Extended update procedure

Ideas and thoughts around it

Based on my post in CFEngine blog called

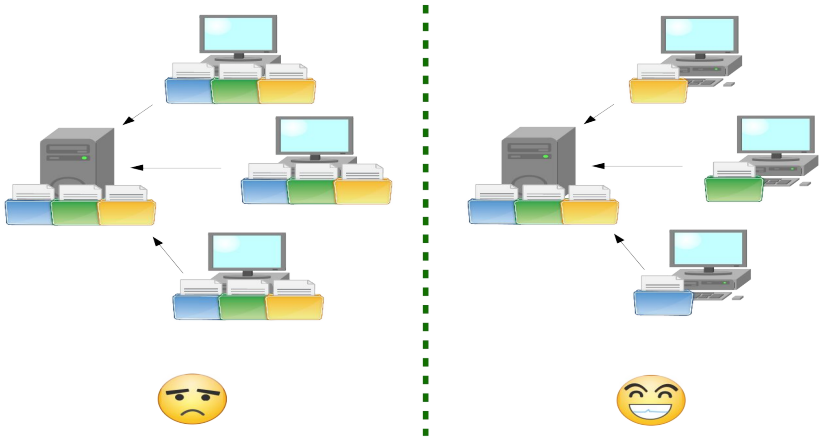
Extending the CFEngine Policy Update Procedure

About me

- jurica.borozan@gmail.com
- https://juricaborozan.blogspot.com
- https://github.com/jborozan/cfengine-repository/
- https://cfengine.com/company/blog-detail/extending-the-cfengine-policy-update-procedure
- Google+ Jurica Borozan, twitter @juricaborozan

Masterfile library update procedure

- It is reliable and optimized process for distribution of policy files to the clients – copies all files from hub masterfiles to clients inputs directory
- In heterogenous computing environment, coping the same content to all clients might be undesired (or just difficult to develop, test and scale)



Questions prior developing this procedure

- How to improve and optimize distribution of policy files
- How to stay compliant/do minimal change to standard master library
- How to ease complexity of policy files content
- How to improve development and test processes
- How to support versatility of computer systems
- How to prepare deployment or movement to cloud infrastructure

This procedure

- Is using „autorun“ feature on CFEngine clients
- Is implemented with 2 extra policy files, one for clients update process and one optional for hub!
- Stays “compliant” with update procedure of mastefiles policy framework
- Requires minimal changes to the files of masterfiles policy framework – (in future possibly none)
- Uses separate or secondary repository on the hub, which is different from masterfiles (and not inside)
- Relies on improved JSON support in CFEngine and it recommends concept of separation executable code from data

Tagging

- This procedure relies on usage of tags
- Tags are flexible way to mark computing resources
- They are not limited by number or format, there can be as many as we wish (but do not exaggerate)
- They enable distinguishing and grouping of computing resources (web, db, app1, app2, etc.) - so it simplifies orchestration
- IP address and hostname are tags too

Using tags on hub

- Inside secondary repository there shall be set of subdirectories named after tags
- Obviously tag names shall be unique just like subdirectory names
- Each subdirectory shall contain set of policy files that will be fetched by clients
- When number of policy files grows it is recommended to have some automated way to maintain these subdirectories and files in them

Using tags on clients

- Each CFEngine client fetches policy files from masterfiles directory and from tag named subdirectory(ies) in secondary repository
- All the policy files from those subdirectory(ies) go to the autorun subdirectory inside inputs directory
- Limitation for the policy files names is that they shall be unique to avoid overwriting
- CFEngine client re-reads tags upon every run – so it could change (dynamically) appliance of computing resources on the run

Assigning tag to clients

- Obviously tags could be placed inside some file – which is located outside masterfiles and inputs directories
- Or they could be fetched from somewhere via curl, wget or CFEngine get_url (e.g. get meta data in AWS or Openstack)
- Which depends heavily on infrastructure and type of services it provides

Policies and policy data

- Separation of policies and data is recommended
- This improves development and testing of policy files
- It could simplify handling platform differences
- I recommend to have each agent bundle in separate file and to have correspondingly named data file (JSON/YAML,) which is read at beginning
- Whole feature resembles CFEngine design studio but it tries to be simpler
- This is optional – extension works with any type of policy files containing bundles marked with “autorun” metatag

Policies and policy data (2)

```
1  bundle agent <...>
2  {
3    meta:
4      "tags" slist => { "autorun" };
5
6    vars:
7
8      linux::
9
10 #  set input files for all bundles
11       "params_files" slist => findfiles("$(this.promise_dirname)/$(this.bundle).json",
12                                         "$(this.promise_dirname)/$(this.bundle).yaml");
13       "params"         data => readdata(nth("params_files", 0), "auto");
14
15 # rm command
16       "cmd_dirs" slist => { "/usr/bin", "/bin" };
17       "cmds" slist => { "rm" };
18
19       "cmd[$(cmds)]" string => "$(cmd_dirs)/$(cmds)",
20         ifvarclass => fileexists("$(cmd_dirs)/$(cmds)");
```

Policies and policy data (3)

- Optionally adding some naming conventions might improve things even more
- Nouns like mysql_*.cf and firewall_*.cf describe what is all about
- Verbs like *_install.cf and *_init.cf do ease understanding what is supposed to happen
- It is better to avoid putting to much functionality inside policy file – since it could become difficult to test and debug

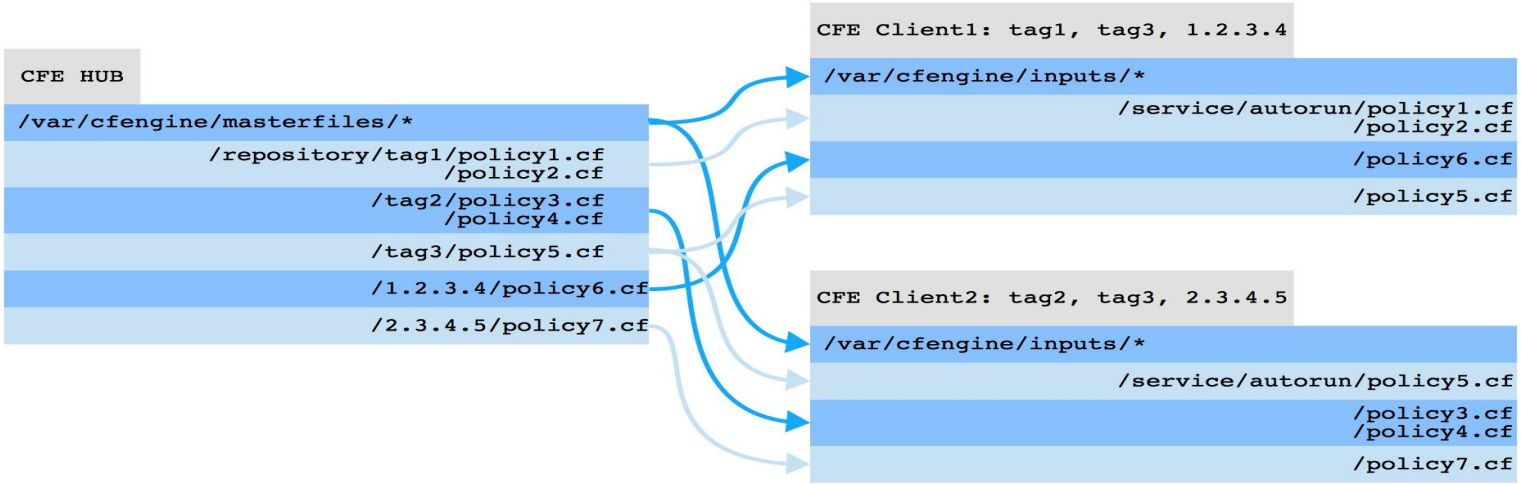
Client side implementation

- There are 2 files:
  - $(sys.inputdir)/jb_update_policy3.cf
  - $(sys.workdir)/node.tags
- Update policy file reads tags from second file each time cf-agent runs it
- Benefits on such dynamic behaviour is flexibility: adding and removing tags controls fetching policy files on updates
- Hostname and IP address are included into tags automatically

Client side implementation (2)

- In the first phase, update process goes as usual – standard CFEngine procedure fetching content of mastefiles directory and coping it to inputs
- Second phase is defined by extended update policy script and it tries to fetch content of subdirectories (named after tags) and copy them to autorun subdirectory
- Extended update policy script uses timestamps to check changes and prevent to much network traffic on hub

Client side implementation (3)



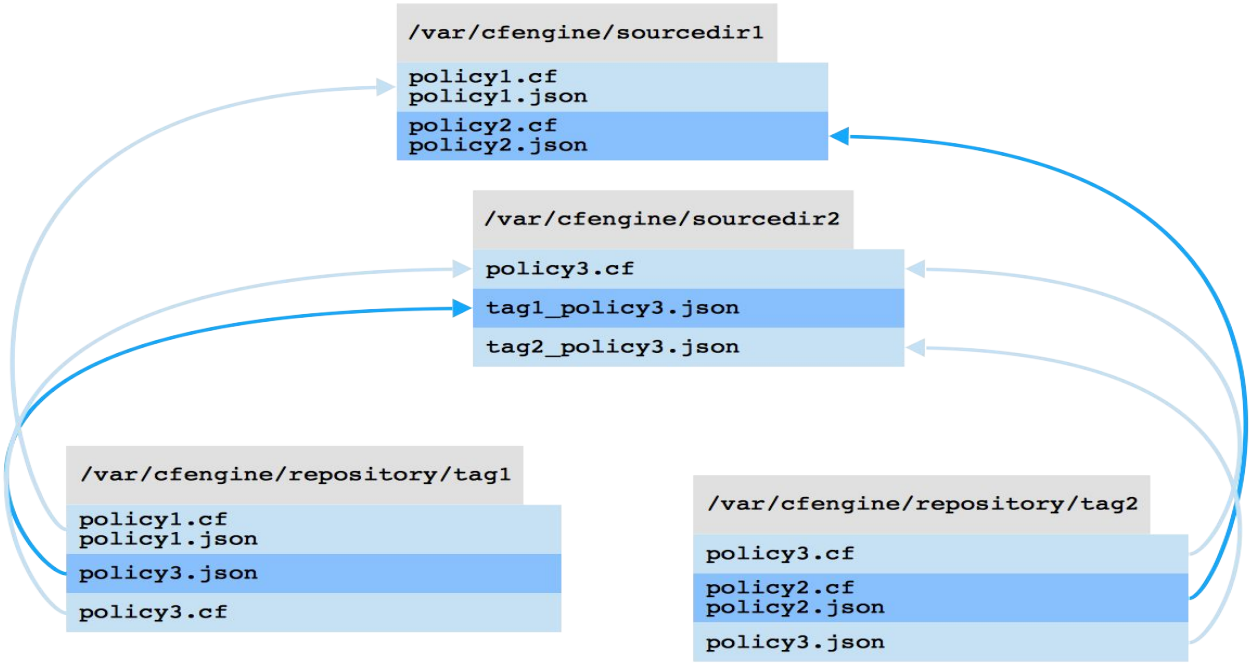
Hub side implementation

- Option of manually maintaining secondary repository directory structure is not recommended
- Repeated coping of policy files and their data to subdirectories is not effective and it is error prone procedure
- CFEngine does not follow symbolic links but using hard links might be useful
- I developed another policy file to maintain secondary repository subdirectories and their content using hard links

Hub side implementation (2)

- There are some rules how my hub script link the files:
  - There can be more source locations with policy files
  - If the filename is prefixed with tag name, that one shall be linked to certain subdirectory but without prefix – this enables having tag specific policy and data files
- All other non desired links and files must be removed from subdirectories in repository
- Timestamp shall be maintained to indicate changes of the files and lessen network traffic

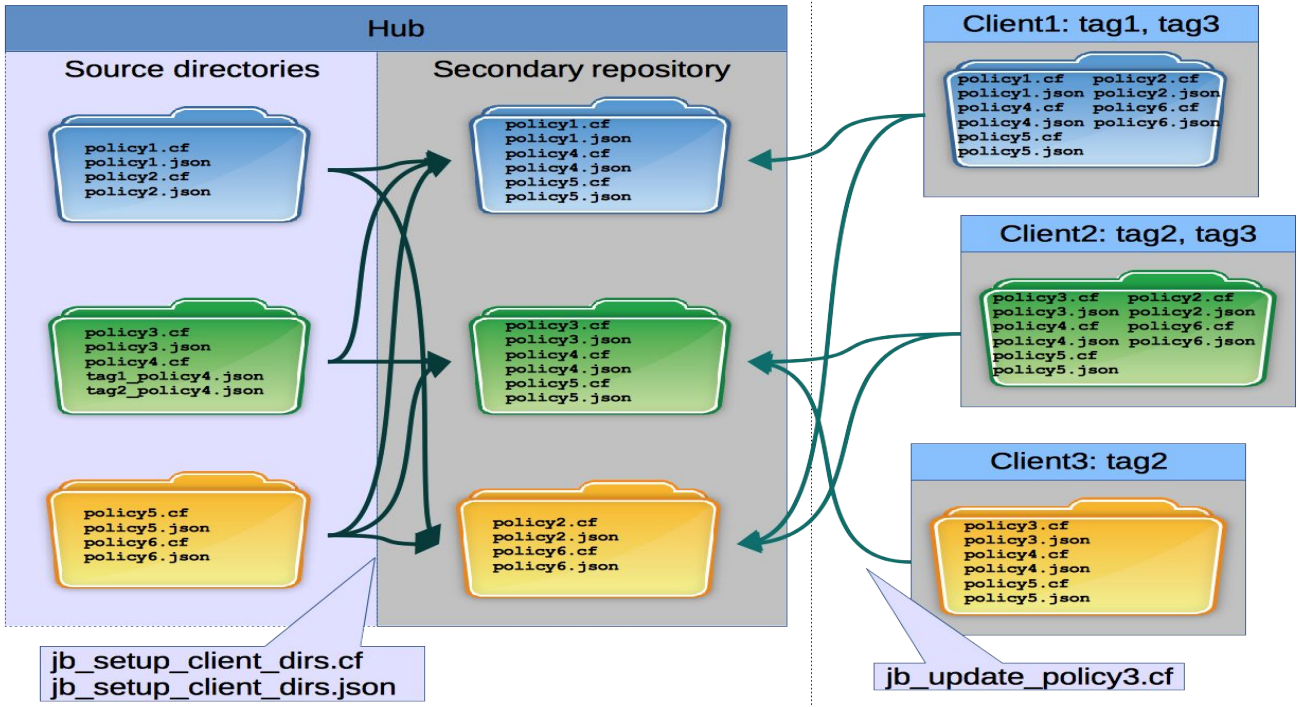
Hub side implementation (3)



Hub side implementation (4)

```
{
  "repository_dir" : "/var/cfengine/repository/cfengine",

  "policies_dirs" : [
                     "/var/cfengine/repository/policies/cfengine",
                     "/var/cfengine/repository/policies/security",
                     "/var/cfengine/repository/policies/utilities",
                     "/var/cfengine/repository/policies/databases",
                     "/var/cfengine/repository/policies/web",
                     "/var/cfengine/repository/policies/cloud",
                     "/var/cfengine/repository/policies/system"
                   ],

  "default" : [
                "jb_user.cf",
                "jb_user.json",
                "jb_iptables_install.cf",
                "jb_iptables_install.json",
...
                "jb_vim.cf",
                "jb_vim.json"
              ],

  "tags" : {
              "172.16.98.149" : [
                                  "jb_firewalld_services_init.cf",
                                  ...
                                  "jb_ssh_user_pubkey_login_setup.json"
                                ],

              "172.16.98.214" : [
                                  "jb_firewalld_services_init.cf",
                                  "jb_firewalld_services_init.json",
                                  ...
                                  "jb_glusterfs_server_init.json"
                                ],

              "172.16.98.130" : [
                                  "jb_yum_repos_mysql_community.cf",
                                  ...
                                  "jb_glusterfs_server_init.json"
                                ]
            }
}
```

Managing secondary repository and distribution of policy files - schema



Cloud appliance

- Question with cloud appliance is whether to use application prepared image (e.g. for mysql, nginx) or CFEngine prepared image which later on takes care of rest
- CFEngine is “lite” – which makes it quite applicable inside prepared images
- Obviously growing number of computer resources in cloud will require some kind of configuration management or orchestration tool
- CFEngine prepared images or generic ones could be setup for any purpose – which is practical during development where changes are the rule, not exception

Cloud appliance (2)

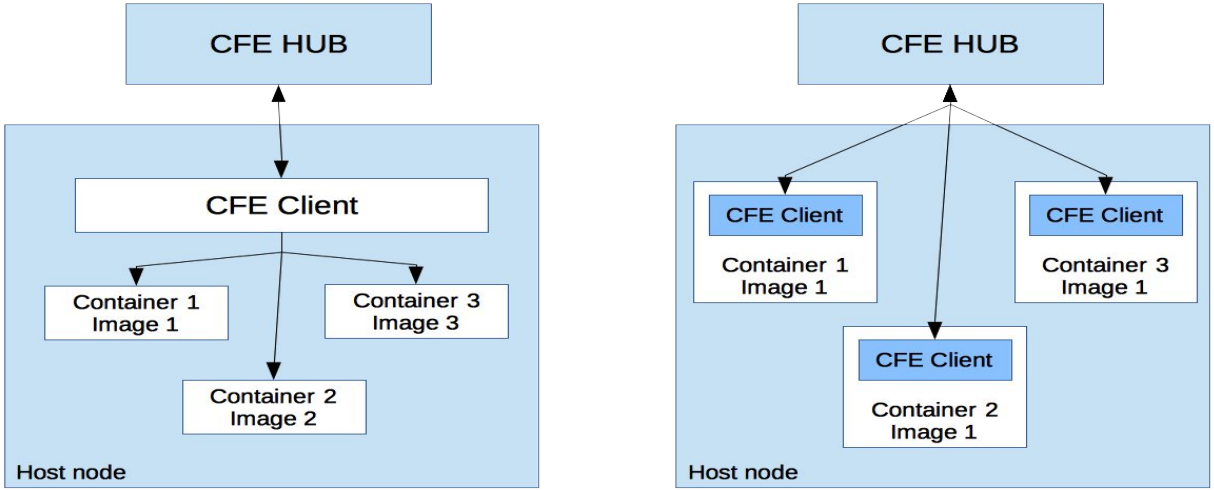
- Most efficient would be to start with some image that has CFEngine installed on it
- But using user data file, CFEngine can be easily installed at boot time (i.e. AWS, Openstack)
- And user data file could be also used to write tags in the tags file too
- Alternative way is to assign tags to an instance would be to fetched them using curl, wget or CFEngine get_url() function during runtime

Cloud appliance (3)

- Using Web based interface is good but tasks around computer resources in cloud shall be automated
- This implies using API or CLI for management (and GUI for control)
- E.g. Think of replicating some computer system of 100 nodes regionally (or for development, test and production)
- CFEngine can help orchestrating resources
- E.g. I use CFEngine to create, run, control and tag computing resources on AWS via CLI

Container appliance

- Similar to cloud appliance question is where to put CFEngine client: inside or outside container



Container appliance (2)

- Depends a lot on someone’s skills to prepare container images (e.g. dockerfile) and skills to use CFEngine
- Usual single appliance of container makes CFEngine more applicable as container controller on the host
- Putting CFEngine inside container makes sense in cases when you need to work with generic image and when appliance configuration requires significant effort
- (Using CFEngine on the host and in the container makes sense usually for the labs and experimenting)

Container appliance (3)

- Some thoughts on how to set tags to container instances in the case when CFEngine client is in the container:
  - Simple way is to prepare tags as local files and mount them in containers (e.g. Docker)
  - Option of fetching them in runtime via curl/wget/get_url() is do-able too (e.g. Docker labels via unix sockets but more complex than fetching AWS metadata)

Future improvements

- Checking existing against fetched files to improve security and avoid possibility to execute some rogue policy file found in autorun directory
- Developing (or extending) way to manage hub secondary repository via CFEngine portal (plugin) – only one JSON file on the hub shall be processed
- (Include update procedure via def.json)

Summary

- This extension enables relatively simple and clean deployment of policy files onto CFEngine client
- It follows convention and practices from CFEngine masterfiles policy framework and requires minimal change on it
- It makes policy files simpler and easier to develop, deploy and test
- It relaxes „selection“ logic in policy files by moving it out to selective distribution of those and it eases orchestration
- It is approaching idea of droplets (or linklets) where policy files and data could be dropped into subdirectory and automatically deployed

Thank you for your attention