Puppet getting started

Best practices on how to turn Your environment into a Puppet managed environment

Puppet Camp 2013 – Nuremberg – 2013-04-18

Bernd Strößenreuther

mailto:puppet-camp@stroessenreuther.info

License

- You may use, change or redistribute this document under the creative commons license http://creativecommons.org/licenses/by-sa/3.0/

Agenda

1. Best Practices
   Some things to consider when introducing puppet in Your environment
2. See it in a demo setup
   Let‘s try to set up a little puppet environment
   and try out what You are interested in

This is a workshop

Interact!

Stop thinking procedural!

- Start thinking declarativ!
- Avoid `exec` where ever possible!!

Example Manifest: SSH

```
class ssh{
    package { 'openssh-server':
        ensure  => installed;
    }
    file { '/etc/ssh/sshd_config':
        owner   => 'root',
        group   => 'root',
        mode    => '0644',
        source  => 'puppet:///ssh/sshd_config',
        require => Package['openssh-server'],
        notify  => Service['ssh'];
    }
    service { 'ssh':
        ensure  => running,
        enable  => true,
        require => File['/etc/ssh/sshd_config'];
    }
}
```

Module inheritance: site.pp

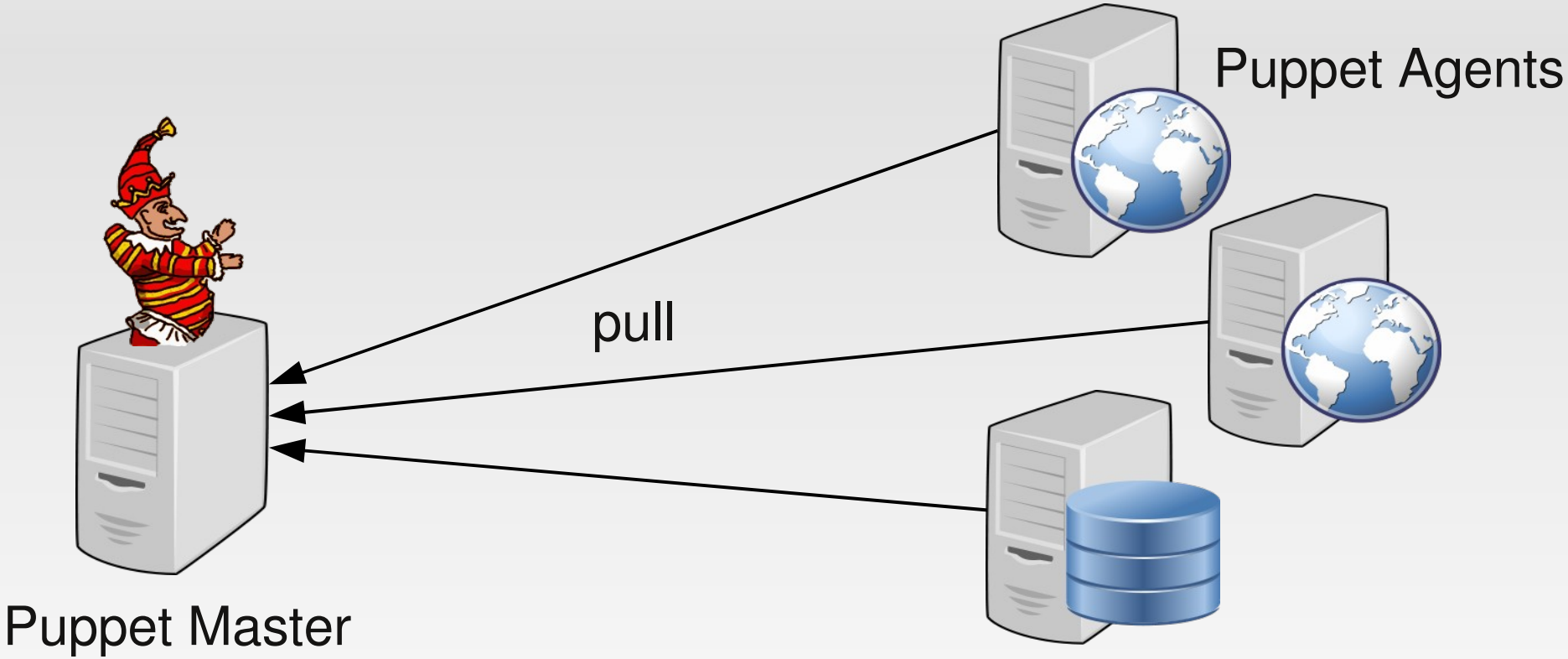
```
node default {
    fail "${fqdn} has no puppet modules assigned to, no
node definition matching"
}

node basenode {
    include 'ssh'
    include 'adminusers'
}

node /webserver[0-9].example.com/ inherits basenode {
    include 'httpd'
}

# including definitions from file another_config.pp
import 'another_config'
```

Puppet Infrastructure



Images from OpenClipart.org, released to the public domain. Thanks to the contributors!

Puppet Camp
[Nuremberg] [Apr 19th 2013]

Which version of Puppet to use?

- At least 2.7.x
- If Your distribution provides only elder versions, You can use the PuppetLabs Repos at http://apt.puppetlabs.com/ or http://yum.puppetlabs.com/
- Use version pinning, if required, see http://docs.puppetlabs.com/guides/upgrading.html

Configuration Management ≠ Software Distribution

- Do not transport software products over Puppet mechanisms onto the agents
- Instead:
  - Put software into rpm or deb packages
  - Put packages into a repository
  - Use Puppets package resource to install
  - If You do not yet have a local repository, You might want to have a look at mrepo
    http://dag.wieers.com/home-made/mrepo/
    (supports yum and apt)

How to start my Puppet rollout?

- With nothing!
- You can bring the Puppet Agent onto a node,
  connect it to Puppet Master,
  have it running and have it configure nothing.
  (Not even a single file or service!)
- You can put more and more resources (files, services, users, …) under control of Puppet afterwards and step by step

Which configuration files and services should I put under control of Puppet first?

- Configure one non-critcal service on view machines first.
- Do the „quick win‘s“ next
- Eye-catching headers in every Puppet managed config file are helpful

Should I use a Version Control System?

- If You already have one for Your config files, You do not want to miss!
- If You do not have one, introducing it together with Puppet is the ideal time.
- Keep site.pp and all Your Puppet modules there
- Use meaningful commit messages:
  - Use not too many words on **what** You did change
  - Tell **why** You did change it
  - One line of text is often enough

Connecting the Version Control System to the Puppet Master

- Changes in version control system should be automatically available on the Puppet master
- Use hook scripts
  - post-commit hook e. g. in Subversion
  - post-update hook e. g. in Git

Staging of Puppet Modules

- Only tested and approved versions of modules should be applied to productive machines
- Productive version and development version of one module should live in the version control system
- Distinguish by different branches or by tags
- Puppet provides “environments” for different types of agents
- Hook script needs to checkout the right branch or tag into the according Puppet environment

Puppet Environments: Config on the master

- extract of /etc/puppet/puppet.conf:

```
[main]
    # ....
[test]
    manifest   = /etc/puppet/test/manifests/site.pp
    modulepath = /etc/puppet/test/modules
[production]
    manifest   = /etc/puppet/production/manifests/site.pp
    modulepath = /etc/puppet/production/modules
```

Puppet Environments: Config on the agent

- extract of /etc/puppet/puppet.conf:

```
[main]
    # ....
    pluginsync=true
    report=true
[agent]
    environment = test
```

Example workfow with branches (1/2)

- You have 2 long living branches
  - `master` for Your test machines
  - `production` for Your productive machines

Example workfow with branches (2/2)

1. You want to change a Puppet module
2. Create a new development branch `feature01` based on `master`
3. Do Your changes in `feature01`, merge them back to `master`
4. Rollout by Puppet onto Your test machines:
   Approve Your changes there
5. If enhancements or bugfixes required: goto 3.
6. If ok: merge branch `feature01` onto `production`
7. `puppetd --test --noop`
8. Rollout by Puppet onto Your productive machines
9. Delete `feature01` branch

Example workfow with tags (1/2)

- Often used with Subversion
- Basic setup:
  - Most of the time You have no branch beside trunk
  - Set up an own project for each Puppet module
    plus one for main manifests (plus one for hieradata)
  - Write a post-commit hook script that checks out trunk into Puppet‘s environment test and the latest tag of each project into production environment
  - Use defined names for Your tags,
    e. g. YYYY-MM-DD_hh-mm

Example workfow with tags (2/2)

1. You want to change a Puppet module
2. Commit Your changes (into `trunk`)
3. Rollout by Puppet onto Your test machines:
   Approve Your changes there
4. If enhancements or bugfixes required: goto 2.
5. If ok: check for other changes on this module not yet tagged
   (`svn diff`)
6. tag last version of the changed Puppet module (Subversion project)
7. `puppetd --test --noop`
8. Rollout by Puppet onto Your productive machines

pre-commit Hook / pre-receive Hook

- Do syntax checks as early as possible: On commit

```
puppet parser validate <filename.pp>
puppet-lint <filename.pp>
cat <filename.erb> | erb -P -x -T - | ruby –c
```

- Save time!
- Never get checked in files that do not even compile or violate agreed coding style
- Avoid frustration of other admins
- Samples:

http://projects.puppetlabs.com/projects/1/wiki/Subversion_Commit_Hooks_Patterns

https://puppetlabs.com/blog/using-puppet-lint-to-save-yourself-from-style-faux-pas/

Puppet‘s Module Path

- By default each Puppet environment has exactly one module path
- For most setups too flat and confusing
- Use at least two:
  - One for third party modules (e. g. PuppetForge)
  - One for Your own modules
- maybe You want to split Your own further e. g. in
  - „generic“ modules
  - „node group specific“ modules

Multiple Module Path Entries

- extract of /etc/puppet/puppet.conf on Puppet master:

```
[main]
    # ....

[test]
    manifest   = /etc/puppet/test/manifests/site.pp
    modulepath = /etc/puppet/test/modules/site:/etc/puppet
/test/modules/thirdparty

[production]
    manifest   = /etc/puppet/production/manifests/site.pp
    modulepath = /etc/puppet/production/modules/site:/etc/
puppet/production/modules/thirdparty
```

Where to assign Puppet modules to nodes (1/3)

- Manually in site.pp

```
node basenode {
    include 'ssh'
    include 'adminusers'
}

node webservers inherits basenode {
    include 'httpd'
}

node 'webserver1.example.com' inherits webservers { }
node 'webserver2.example.com' inherits webservers { }
```

Where to assign Puppet modules to nodes (2/3)

- By convention
  - Strict naming convention for hostnames required
  - Regular expressions are allowed in site.pp

```
node basenode {
    include 'ssh'
    include 'adminusers'
}

node /webserver[0-9].example.com/ inherits basenode {
    include 'httpd'
}
```

Where to assign Puppet modules to nodes (3/3)

- In Your CMDB by using External Node Classifiers (ENC)

  http://docs.puppetlabs.com/guides/external_nodes.html

Puppet Agents in the DMZ (1/6)

How do I get the servers in my DMZ connected to Puppet if the security policy of my company does not allow connections from outside (DMZ) to inside (to my Puppet master)?

- You can use a Remote SSH Tunnel for this
- Create an user for this task on Your Puppet master and all of Your DMZ agents
- Enable key authentication for SSH from master to <dmz-agent>

```
[puppetuser@master ~]$ ssh-keygen
[puppetuser@master ~]$ ssh-copy-id <dmz-agent>
```

Puppet Agents in the DMZ (2/6)

- Configure reverse SSH Tunnels for all connections to DMZ agents

```
[puppetuser@master ~]$ cat ~.ssh/config
Host *
    RemoteForward 8140 127.0.0.1:8140
    StrictHostKeyChecking no
    BatchMode yes
```

- Tell Puppet on DMZ agents to use Puppet master at localhost

```
[puppetuser@dmz-agent ~]$ cat /etc/puppet/puppet.conf
# ...
[agent]
    server = localhost
# ...
```

Puppet Agents in the DMZ (3/6)

- Allow puppetuser on DMZ agents to run Puppet as root by sudo

```
[puppetuser@master ~]$ cat /etc/sudoers
# ...
Defaults:puppetuser !requiretty
puppetuser ALL=(root) NOPASSWD: /usr/sbin/puppetd
```

- Add a forced command to the SSH key that You just created (You may also restrict IPs to Your Puppet masters)

```
[puppetuser@dmz-agent ~]$  cat ~/.ssh/authorized_keys
from="10.0.0.10,10.0.0.11",
command="/usr/bin/sudo –H /usr/sbin/puppetd --test",
no-X11-forwarding,no-agent-forwarding ssh-rsa
AAAAB3NzaC12[...]tooxPKT/BSGNw== puppet push account
```

Puppet Agents in the DMZ (4/6)

- Use a variable “puppetmaster” in all Your file resources filled in site.pp:

```
node basenode {
    $puppetmaster = $network_zone_int_ext ? {
        'ext'   => 'localhost',
        default => $servername
    }
}
```

- Used in every file resource in all Your modules

```
file { '/etc/foo':
  source => "puppet://$::puppetmaster/modules/mymod/foo",
  owner  => 'root‘;
}
```

Puppet Agents in the DMZ (5/6)

- Where `network_zone_int_ext` can be a custom fact defined in `mymod/lib/facter/network_zone_int_ext.rb`

```
require 'facter'
Facter.add("network_zone_int_ext") do
  setcode do
    network_zone_int_ext = "int"
    if Facter.value(:ipaddress).match(/^(10\.1\.|10\.2\.)/)
      network_zone_int_ext = "ext"
    else
      network_zone_int_ext = "int"
    end
  end
end
```

Puppet Agents in the DMZ (6/6)

- Set up a cronjob for puppetuser on Puppet master, that regularly calls a ssh to every DMZ agent
- The list of all DMZ agents can automatically be filled by a exported resource

PuppetForge

- A public repository for Puppet Modules
- https://forge.puppetlabs.com/
- Quality of modules differs very much

Hands on...

Live Demo

Any more questions?

- Now is a good time to ask
- Grab me on the camp
- I‘ll be around here tomorrow!
- Hear all the other interesting talks on the camp

Puppet Camp
[Nuremberg] [Apr 19th 2013]

Appendix

hiera

- A hierachical store for `name=value` pairs
- The hierachy can be configured according to Your needs
- The most specific entry is taken
- Can easily be queried by puppet
- Put variables here
- Ideal if You have many common servers and view exceptions

Facts

- Puppet queries many details of the system it configures, facter puts these into single variables
- They can be used in templates and manifests

```
[booboo@dunno ~]$ facter
architecture => i386
domain => example.com
fqdn => dunno.example.com
hardwareisa => i686
hostname => dunno
interfaces => eth0,lo,peth0,sit0,veth1,vif0_0,vif0_1
ipaddress => 10.0.0.182
ipaddress_eth0 => 10.0.0.182
ipaddress_lo => 127.0.0.1
is_virtual => false
...
```

Facter example

- Your hosts have a productive network interface and one for management
- You want Your apache to listen only on the productive interface
- Unable to use:

```
Listen 80
```

- Use instead:

```
Listen <%= ipaddress_eth0 %>:80
```

Custom Facts

- E. g. stage or datacenter
- Write a little bit of ruby code
- Put it into `<mymodule>/lib/facter/<factname>.rb`
- Set in `/etc/puppet/puppet.conf` at the agent:

```
[main]
    # ....
    pluginsync=true
```

Exported Resources

- Whenever You add a new host under control of Puppet You might want to add basic monitorings (disk space, CPU usage, …) to Your monitoring system (running on another node)
- Whenever You add Your Puppet module „apache“ to a host You need to configure a regular check of HTTP on this host in Your monitoring system
- Let Puppet do this for You automatically!
- Sounds useful? Use Exportet Resources to configure this.

Exported Resources Example (monitored machine)

```
class apache {
   service {'httpd':
        ensure    => running,
        enable    => true,
        hasstatus => true;
   }
   @@nagios_service { "check_http_${hostname}":
        check_command       => 'check_http_port_path!80!/',
        use                 => 'generic-service',
        host_name           => $hostname,
        notification_period => '24x7',
        service_description => 'HTTP GET /',
        target              => '/etc/icinga/objects.puppet.
                               autogen/services.http.cfg';
   }
}
```

Exported Resources Example (monitoring machine)

```
class icinga-server {
    file { '/etc/icinga/objects.puppet.autogen/services.
           http.cfg':
        owner   => 'root',
        group   => 'root',
        mode    => '0644';
    }

    # collect resources
    # and populate
    # /etc/icinga/objects.puppet.autogen/*.cfg
    Nagios_service <<||>>
}
```

Exported Resources Example: Result

```
[booboo@icinga-server ~]$ cat services.http.cfg
# HEADER: This file was autogenerated at
# HEADER: Fri Dec 14 13:53:26 +0100 2012
# HEADER: by puppet.  While it can still be managed
# HEADER: manually, it is definitely not recommended.

define service {
        ## --PUPPET_NAME-- (called '_naginator_name' in
        ## the manifest) check_http_dunno1
        use                             generic-service
        service_description             HTTP GET /
        check_command                   check_http_port_path!80!/
        host_name                       dunno1
        notification_period             24x7
}
```