Falconieri: Remote Provisioning Service as a Service

A new, modern, open source and cloud native remote provisioning service gateway.

nethesis

Matteo Valentini

@_Amygos

FOSDEM'20

Intro: Remote Provisioning Service Theory

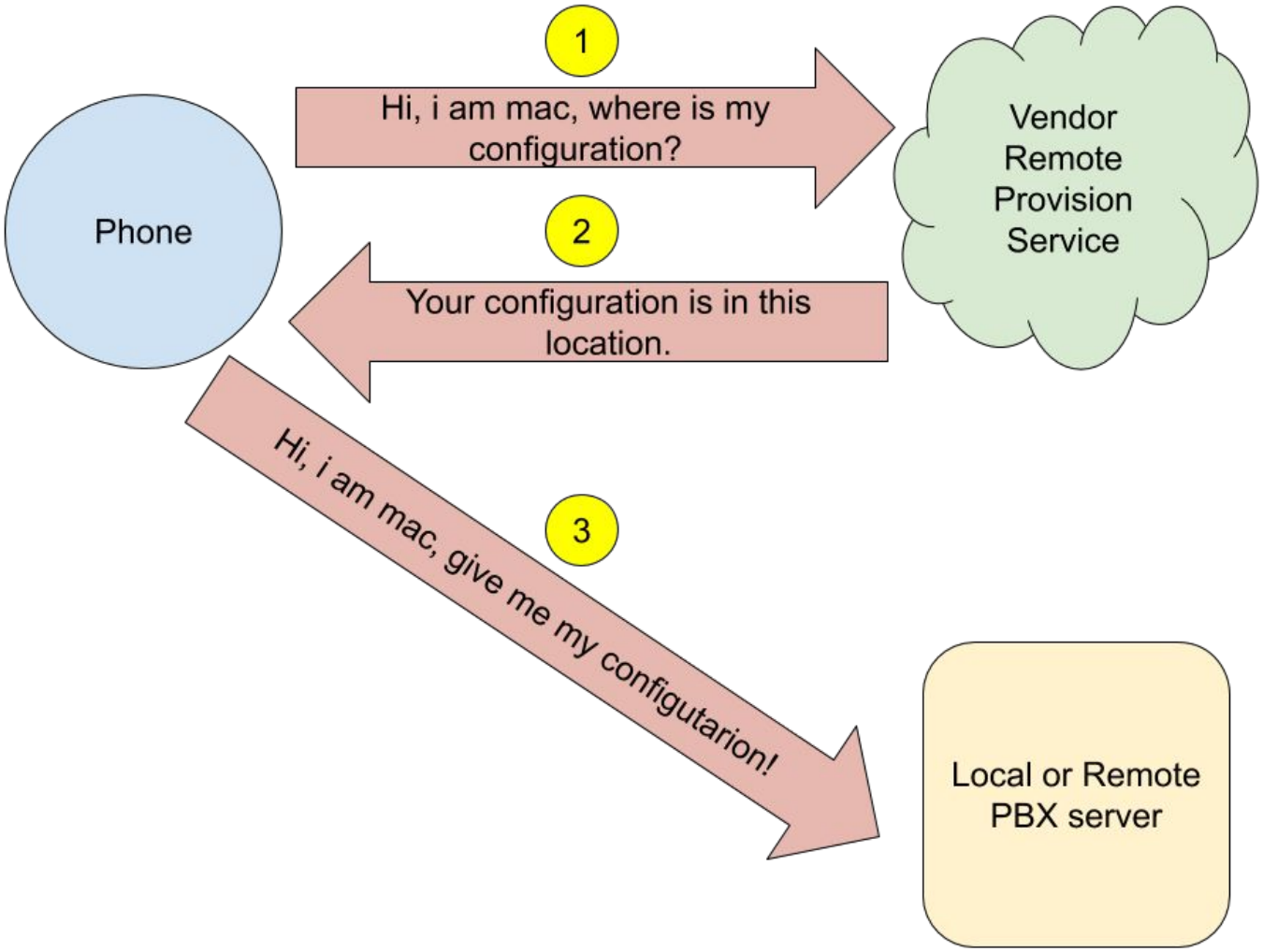
Intro: What is it a Remote Provisioning Service?

The scope of Remote Provisioning Service is to solve the problem of the first time phone configuration.

Without a RPS the phone must rely on local mechanism for initial provisioning, like:

- DHCP Option 66
- UPnP

Intro: What is it a Remote Provisioning Service?

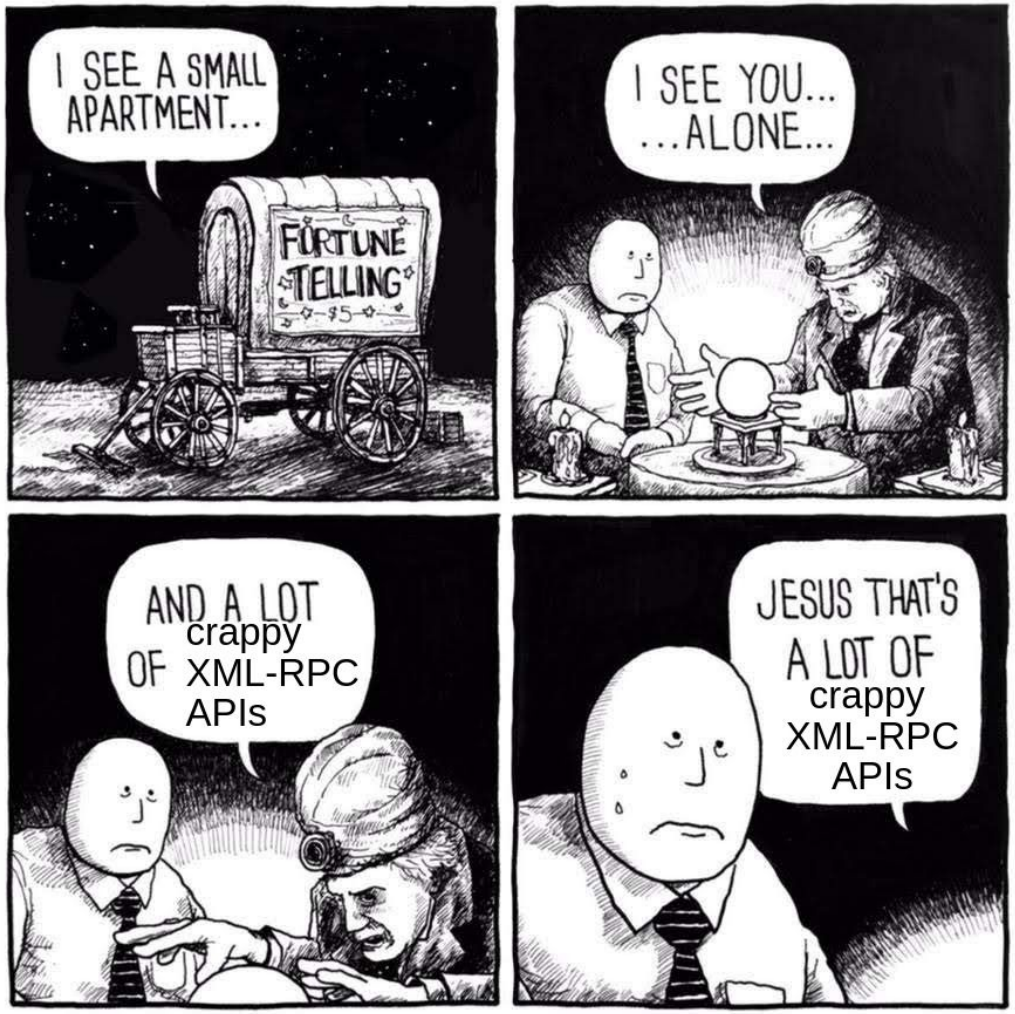


Intro: What can do a Remote Provisioning Service?

- Assign a configuration to a device even before is out of the box
- Massive configuration of multiple device via APIs

Why building a RPS gateway?

Why: Vendors implementations



Why: Vendors implementations

- Not standard set of features between vendors
- Different APIs each vendors
- XML-RPC

Why: The Leopard project

The scope of the project is refactoring the phone provisioning component of NethVoice, the Nethesis PBX solution.

With these goals:

- Use most modern technologies
- Introduction of new provisioning mechanisms (like RPS)
- Support of a well defined set of selected phone vendors
  - SNOM
  - Gigaset
  - Yealink
  - Fanvil
- Release most of the project’s components as Open Source projects

Falconieri is one of the first components released as Open Source

Why: The role of Falconieri

The role of Falconieri is to:

- Provide a unified HTTP rest interface to the vendors RPS service
- Store the credentials for access to the vendors RPS services

The vendors APIs

The Good, the Bad and the Ugly

(Fanvil, Gigaset, SNOM, Yealink)

The vendors APIs: the semantic

For every vendor we want create an API that:

- Given a specific mac address, ***create a new configuration*** for that mac address if the mac address is not already configured
- Given a specific mac address, ***override the previous configuration*** for that mac address if the mac address was already configured

The vendors APIs: the Good

- SNOM
  - Good documentation
    - https://service.snom.com/display/wiki/XML-RPC+API
  - Simple APIs
    - 7 APIs
  - HTTPS endpoint

The vendors APIs: the Good (SNOM)

Api calls for implementing Falconieri semantic:

1. `redirect.registerPhone(mac, provisioningUrl)`

The vendors APIs: the Bad (Gigaset)

- Gigaset
  - Public documentation
    - https://teamwork.gigaset.com/gigawiki/display/GPPPO/Gigaset+Redirect+server
    - Better documentation in the service portal (after obtained a user/password from Gigaset)
  - Simple APIs
    - 7 APIs
  - HTTPS endpoint

Why the Bad?

- Require a CRC code within the mac
- The CRC code in printed in the phone label (whit no public formula for calcualtion)
- The mandatory CRC code make almost impossible an automated device discovery and configuration.

But maybe you can have the CRC code disable for your account if you ask.

The vendors APIs: the Bad (Gigaset)

Api calls for implementing Falconieri semantic:

1. `autoprov.deregisterDevice(macID)`
   - **`macID`**: ”<MAC address> - <CRC code>”
   - We don’t care about success or not!
2. `autoprov.registerDevice(macID, provisioningUrl, Provider)`
   - **`Provider`**: in this case can be anything

The vendors APIs: the Ugly

- Yealink
- Fanvil

The vendors APIs: the Ugly (Yealink)

Yelink

- Pubblic documentation
  - http://support.yealink.com/documentFront/forwardToDocumentDetailPage?documentId=257
- Too many APIs
  - 16 APIs
- HTTPS endpoint

Why in the ugly?

- The APIs are overloaded and redundant.
- Very bad API design

The vendors APIs: the Ugly (Yealink)

Api calls for implementing Falconieri semantic:

1. `redirect.registerDeviceWithUniqueUrl(mac, serverName, provisioningUrl, isOverride)`
   - **`serverName`**: in this case can be anything, `provisioningUrl` take the precedence
   - **`isOverride:`** if 1 override the previous configuration

The vendors APIs: the Ugly (Fanvil)

Fanvil:

- No public documentation!
- Too many APIs!
  - 19 APIs!
- HTTP endpoint…

Why the Ugly

- No HTTPS, require a double hash of the password for the authentication `(md5(md5(password)))`!
- Too many steps to implement the simple Falconieri semantic.

The vendors APIs: the Ugly (Fanvil)

1. `redirect.addServer(serverName, provisioningUrl)`
   - The **`serverName`** and **`provisioningUr`** actually are the same
   - Don’t care if the Server already exist
2. `redirect.deRegisterDevice(mac)`
   - Don’t care about the success.
3. `redirect.registerDevice(mac, serverName)`

Falconieri

Falconieri APIs

PUT /providers/:provider/:mac

Path variables

- ***provider***: Name of the remote provider.
- ***mac***: Mac address of the device, represented in the EUI-48 IEEE RA

Query parameters

- ***crc***: mac address CRC code, only valid with Gigaset provider.

Body

A JSON object with the url field:

- ***url***: URL of configuration server.

Falconieri Usage

```
Usage of ./falconieri:

  -c string

  Path to configuration file (default "/opt/falconieri/conf.json")
```

Falconieri configurations

Falconi can be configured in two way:

- JSON file
- Environment Variables

The configuration passed via environment variables **take the precedence**.

Falconieri JSON configuration

```
{
  "providers": {
    "snom": {
      "user":"user",
      "password": "password",
      "rpc_url":
"https://secure-provisioning.snom.com:8083/xmlrpc/",
       "disable": false
    }
}
```

Falconieri characteristics

- Opensource (AGPL v3)
- Single Go Lang binary
- Easily deployment with provided ansible role.
- Created with “*12 factor app*” in mind
- Stateless
- Easily vertically and horizontally scalable

Falconieri TODOs

- Client authentication
- Configuration of a list of devices
- More deployment strategy: RPM, DEB, Docker, ELM ecc..
- Deletion APIs?

Every Pull Request, enhancement, critique are very welcome!

https://github.com/nethesis/falconieri

Thanks for listening!
Questions?

Matteo Valentini

Developer @ Nethesis (mostly Infrastrutture Developer)

 Amygos

 @_Amygos

 amygos@paranoici.org, matteo.valentini@nethesis.it