"params\": [],
    \"id\": 1,
    \"auth\": \"$auth\"
}
" $url
```

Patch infinite componente zabbix

```
class ZabbixThread(threading.Thread):
    """A threaded event handler class."""
    # Rename to TRIES and set to 0
    MAX_TRIES = 3

    def __init__(self, hass, zabbix_sender, event_to_metrics):
        """Initialize the listener."""
        threading.Thread.__init__(self, name="Zabbix")
        self.queue = queue.Queue()
        self.zabbix_sender = zabbix_sender
        self.event_to_metrics = event_to_metrics
        self.write_errors = 0
        self.shutdown = False
        self.float_keys = set()
        self.string_keys = set()

    def setup(self, hass):
        """Set up the thread and start it."""
        hass.bus.listen(EVENT_STATE_CHANGED, self._event_listener)
        hass.bus.listen_once(EVENT_HOMEASSISTANT_STOP, self._shutdown)
        self.start()
        _LOGGER.debug("Started publishing state changes to Zabbix")

    def _shutdown(self, event):
```

```
        """Shut down the thread."""
        self.queue.put(None)
        self.join()

    @callback
    def _event_listener(self, event):
        """Listen for new messages on the bus and queue them for Zabbix."""
        item = (time.monotonic(), event)
        self.queue.put(item)

    def get_metrics(self):
        """Return a batch of events formatted for writing."""
        # Replace MAX_TRIES to TRIES
        queue_seconds = QUEUE_BACKLOG_SECONDS + self.MAX_TRIES * RETRY_DELAY

        count = 0
        metrics = []

        dropped = 0

        with suppress(queue.Empty):
            while len(metrics) < BATCH_BUFFER_SIZE and not self.shutdown:
                timeout = None if count == 0 else BATCH_TIMEOUT
                item = self.queue.get(timeout=timeout)
                count += 1

                if item is None:
                    self.shutdown = True
                else:
                    timestamp, event = item
                    age = time.monotonic() - timestamp

                    if age < queue_seconds:
                        event_metrics = self.event_to_metrics(
                            event, self.float_keys, self.string_keys
                        )
                        if event_metrics:
                            metrics += event_metrics
                    else:
                        dropped += 1

        if dropped:
            _LOGGER.warning("Catching up, dropped %d old events", dropped)

        return count, metrics

    def write_to_zabbix(self, metrics):
        """Write preprocessed events to zabbix, with retry."""
        # while True:
        for retry in range(self.MAX_TRIES + 1):
            try:
```

```
                self.zabbix_sender.send(metrics)
                if self.write_errors:
                    _LOGGER.error("Resumed, lost %d events",
self.write_errors)
                    self.write_errors = 0

                _LOGGER.debug("Wrote %d metrics", len(metrics))
                # Put [self.TRIES = 0]
                break

            except OSError as err:
                # [time.sleep(RETRY_DELAY)] out of [if retry <
self.MAX_TRIES:]
                if retry < self.MAX_TRIES:
                    # Put [self.TRIES += 1]
                    time.sleep(RETRY_DELAY)
                else:
                    if not self.write_errors:
                        _LOGGER.error("Write error: %s", err)
                    self.write_errors += len(metrics)
```