



Cockpit

Intégration via API

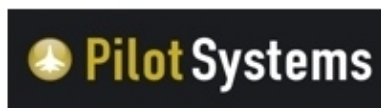


Table des matières

1. Révisions du document.....	1
2. Présentation générale.....	2
3. Configuration SQL.....	4
4. Effectuer un appel RPC.....	6
5. API actuelle.....	9
6. Quelques cas d'utilisations.....	13
7. Exemples d'appels.....	14

1. Révisions du document

Historique des révisions du document

Version	Date	Remarques
1.1.0	19/07/2010	Ajout de l'appel <code>del_tag</code> et de paramètres à <code>add_tag</code> .
1.0.1	05/11/2009	Ajout de paramètres à <code>send_one</code> , précisions sur l'importance de la clé.
1.0	02/11/2009	Version initiale.

2. Présentation générale

Objectifs du système

Les objectifs du système de RPC de Cockpit sont les suivants :

- Permettre une granularité très fine de la sécurité (pouvoir autoriser la création de contacts uniquement avec certains tags, ...)
- Permettre une traçabilité des appels (quelles sont les appels autorisés, avec quels paramètres, combien de fois ont-ils été effectués, ...)
- Permettre une invalidation et une modification centralisée des appels (changer le tag ajouté à la création, ...)
- Avoir une sécurité forte, sans obliger l'utilisation d'https (bien qu'il soit possible de l'utiliser aussi)
- Ne pas trop complexifier la création d'une "télécommande" capable de contrôler beaucoup de fonctionnalités d'un coup.

Terminologie

- **RPC** : *remote procedure call* : appel, à distance, d'une procédure effectuant un traitement sur un serveur ;
- **JSON** : *javascript object notation* : format d'échange de données, simple et générique, standardisé sur <http://tools.ietf.org/html/rfc4627> ;
- **API** : *application programming interface* : dans le cas de Cockpit, la liste et description des appels (ou RPCs) possibles ;
- **Règle RPC** : règle, stockée dans la base SQL, qui autorise ou non des RPCs avec certains paramètres ;
- **Clé secrète RPC** : clé secrète, unique pour chaque **règle RPC** permettant de signer les RPCs ;
- **Signature RPC** : signature cryptographique correspondant un RPC en particulier avec des paramètres fixés ;
- **REST** : *representational state transfer* : architecture de conception de systèmes distribués, reposant le plus souvent sur HTTP, dont Cockpit s'inspire.

Fonctionnement général

Dans la base de données SQL (administrée via l'interface Web ou directement en SQL), sont stockées des règles RPC. Ces règles sont identifiées par un numéro unique, et associées à une clé secrète. Elles décrivent quels sont les appels autorisés via cette clé secrète, avec quels paramètres, ...

Un RPC sur Cockpit s'effectue en utilisant une requête GET ou POST sur une URL du type :

```
http://<cockpit>/rest/<module>/<fonction>
```

Cette requête doit contenir au minimum, dans les paramètres GET :

- **id** : l'identifiant d'une règle RPC ;
- **key** : une signature, calculée par l'appelant.

Tous les autres paramètres de la requête sont envoyés, en GET ou en POST (de préférence en POST) en JSON.

Si la fonction est autorisée par la règle RPC, elle est alors exécutée, et une réponse en JSON est renvoyée. En cas d'erreur, un code d'erreur HTTP est renvoyé.

3. Configuration SQL

Présentation générale

Les appels autorisés doivent être configurés dans la base SQL. Deux séries de fonctionnalités sont fournies à ce niveau :

1. Un contrôle d'accès, fin, qui permet d'autoriser certaines fonctions uniquement, voir avec certains paramètres uniquement.
2. La spécification de valeurs par défaut aux paramètres.

Pour chaque règle, une liste de paramètres est associée. Chacun des paramètres peut être :

- **free** (libre) : l'appelant peut utiliser la valeur qu'il veut ;
- **filtered** (filtré) : l'appelant peut spécifier uniquement certaines valeurs (regexp) ;
- **fixed** (fixé) : la valeur du paramètre est fixée par la règle, l'appelant ne peut pas la modifier.

Détails de la table RpcRule

La table RpcRule contient les champs suivants :

Champ	Type	Description
id	int	Identifiant, unique, autoincrémenté
key	string	Clé secrète associée à cet appel. Attention , cette clé doit être longue et aléatoire, sur le modèle de la chaîne aléatoire de 64 caractères générée par défaut.
module	string	Regexp filtrant les modules autorisés
fonction	string	Regexp filtrant les fonctions autorisés
counter	int	Compteur du nombre d'appels à la règle RPC
active	boolean	Permet de désactiver temporairement une règle en le mettant à Faux
created	date	Date de création de la règle
last_used	date	Date de dernière utilisation de la règle

Détails de la table RpcParam

La table RpcParam contient les champs suivants :

Champ	Type	Description
id	int	Identifiant, unique, autoincrémenté
rule_id	int	Clé étrangère vers un RpcRule
param	string	Nom du paramètre
state	enumeration	Une valeur parmi : <i>free</i> (libre), <i>filtered</i> (filtré), <i>fixed</i> (fixé)
value	string	Regexp de contrôle pour un paramètre filtré, valeur à utiliser pour un paramètre fixé, inutilisé pour un paramètre libre
default	string	Valeur par défaut si le paramètre n'est pas précisé par l'appelant, pour des paramètres libres ou filtrés.

Notes :

- tous les paramètres non présents dans `RpcParam` sont considérés comme étant libres, la valeur par défaut étant celle fournie par l'implémentation du RPC ;
- dans le cas d'un filtre sur un paramètre multivalué (type "liste des tags"), le filtre est appliqué sur chaque valeur possible ;
- les `value` et `default` peuvent être donné sous forme de JSON pour des paramètres compliqués.

4. Effectuer un appel RPC

Procédure générale

La procédure générale est la suivante :

1. Trouver l'URL correspondant à la fonction à appeler, du type :

```
http://<cockpit>/rest/<module>/<fonction>
```

2. Obtenir un identifiant de règle RPC ainsi que la clé secrète associée.
3. Effectuer un appel GET ou POST sur l'URL, en précisant :
 - ◆ l'identifiant de la règle en paramètre GET (*query string*) `id` ;
 - ◆ les autres paramètres de l'appel en valeurs POST ou GET, encodées en JSON ;
 - ◆ une signature cryptographique de la requête, dans le paramètre GET `key`.

Signature cryptographique

Principe de fonctionnement

La signature cryptographique doit être un hash SHA-1 de la chaîne suivante :

```
<fonction>--<GET>--<POST>--<clé>
```

Les éléments étant :

- fonction : le nom de la fonction appelée (module/fonction) ;
- GET : la chaîne de paramètres GET (hors le paramètre `key` bien sûr) ;
- POST : la chaîne de paramètres POST ;
- clé : la clé secrète de la règle RPC.

Exemple simple

Par exemple, si on souhaite appeler la fonction `rpc/version` (fonction simple indiquant la version de l'API), autorisée par la règle RPC numéro 2 dont la clé secrète est `zeezikeeL8ec5eiz0Eishab6ecuXeik5`, on doit générer un SHA1 de :

```
rpc/version-id=2--zeezikeeL8ec5eiz0Eishab6ecuXeik5
```

Ce qui donne `53e560d83052b5e3abf7f2365f8720bbdd285cdc`. On doit donc ensuite appeler l'URL :

```
http://<cockpit>/rest/rpc/version?id=2&key=53e560d83052b5e3abf7f2365f8720bbdd285cdc
```

Exemple plus compliqué

Si on souhaite, avec la même règle RPC, appeler la fonction `cockpit/add_contact` avec un `primaryemail` à `arthur.dent@h2g2.com`, un `firstname` à Arthur, un `lastname` à Dent, et les tags Terrien et Anglais, les paramètres POST, une fois en JSON et sous la forme `x-www-form-urlencoded` sont :

```
lastname=%22Dent%22&primaryemail=%22arthur.dent%40h2g2.org%22&firstname=%22Arthur%22&\ntags=%5B%22Terrien%22%2C%22Anglais%22%5D
```

La valeur sur laquelle calculer le SHA est donc :

```
cockpit/add_contact-id=2-lastname=%22Dent%22&primaryemail=%22arthur.dent%40h2g2.org%22&\nfirstname=%22Arthur%22&tags=%5B%22Terrien%22%2C%22Anglais%22%5D-\nzeezikeeL8ec5eiz0Eishab6ecuXeik5
```

Ce qui donne un SHA de a9d001d6c7b268adfeab986029986d63b30e42c7 et donc une URL finale de (les autres paramètres étant en POST) :

```
http://<cockpit>/rest/cockpit/add_contact?id=2&key=a9d001d6c7b268adfeab986029986d63b30e42c7
```

Utilisation d'une en-tête

Il est aussi possible, au lieu d'inclure la signature dans les paramètres GET (la *query string*) de spécifier un header HTTP nommé `X-Cockpit-Signature` qui contient cette même signature. L'intérêt principal est d'éviter que la signature n'apparaisse dans les logs.

GET ou POST ?

Les deux paramètres `id` et `key` doivent être dans le GET. Les autres paramètres peuvent être mis dans les données POST (sous la forme `x-www-form-urlencoded` uniquement), ou en paramètres GET.

Il est conseillé de ne pas envoyer de valeurs trop longue ou trop compliquée dans les paramètres GET, et d'utiliser les paramètres POST dans ces cas là.

Autrement, Cockpit ne fait aucune différence entre le GET et le POST, mais il est aussi important de se souvenir qu'un POST ne doit pas être caché, tandis qu'un GET peut l'être. Toute action qui effectue des modifications devrait donc être un POST.

Interprétation intelligente des paramètres

L'API officielle indique les paramètres doivent être en JSON. Cependant Cockpit est intelligent, et est capable de comprendre des paramètres plus simples :

- les chaînes de caractères sans "" seront acceptées ;
- les listes peuvent être données sur une forme plus simple, comme une simple chaîne séparée par des virgules : dans l'exemple précédant les deux formes suivantes sont équivalentes :

```
tags=[ "Humain", "Anglais" ]
```

```
tags=Humain,Anglais
```

Attention cependant, si la chaîne contient des caractères spéciaux, il est nécessaire de l'encoder en JSON, sinon le résultat n'est pas garanti.

Paramètres supplémentaires

Des paramètres supplémentaires, optionnels, sont supportés par tous les RPCs :

- *redirect* : `string` : si ce paramètre est présent, au lieu de la réponse normale, Cockpit effectue un HTTP Redirect vers l'URL donnée. Ceci permet d'utiliser l'API en passant par le navigateur de l'utilisateur, au lieu de faire un appel serveur à serveur (par exemple si l'appel est interdit par l'hébergeur, ou si l'extension PHP curl ne peut pas être installée). La redirection est effectuée avec un code 302, que la plupart des navigateurs interprètent comme "effectuer une requête GET sur la nouvelle URL", et ne font donc pas de nouvelle soumission du formulaire POST.

Valeur de retour

En cas de succès, une chaîne JSON est renvoyée dans un HTTP 200 OK, contenant des informations dépendant de l'appel.

En cas d'erreur diverses (signature invalide, paramètres manquants, erreur interne), un code HTTP d'erreur est renvoyé, et une page HTML contenant une description plus ou moins détaillée de l'erreur.

5. API actuelle

Note générale

L'API de Cockpit est construite à la volée, suivant les cas d'utilisation pratiques qui nous sont présentés, et ceci afin de s'adapter au mieux au besoin. Les appels sont donc peu nombreux pour l'instant, mais la liste est destinée à s'accroître rapidement.

Attention la liste des erreurs possibles n'est pas forcément exhaustive, elle ne couvre que les erreurs les plus fréquentes.

Dans tous les cas, les erreurs suivantes peuvent être renvoyées :

- HTTP 404 Not Found : fonction non existante, règle RPC non existante ;
- HTTP 400 Bad Request : `id` ou `key` manquant, règle RPC non conforme à la fonction appelée ;
- HTTP 403 Forbidden : signature invalide, règle RPC désactivée, paramètres non autorisés par la règle RPC.

Module `rpc`

Le module `rpc` contient des fonctions générales d'introspection.

Fonction `version`

URL

`rpc/version`

Description

Renvoi le numéro de version de l'API.

Paramètres

Aucun

Valeur de retour

Un triplet (majeur, mineur, bugfix)

Fonction `list_modules`

URL

`rpc/list_modules`

Description

Retourne la liste des modules de Cockpit fournissant des RPCs.

Paramètres

Aucun

Valeur de retour

Une liste de modules.

Fonction `list_functions`

URL

`rpc/list_functions`

Description

Renvoi la liste des fonctions d'un ou plusieurs modules.

Paramètres

- `modules : list` : liste des modules, par défaut tous les modules.

Valeur de retour

Un tableau associatif, indiquant pour chaque module la liste des fonctions disponibles.

Module `cockpit`

Fonction `add_tag`

URL

`cockpit/add_tag`

Description

Ajoute une tag à un contact (identifié par son email), créant le contact s'il n'existait pas.

Paramètres

- `email : string` : email du contact.
- `tags : list` : tags à utiliser.
- `having_tags : 'list'` : n'affecte/ne regarde que les contacts ayant ce(s) tag(s)-là.

Valeur de retour

Un couple (`mode` , `id`), `mode` indique si le contact a été créé (`created`) ou modifié (`changed`), et `id` indique l'identifiant du contact créé/modifié.

Précisions

- Si plusieurs contacts partagent le même email, tous les contacts sont modifiés, mais l'id retourné peut être celui de n'importe quel contact, sans aucune garantie de stabilité (le même appel peut retourner des contacts différents).
- Seul le mail principal est considéré, pas les mails secondaires.
- Le paramètre `having_tags` n'existe qu'à partir de la version 1.1.0 de l'API.

Fonction `del_tag`

URL

`cockpit/del_tag`

Description

Supprime une tag à un contact (identifié par son email).

Paramètres

- `email : string` : email du contact.
- `tags : list` : tags à supprimer.
- `having_tags : 'list'` : n'affecte/ne regarde que les contacts ayant ce(s) tag(s)-là.

Valeur de retour

Une chaîne de caractère mode, valant `changed` si au moins un contact a été modifié, et `notexisting` sinon.

Précisions

- Si plusieurs contacts partagent le même email, tous les contacts sont modifiés.
- Seul le mail principal est considéré, pas les mails secondaires.
- N'existe qu'à partir de la version 1.1.0 de l'API.

Fonction `add_contact`

URL

`cockpit/add_contact`

Description

Créé un nouveau contact, avec une liste de tags et d'attributs. Un mécanisme de détection des doublons peut être spécifié, et dans ce cas, des tags supplémentaires ajoutés au contact.

Paramètres

- `tags : list` : tags à utiliser.
- `extra_tags : list` : tags supplémentaires à utiliser (la présence de deux paramètres permet d'en définir un comme `fixed` et l'autre comme `free` ou `filtered`, permettant de forcer certains tags, tout en laissant l'appelant préciser les autres).

- *duplicate_keys* : list : liste des champs qui doivent être communs à deux contacts pour que ces contacts soient considérés comme des doublons.
- *duplicate_tags* : list : tags supplémentaires qui seront ajoutés si un doublon est trouvé.
- tous les attributs d'un contact Cockpit.

Valeur de retour

Un couple (*mode*, *id*), *mode* est toujours `created`, et *id* indique l'identifiant du contact créé.

Module newsletter

Fonction `send_one`

URL

`newsletter/send_one`

Description

Envoie une newsletter à un contact.

Paramètres

- *nl_id* : int : identifiant de la newsletter.
- *nl_email* : string : email du contact.
- *firstname* : string : prénom du contact.
- *lastname* : string : nom du contact.
- *filter_duplicate* : bool : désactivée par défaut, cette option permet de filtrer les doublons,

Valeur de retour

Un triplet (*action*, *nl_id*, *nl_email*), *action* étant :

- `sent` en cas de succès ;
- `duplicate` en cas de doublon, si le filtre est activé ;
- `unsubscribed` si le contact est désinscrit.

Erreurs possibles

- HTTP 400 Bad Request : des paramètres (comme le nom et le prénom) utilisés dans le template de newsletter n'ont pas été spécifiés ;
- HTTP 404 Not Found : newsletter spécifiée inexistante.

6. Quelques cas d'utilisations

Télécommande

Le mode "télécommande" est le mode où on fait confiance totalement à une application (par exemple un "client lourd", ou une application embarquée, ...) pour contrôler le cockpit à distance.

Dans ce cas, on crée une seule règle RPC, autorisant toutes les fonctions de tous les modules, avec `.*` et `.*` dans les champs appropriés. Tous les paramètres sont laissés libres.

Ajout de contacts

Un autre cas d'utilisation typique est autoriser l'ajout de contacts par des personnes tierces, en qui on a plus ou moins confiance. On a plusieurs options dans ce cas, données par ordre incrémental de contrôle ::

1. Autoriser le RPC `cockpit/add_contact` avec tous les paramètres en libre, ce qui permet à l'appelant de créer le contact exactement comme il le souhaite.
2. Autoriser le RPC mais en fixant les paramètres de détection de doublon (`duplicate_keys` et `duplicate_tags`). Ainsi, l'appelant n'a pas à les préciser, et on est sûr que la détection de doublons est activée.
3. Fixer aussi l'un des deux paramètres de tags (`tags` ou `extra_tags`), afin que les contacts créés depuis cette source aient des tags imposés, pour permettre une revue manuelle ultérieure, par exemple.
4. Filtrer l'autre paramètre de tags, par exemple, pour n'autoriser que les tags commençant par `Conférence`.

D'autres options sont possible, comme imposer certains autres champs (commentaires par exemple), contrôler totalement les tags, ...

7. Exemples d'appels

Python

Voici un exemple de code en Python pour créer un `rpc broker` générique sur Cockpit, ainsi que des exemples d'utilisation du `rpc broker` :

```
# Cockpit RPC broker, Copyright (C) 2009 Pilot Systems
# http://www.pilotsystems.net
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 US

import urllib2, urllib, json, copy
import sha

class Handler(object):
    """
    RPC Handler
    """
    def __init__(self, url, key, rpc_id, use_get = False, use_header = True):
        """
        Constructor
        """
        self.url = url
        self.rpc_id = rpc_id
        self.path = ''
        self.key = key
        self.use_get = use_get
        self.use_header = use_header

    def __call__(self, **kwargs):
        """
        Call with flexible parameters
        """
        request = self.prepare(kwargs)
        f = urllib2.urlopen(request)
        return json.read(f.read())

    def __getattr__(self, value):
        """
        Allow pretty Python-like usage
        """
        # Warning, don't catch _ and __ special functions
        if value.startswith('_'):
            raise AttributeError, value
        path = self.path
        if path and not path.endswith('/'):
            path = path + '/'
```

```

        path += '/'
    path += value
    # Use copy so we don't have to give again all parameters
    hdlr = copy.copy(self)
    hdlr.path = path
    return hdlr

def make_key(self, *args):
    """
    Make a SHA hash of args
    """
    key = "-".join([ str(s) for s in args ] + [ self.key ])
    return sha.sha(key).hexdigest()

def prepare(self, params):
    """
    Prepare data (return URL and POST data)
    """
    args = [ (k, json.write(v)) for k,v in params.items() ]
    data = urllib.urlencode(args)
    getargs = urllib.urlencode({ "id": self.rpc_id })
    if self.use_get:
        getargs += '&' + data
        data = ''
    key = self.make_key(self.path, getargs, data)
    if self.use_header:
        headers = { 'X-Cockpit-Signature': key }
    else:
        getargs += '&key=%s' % key
        headers = {}

    url = "%s/%s?%s" % (self.url, self.path, getargs)
    request = urllib2.Request(url, data or None, headers)
    return request

KEY = "quo8Ahbooh9ahQuuVoo6si8uchewah6t"
ID = 1
URL = "http://localhost:8000/rest/"

handler = Handler(URL, KEY, ID, use_header = False)
print handler.rpc.version()
handler.use_header = True
print handler.rpc.list_modules()
print handler.rpc.list_functions()
print handler.rpc.list_functions(modules = [ "cockpit" ])
handler.use_get = True
print handler.rpc.list_functions(modules = [ "cockpit" ])
handler.use_get = False

print handler.cockpit.add_contact(primaryemail = 'arthur.dent@h2g2.org',
                                 firstname = 'Arthur', lastname='Dent',
                                 tags = [ 'Terrien', 'Anglais' ], )

```

PHP

Voici un exemple simple de code en PHP. Il est moins complet que celui en Python, mais est fonctionnel :

```

<?php
/*
    Cockpit RPC broker, Copyright (C) 2009 Pilot Systems

```

<http://www.pilotsystems.net>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 US
*/

```
class Handler
{
    public $url, $id, $key;

    public function __construct($url, $key, $id)
    {
        $this->url = $url;
        $this->id = $id;
        $this->key = $key;
    }

    /*
    Build a sha signature
    */
    public function make_key($args)
    {
        $s = '';
        foreach ($args as $arg)
            $s = $s . $arg . '-';
        $s = $s . $this->key;
        return hash("sha1", $s);
    }

    /*
    Build cockpit URL, using everything into GET
    */
    public function build_cockpit_url($path, $values)
    {
        $url = $this->url . '/' . $path;
        $values['id'] = $this->id;
        $args = http_build_query($values);
        $key = $this->make_key(array($path, $args, ''));
        return $url . '?' . $args . '&key=' . $key;
    }

    /*
    Build an URL to add a contact. Values must be an associative array,
    containing some from the $COCKPIT_ALLOWED_KEYS.
    */
    public function build_add_contact_url($values)
    {
        return $this->build_cockpit_url('cockpit/add_contact', $values);
    }
}
```

```

/*
 * Do a CURL call
 */
public function curl_call($url)
{
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    $output = curl_exec($ch);
    curl_close($ch);
    return $output;
}

/*
Perform the add contact query, using CURL extension.
*/
public function add_contact_curl($values)
{
    return $this->curl_call($this->build_add_contact_url($values));
}

/*
Perform add contact through a double redirect. This is will always work.
You *must* provide a "redirect" URL in $values, or else the user will
see a "Contact created, with id 57687" page and that's it.
*/
public function add_contact_redirect($values)
{
    header('HTTP/1.1 303 See Other');
    header('Location: ' . $this->build_add_contact_url($values));
}

/*
Build a send newsletter URL
*/
public function build_send_newsletter_url($nl_id, $email, $values)
{
    $values["nl_id"] = $nl_id;
    $values["nl_email"] = $email;
    return $this->build_cockpit_url('newsletter/send_one', $values);
}

/*
Send a newsletter through the on-the-fly sending API
Use CURL
*/
public function send_newsletter_curl($nl_id, $email, $values)
{
    $url = $this->build_send_newsletter_url($nl_id, $email, $values);
    return $this->curl_call($url);
}
}

$KEY = "quo8Ahbooh9ahQuuVoo6si8uchewah6t";
$ID = 1;
$URL = "http://localhost:8000/rest/";
$hdlr = new Handler($URL, $KEY, $ID);

echo $hdlr->add_contact_curl(array("firstname" => "Arthur",
                                "lastname" => "Dent",
                                "primaryemail" => "arthur.dent@h2g2.org",

```

```
        "tags" => "Terrien,Anglais"));  
echo $hdlr->send_newsletter_curl(7, "arthur.dent@h2g2.org",  
    array("firtsname" => "arthur"));  
  
?>
```