



2 September 2022  
girones96@gmail.com

# CERNphone automated testing

Andreu Girones de la Fuente  
CERN, Building 31/R-020 CH-1211 Geneva, Switzerland

Keywords: Automated testing, VoIP, Puppet, Openstack, Docker

---

## Summary

This document describes the automatization of CERNphone testing. Tests are based on the VOLTS project. Tests are executed each hour, and the state is sent to MONIT, to keep track of the service's status. All tests are running in a virtual machine on top of OpenStack and managed by Puppet. The architecture of the project is composed of four docker containers i.e. prepare, database, vp, and report. In addition, a work environment for developing the tests locally before deploying is provided.

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>State of the Art</b>	<b>3</b>
2.1	VoIP Patrol . . . . .	3
2.2	Puppet . . . . .	4
2.3	CERNphone . . . . .	4
2.3.1	Kamailio . . . . .	5
2.3.2	Asterisk . . . . .	5
<b>3</b>	<b>Tests run</b>	<b>6</b>
3.1	prepare . . . . .	6
3.2	database . . . . .	6
3.3	vp . . . . .	7
3.4	report . . . . .	7
<b>4</b>	<b>Automated testing</b>	<b>7</b>

5	Future Work	9
A	Appendix: prepare configuration	11
B	Appendix: Hiera configuration variables	12

# 1 Introduction

This work had been done during the CERN summer student programme in 2022, in the IT-CS-TR group, under the supervision of Ihor Olkhovskiy.

Regression testing is beneficial for software projects [11]. For instance, it permits a safer development, improves quality, and reduces costs. So it was decided to add it for CERNphone. To do so, a working environment for testing the tests has been implemented [10]. Also, a VM running on top of OpenStack that is continuously testing CERNphone has been deployed [7]. The tests are end-to-end tests, based on the VOLTS project [10]. Where actions can be defined in an XML format e.g. calling a number, as well as, the expected behaviour. Finally, the result tests are stored on logs. Moreover, metrics from the tests are sent to MONIT [2]. In the following sections, it will be discussed the different technologies involved in the project, how we run the tests, and the automatization of those.

## 2 State of the Art

### 2.1 VoIP Patrol

VoIP Patrol is an application that allows performing end-to-end tests. It runs on top of PJSUA2 Fig.1.

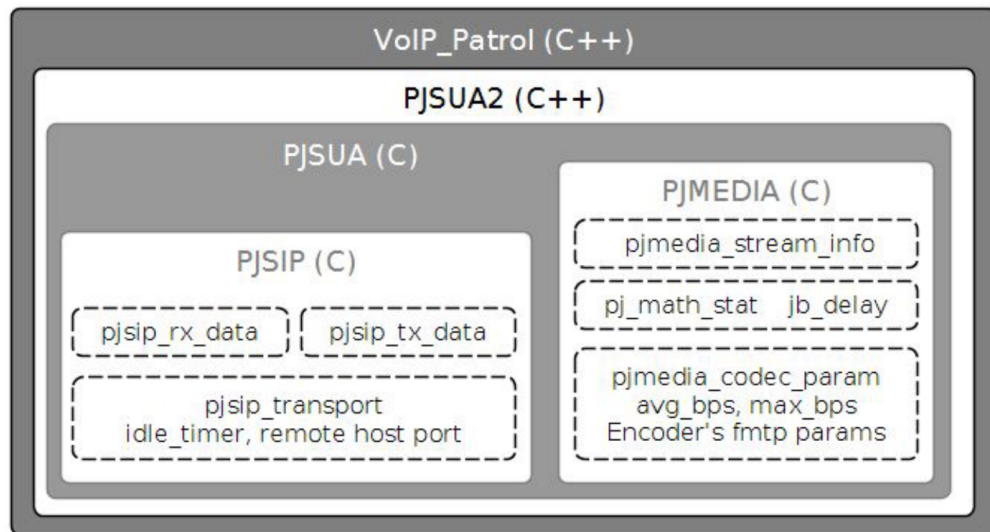


Figure 1: Low level libraries under VoIP Patrol

The tests are defined in an XML file. For each tests distinct actions could be specified e.g. making a call, sending an alert Fig.2.

After performing the tests, a JOSNL file is generated. This file contains the information of all of the performed tests Fig.3.

```

<config>
  <actions>
    <action type="call" label="us-east-va"
      transport="tls"
      expected_cause_code="200"
      caller="15147371787@noreply.com"
      callee="12012665228@target.com"
      to_uri="+12012665228@target.com"
      max_duration="20" hangup="16"
      auth_username="VP_ENV_USERNAME"
      password="VP_ENV_PASSWORD"
      realm="target.com"
      rtp_stats="true"
    >
      <x-header name="X-Foo" value="Bar"/>
    </action>
    <!-- note: param value starting with VP_ENV_ will be replaced by environment variables -->
    <!-- note: rtp_stats will include RTP transmission statistics -->
    <!-- note: x-header tag inside an action will append an header -->
    <action type="wait" complete="true"/>
  </actions>
</config>

```

Figure 2: Making a call XML file example

```

{"scenario": {"state": "start", "name": "/xml/01-register.xml", "time": "31-08-2022 13:33:25"}}
{"1/1": {"label": "Register 88881-31896", "start": "31-08-2022 13:33:25", "end": "31-08-2022 13:33:25", "action": "register", "from": "88881", "to": "88881", "result": "PASS", "result_text": "Main test passed", "expected_cause_code": 200, "cause_code": 200, "cancel_behavoir": "", "reason": "OK", "callid": "", "transport": "TLS", "srtp": "", "peer_socket": "", "duration": 0, "expected_duration": 0, "max_duration": 0, "hangup_duration": 0, "call_info": {"local_uri": "", "remote_uri": "", "local_contact": "", "remote_contact": ""}}}
{"scenario": {"state": "end", "result": "PASS", "name": "/xml/01-register.xml", "time": "31-08-2022 13:33:27", "total tasks": "1", "completed tasks": "1"}}

```

Figure 3: Resultant JSONL file after a test run

## 2.2 Puppet

Puppet permits automating the server management [4]. It has a server-client architecture. On the one hand, there is the puppet server, where is stored the puppet code which defines the desired state of the node. On the other hand is the puppet agent, which is running in each of the nodes Fig.4. Periodically, the puppet agent asks for the puppet code i.e. the manifest, and checks if the desired state and the actual state of the node match. If not it takes the necessary actions to make it match. Leaving the node as the developer has defined.

## 2.3 CERNphone

CERNphone is a telephony solution that replaces CERN's legacy Alcatel fixed telephone line.

The IT department has implemented two CERNphone applications. On one hand, there is CERNphone mobile for Android and iOS. On the other hand, CERNphone desktop is

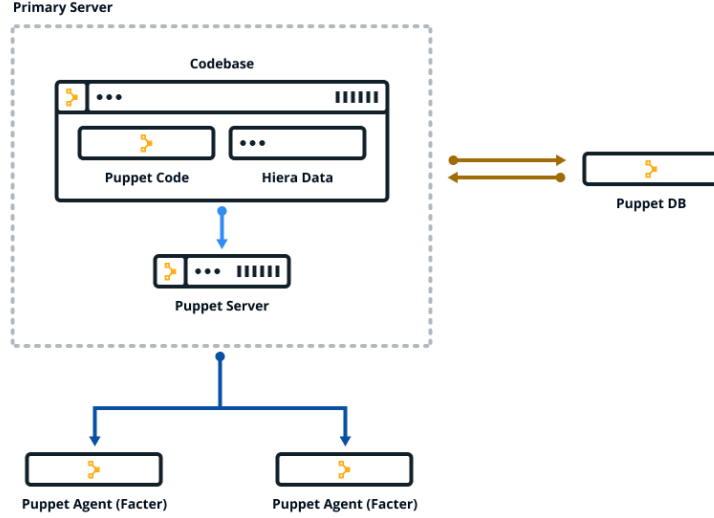


Figure 4: Puppet server-agent architecture

designed to run on Windows, macOS, and Linux. Moreover, IP phones compatible with SIP are also supported.

CERNphone relies on Telephony Open-source Network Evolution (TONE) for routing external calls [6]. However, CERNphone provides its own SIP backend. It is composed of a frontend i.e. Kamailio, and a routing engine i.e. Asterisk Fig.5. Different machines are running those, and are distributed in two different CERN sites i.e. Meyrin and Prévessin.

### 2.3.1 Kamailio

Kamailio is the frontend of the SIP backend. The control communication to the CERNphone mobile travels on top of SIPS and TLS, whereas the media is sent over UDP. For the CERNphone desktop, the WebRTC standard is used to communicate via RTP. In addition, it also manages the authentication of the client.

### 2.3.2 Asterisk

Asterisk is the routing engine of the system. It routes all the calls. There are two different casuistics. A call between CERNphones, which is completely managed by CERNphone Asterisk's servers. And a call between CERNphone and an external number. In this case, CERNphone relies on TONE infrastructure, which is also composed of Kamailio and Asterisk servers. An external phone could be either a phone not using CERNphone or an Alcatel CERN's landline device.

Asterisk also manages all the necessary data for the system to work properly i.e. user's

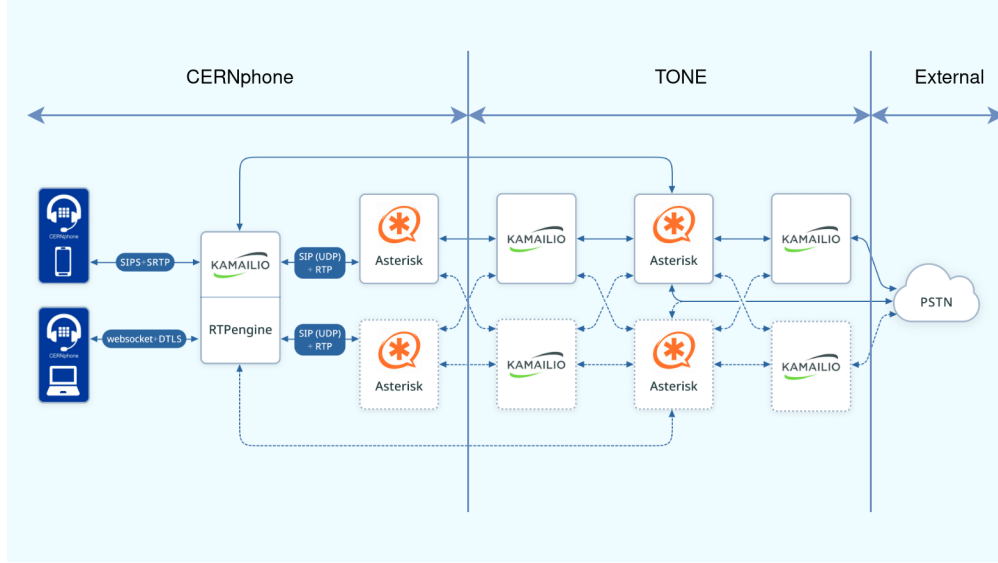


Figure 5: CERNphone external call going through CERNphone’s SIP backed, and TONE

names, phone numbers, team phone numbers, etc. Nevertheless, critical data is managed by the FAP department.

### 3 Tests run

Four docker containers are in charge of running tests i.e. prepare, database, vp, and report [1]. They run in sequential order. In the following sections, you will find in detail the process of reading the test definition, running the test, and obtaining the report.

Tests can be grouped into scenarios. Thus, each scenario contains one or more tests.

#### 3.1 prepare

*prepare* is the first container that takes action. It parses the YAML configuration file and the XML scenario definitions. The YAML configuration file permits to set communication parameters used among different scenarios e.g. user accounts, transport protocol, etc. Furthermore, the XML files follow a Jinja2 template style. In this file, all the tests and database operations are defined. An example of both files is shown in Appendix A.

According to these files, *prepare* generates a unique XML file per scenario. This file contains the needed information for the next steps.

It creates a file *scenarios.done* to certify that all the scenarios are prepared.

#### 3.2 database

The *database* container runs two times for each scenario. In the first one, it parses the prepared XML file and performs all the required database actions for the running scenario

e.g. insertion of a user. Leaving the databases ready for the tests Fig.6. For each action, it logs a JSON into the *database.jsonl* file, and after it will be used to generate the report.

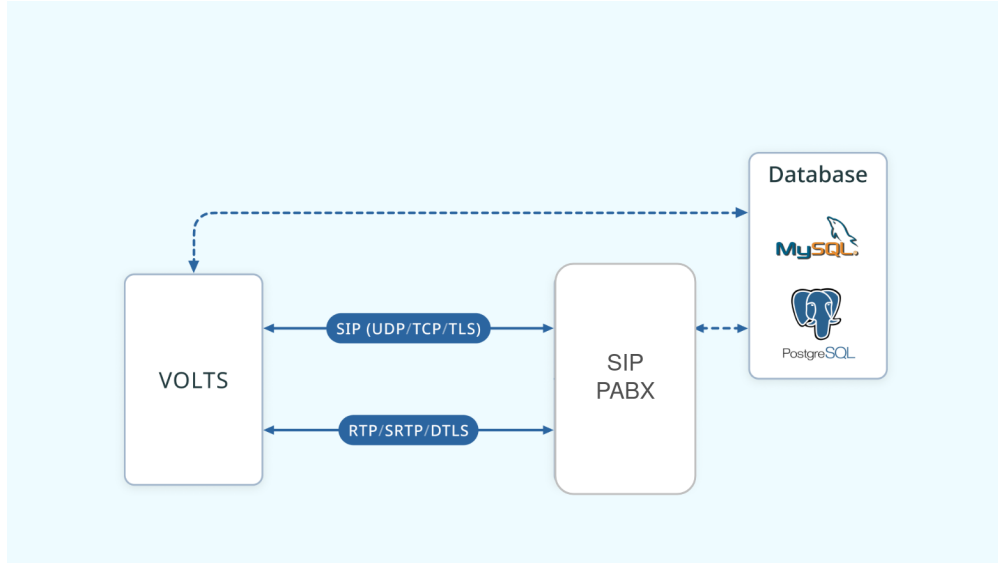


Figure 6: VOLTS typical use case

The second run is triggered when the *vp* container has finished. In this run, all the database operations are reverted. Therefore, the database is cleaned up and left as before the scenario run.

### 3.3 *vp*

*vp* reads the prepared XML and runs the tests in the scenario using VoIP Patrol. At this stage, all the mock actions are performed e.g. making a call. All the results are stored in the *voip\_patrol.jsonl* file. Which is used to generate the report.

### 3.4 report

The *report* container parses the JSONL files from the *vp* and the database containers and generates a report. This report could be printed as a table or as JSON. Furthermore, it processes the results and sends how many tests have failed to MONIT via HTTP. Therefore, it is possible to monitor CERNphone status via MONIT. MONIT offers distinct tools to keep track of the metrics, among them are InfluxDB and Grafana [2].

## 4 Automated testing

OpenStack is the infrastructure-as-a-service used within CERN computing facilities [3]. On top of OpenStack, I did deploy a VM that is performing periodical testing. As this project is planned to be a long-term project, it was decided to use Puppet to manage the state of the VM. Therefore, a new module called VOLTS was created, as well as, a new environment.

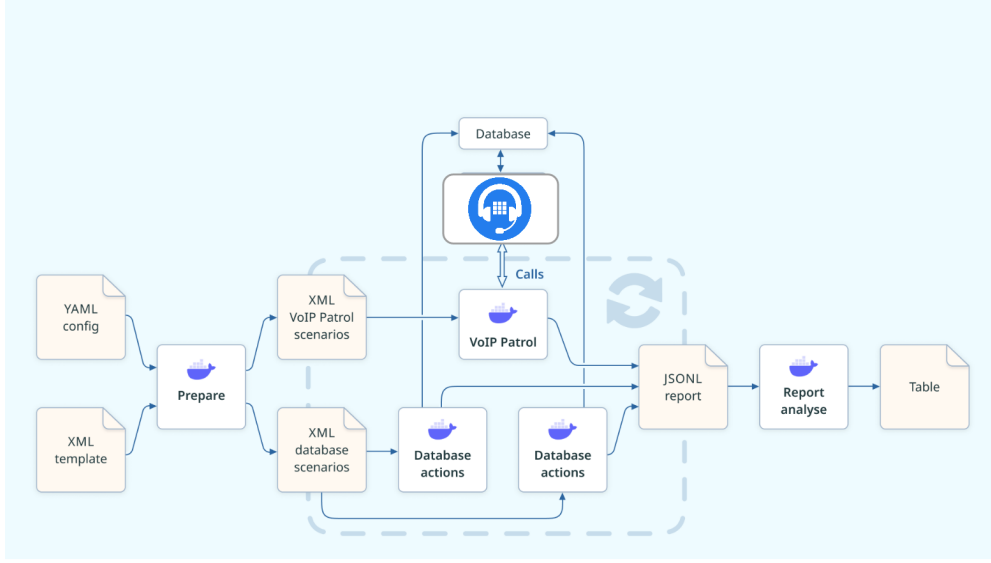


Figure 7: Test run

Furthermore, all the configuration variables are managed by Hiera (see Appendix B). In puppet, I defined the file system according to functional-testing [10], the docker credentials using *Teigi*, a cron job that runs the tests periodically, and the necessary docker images to run the tests.

The running of tests are directed by two scripts i.e. *run.sh* and *run\_and\_log.sh*. Both are located in the */root* directory Fig.8.

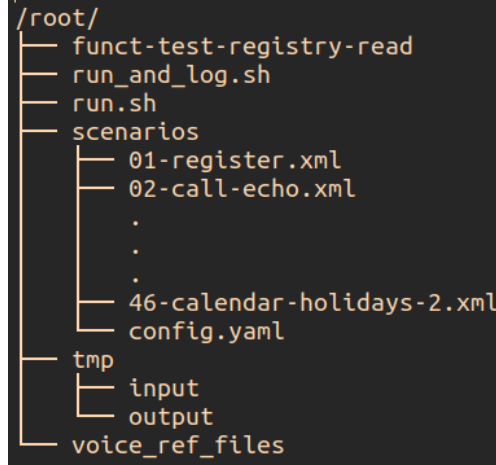


Figure 8: VM */root* directory

All the log files are stored in the */var/log/volts/* directory. For each run, two files are stored i.e. a text file and a JSONL. Both are named following the same pattern i.e. *yy-mm-dd.hh:mm*, corresponding to the start of the run. Nonetheless, the text file containing the logs from the docker containers, and the tests run report. It has the file extension *.log*. And the JSONL file contains the JSONLs produced by the database and vp containers. Moreover, each time a new log is registered all the log files older than 30 days are removed



(see `logs.backup_days` in Appendix B), to limit system memory.

Furthermore, to ensure mutual exclusion between distinct test runs, a lock is used. I decided to create a directory as a lock. Thus, when a tests run begins it tries to create the `/tmp/volts.lock/` directory if it can not because it exists an error message is sent and the run ends. Otherwise, the run executes.

## 5 Future Work

Considering that each scenario is independent of the other. I propose to parallelize the tests. Because it would have a significant impact on the execution time, which currently is 14 minutes for 46 scenarios. In addition, it would have superior scenario scalability.

However, to do so database operations should be changed. Because, the second run of the database container i.e. the cleaning up, could have a non-desired effect on other scenarios.

A possible approach could be to perform all the necessary database operations for all the scenarios, after the prepare container finishes. Then, `vp` can run all the scenarios. Finally, when all the scenarios are run, the cleaning up could be executed.

Another challenge of this proposal is to manage correctly the data used in multiple scenarios e.g. user account. To overcome it, a dynamic generation of user accounts could be implemented.

As the log system is certainly simple it could be improved by using a logging framework e.g. Syslog [5].

Currently, VOLTS support TCP and UDP as transport protocols. I propose to add also WebSocket, due to, nowadays, communication via desktop has become more important [8].

Finally, I propose adding voice quality tests to VOLTS. A potential approach could be the use of Google's ViSQOL [9]. Which permits making a call, reproducing mock audio, and recording the echo. After ViSQOL processes both files i.e. the original mock audio and its echo. And generates a MOS-LQO quality metric.

## References

- [1] Docker manual. <https://docs.docker.com/desktop/>, 2022.
- [2] Monit documentation. <https://monit-docs.web.cern.ch/>, 2022.
- [3] Openstack documentation. <https://clouddocs.web.cern.ch/>, 2022.
- [4] Puppet documentation. <https://config.docs.cern.ch/>, 2022.
- [5] Syslog manual. <https://man7.org/linux/man-pages/man3/syslog.3.html>, 2022.
- [6] Tone documentation. <https://tone.docs.cern.ch/>, 2022.
- [7] Andreu Girones de la Fuente. it-puppet-module-volts. <https://github.com/agirones/it-puppet-module-volts>, 2022.
- [8] Ian Fette and Alexey Melnikov. The websocket protocol. Technical report, 2011.
- [9] Andrew Hines, Jan Skoglund, Anil Kokaram, and Naomi Harte. Visqol: an objective speech quality model. *EURASIP Journal on Audio, Speech, and Music Processing*, 2015 (13):1–18, 2015.
- [10] Ihor Olkhovskiy. functional-testing. <https://gitlab.cern.ch/cernphone/functional-testing>, 2022.
- [11] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen, and Mika V. Mäntylä. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *2012 7th International Workshop on Automation of Software Test (AST)*, pages 36–42, 2012.

## A Appendix: prepare configuration

```
global:
  domain: '<SOME_DOMAIN>'
  transport: 'tls'
  srtp: 'dtls,sdes,force'
  play_file: '/voice_ref_files/8000_12s.wav'
databases:
  'kamdb':
    type: 'mysql'
    user: 'kamailiorw'
    password: 'kamailiorwpass'
    base: 'kamailio'
    host: 'mykamailiodb.local'
accounts:
  '88881':
    username: '88881'
    auth_username: '88881-1'
    password: 'SuperSecretPass1'
  '88882':
    username: '88882'
    auth_username: '88882-67345'
    password: 'SuperSecretPass2'
  '90001':
    username: '90001'
    auth_username: '90001'
    password: 'SuperSecretPass3'
```

Figure 9: Example of a *config.yaml* file

```
<!-- Test simple register -->
<config>
  <section type="database">
    <actions>
      <!-- "kamdb" here is referring to entity in "databases" from config.yaml. "stage" is explained a bit below -->
      <action database="kamdb" stage="pre">
        <!-- what data are we gonna insert into "subscriber" table? -->
        <table name="subscriber" type="insert" cleanup_after_test="true">
          <field name="username" value="{{ a.88881.username }}" />
          <field name="domain" value="{{ c.domain }}" />
          <field name="password" value="{{ a.88881.password }}" />
        </table>
      </action>
    </actions>
  </section>
  <section type="voip_patrol">
    <actions>
      <action type="register" label="Register {{ a.88881.label }}"
        transport="{{ a.88881.transport }}"
        <!-- Account parameter is more used in receive call on this account later -->
        account="{{ a.88881.label }}"
        <!-- username would be a part of AOR - < sip:username@realm -->
        username="{{ a.88881.username }}"
        <!-- auth_username would be used in WWW-Authenticate procedure -->
        auth_username="{{ a.88881.auth_username }}"
        password="{{ a.88881.password }}"
        registrar="{{ c.domain }}"
        realm="{{ a.88881.realm }}"
        <!-- We're expecting get 200 code here, so REGISTER is successful -->
        expected_cause_code="200"
      </action>
      <!-- Just wait 2 sec for all timeouts -->
      <action type="wait" complete="true" ms="2000"/>
    </actions>
  </section>
</config>
```

Figure 10: Example of a XML scenario file, where database actions and tests are defined

## B Appendix: Hiera configuration variables

Hiera variable	
report_type	Supported options: <b>table</b> , <b>table_full</b> , <b>json</b> , and <b>json_full</b>
monit_address	Target address to send data to MONIT team
threshold_degraded	Indicates the threshold of failed tests needed to show the application as degraded
threshold_unavailable	Indicates the threshold of failed tests needed to show the application as unavailable
cron_min	Regulates the frequency of testing (cron notation)
logs_backup_days	How long the logs would be stored (in days)
is_send_to_monit	Specifies if the data should be sent to MONIT
docker_repository	Docker repository where the images are pulled from
email	Email address provided to MONIT team
docker_username	Docker username for pulling the images from the registry
vp_result_file_name	Results file name of vp container
db_result_file_name	Results file name of database container