Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement gut und wofür besser nicht?
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- Übersicht zu den Möglichkeiten der Organisation von Puppet-Konfiguration
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

whoami

Vita: Sascha Brechmann

- Linux-User seit Debian „Bo/Hamm“
- Nagios-User seit 2005
- Puppet-User seit 2010
- Check_MK User seit 2010/2011

Konfigurationsmanagement mit Puppet

- **Grundlagen des Konfigurationsmanagement**
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement und wofür besser nicht?
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- Übersicht zu den Möglichkeiten der Organisation von Puppet-Konfiguration
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

Grundlagen des Konfigurationsmanagement

- Was ist Konfigurationsmanagement
- Warum macht man ein Konf.Mgmt
- Welche Möglichkeiten für ein Konfigurationsmanagement gibt es
- Warum sollte ich ein Konfigurationsmanagement Framework wie Puppet nutzen und keine „Shell-Skripte“

Was ist Konfigurationsmanagement

Konfigurationsmanagement (KM) ist eine Managementdisziplin, die **organisatorische** und **verhaltensmäßige Regeln** auf den **Produktlebenslauf** einer **Konfigurationseinheit** von seiner Entwicklung über Herstellung und Betreuung anwendet.

Konfigurationseinheit meint in diesem Zusammenhang eine „beliebige **Kombination** aus **Hardware**, **Software** oder **Diensten**“.

© Wikipedia

Was ist Konfigurationsmanagement

Konfigurationsmanagement sind also Regel und Prozesse, welche auf einer beliebigen Anzahl an (Server-)Systemen koordiniert angewendet und überwacht zu werden.

Hierbei soll folgendes erreicht werden:

- **Einheitlichkeit**
- **Überprüfbarkeit**
- **Nachvollziehbarkeit**

Ziele des Konfigurationsmanagement

- **Einheitlichkeit**
  Eine Konfigurations-Regel soll bei allen betroffenen Systemen immer zum **gleichen Ergebnis** führen. D.H. Server(/Rollen) sind nicht mehr unterschiedlich!
- **Überprüfbarkeit**
  Es ist möglich **vorab** zu sehen, **ob** und **was** geändert werden würde. Es ist auch möglich zu sehen **was , wann und wie** Verändert **wurde**. (Logging + Auditing + Compliance)
- **Nachvollziehbarkeit**
  Jede Änderung wird **Protokolliert** und kann g.g. wieder zurück genommen werden. (Change-Log + Rollback)

Warum macht man ein Konfig.Mgmt

- **rechtliche oder vertragliche Vorgaben**
- ITIL / ITSM
- Vermeidung von **Abhängigkeiten** (Nur Mitarbeiter „X“ weiß, wie dieses System installiert wird)
- **Wiederherstellbarkeit** von **Systemen**, trotz des Delta zwischen Backup und Ausfall (**nicht die Wiederherstellbarkeit von Daten!**)
- **Standardisierung** (Basis-Installation; Default-Tools; Default-User; Default-Verzeichnispfade; Einheitliches Bild von Konfig-Dateien)
- **[Fähigkeit zum Skalieren (Mitarbeiter + Systeme)]**

Warum ein Konf.Mgmt Framework und keine „Shell“-Skripte

- Shell-Skripte sind meistens nicht besonders Abstrakt und sehr oft Komplex.
- Shell-Skripte sind meistens auf eine Umgebung optimiert, schon ein Versions-Update kann hier zu Problemen führen.
- Shell-Skripte funktionieren eigentlich nie OS- oder Distributions- Übergreifend

Konfigurationsmanagement Framework

Eine kleine Liste, der bekanntesten Konf.Mgmt Frameworks:

- **Cfengine** (Urvater aller KMF)
- **Puppet** („einfache“ Syntax zur Konfigurations-Beschreibung)
- **Chef** (Entwickler lieben es, weil auch die Konfiguration in Ruby „Programmiert“ wird)
- **Ansible** (New Kid on the Block)

Vorteile von Konfigurationsmanagement Software

- Zentralisiertes und automatisches Management von Software und Konfiguration
- Schnelle und Skalierbare Deployments
- Konfigurationen sind leicht anpassbar
- Einhaltung von Abhängigkeiten
- Möglichkeiten zur (automatischen) Dokumentation

Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- **Wofür ist ein Konfigurationsmanagement und wofür besser nicht?**
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- Übersicht zu den Möglichkeiten der Organisation von Puppet-Konfiguration
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

Was sich gut mit einem Konf.MGMT verwalten lässt

- Konfigdateien / Text-Datei „z.B. alles unter /etc/“
- Services/Dienste „Einrichten/Starten/Stoppen/Löschen“
- Software-Packete „Installieren/Deinstallieren/Versionen“
- User/Gruppen/Rechte-Verwaltung
- Mountpoints
- Cronjobs

Was sollte man nicht mit einem Konf.Mgmt Framework machen

- Datei-Server / Dateien Kopieren.
  Hier besser NFS/Rsync nutzen
- (Shell-)Befehle ausführen. Hier besser eine Orchestration-Software wie MCollective nutzen.
- „Software-Deployment“. Applikations-Software, wie z.B. PHP-Applikationen.
  Hier besser Cappistrano oder Func
- Scheduler.
  Hier besser etwas wie Cron, GNUBatch oder ähnliches nutzen

Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement und wofür besser nicht?
- **Best Practices beim Umgang mit Puppet**
- Übersicht der Konfigurations-Möglichkeiten
- Übersicht zu den Möglichkeiten der Organisation von Puppet-Konfiguration
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

Best Practices beim Umgang mit Puppet

- **Trennung** von „Puppet“**-Code** und **Daten** (Params.pp / Hiera / ENC)
- Puppet-Code (**Module**) in **Versions-Kontrolle** (--> GIT)
- **Konfiguration-Daten** in **Versions-Kontrolle** (Hieradata --> GIT)
- **Sensible Daten** in „sicheren Konfiguration-Backend“ (Hiera-GPG/Hiera-Eyaml) / Verschlüsselung nutzen!
- Stages/**Environments** für Puppet-Code (DEV/Stage/Prod, auch für Puppet)
- Module von Puppetforge nutzen
- **Style-Code** (puppet-lint) / Code-Testing (rspecs)/ Unit-Tests einbauen
- „**Infrastructure as code**“ / Software-Entwicklungs-Prinzipien auf die Server- / VM-Landschaft anwenden

Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement und wofür besser nicht?
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- Möglichkeiten der Organisation von Puppet-Konfiguration
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

Möglichkeiten für ein „Konfigurationsmanagement“

- „Changelog-File“
- „for i in ...“ (Shell-)Skripte
- Rsync
- Golden-Image
- Konfigurationsmanagement Framework

Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement und wofür besser nicht?
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- **Möglichkeiten der Organisation von Puppet-Konfiguration**
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

Organisation von Puppet-Konfiguration

- Klassen sind kleine Konfigurations-Einheiten, z.B. Apache-Package Installation
- Module bestehen aus mehreren Klassen Technical-View, z.B. Apache-Module
- Profile enthalten „konfigurierte“ Module Organisation-View, z.B. Apache-Vhost
- Rollen sind eine Sammlung von Profilen Business-View, z.B. Webshop auf Apache

Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement und wofür besser nicht?
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- Möglichkeiten der Organisation von Puppet-Konfiguration
- **Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK**
- Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

Beispiele zur Nutzung von Puppet in Verbindung mit Check_MK

- Setzen der Check_MK Tags via Puppet
  - Via **Puppet Facts**
    - FQDN
    - Operationsystem
    - usw. ->Facter
  - Via Custom Facts
  - Via if „has_variable“
- Setzen von Host-Alias und Notes_/Action_URL
- Setzen der Host-Parents

Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK

- Puppet-Klasse „check_mk::agent“ im Basis-Client. **Exportiert** :
  „all_hosts +=[<FQDN>|<TAGs>]“
  all_hosts +=[host.example.de|tcp|beispiel]“
- Puppet-Klasse „check_mk::server“ **importiert** die Client-Konfig von oben. Damit sind automatisch alle Puppet-Managed-Hosts im Check_MK

Puppet Tags im host.erb Template

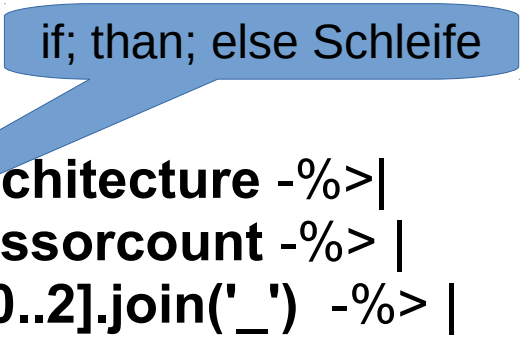
all_hosts += [

" <%= fqdn -%>**|CG_DEFAULT** | CG_customer |
<%= **architecture** -%>| <%= operatingsystem -%> |
processorcount_<%= **processorcount** -%> |
network_<%= scope.lookupvar('::ipaddress').split(".")[0..2].join('_') -%> |
]

Beispiel: all_hosts +=[

„host.example.de|**CG_DEFAULT** | CG_customer | x86_64|...

Puppet Tags im host.erb Template




```
all_hosts += [
    " <%= fqdn -%>|CG_DEFAULT | CG_customer | <%= architecture -%>|
<%= operatingsystem -%> |  processorcount_<%= processorcount -%> |
network_<%= scope.lookupvar('::ipaddress').split(".")[0..2].join('_')  -%> |
<% if hostname[/stage|dev/] == nil %>tcp<% else %>ping<% end %> |
",
]
```

Beispiel: **all_hosts += [**

„dev-host.example.de|**CG_DEFAULT** | CG_customer | ping|...“

„prod-host.example.de|**CG_DEFAULT** | CG_customer | tcp|...“

Puppet Tags im host.erb Template

Transformation einer Variable

```
all_hosts += [
    " <%= fqdn -%>|CG_DEFAULT | CG_customer |<%= architecture -%>|
<%= operatingsystem -%> |  processorcount_<%= processorcount -%> |

network_<%= scope.lookupvar('::ipaddress').split(".")[0..2].join('_')  -
%> |
",
]
```

Beispiel: all_hosts +=[

„host.example.de|CG_DEFAULT | CG_customer | network_192_168_123|...

Puppet Tags im host.erb Template

```
all_hosts += [
    " <%= fqdn -%>|CG_DEFAULT | CG_customer | <%= architecture -%>|
<%= operatingsystem -%> |  processorcount_<%= processorcount -%> |
network_<%= scope.lookupvar('::ipaddress').split(".")[0..2].join('_')  -%> |
<% if hostname[/stage|dev/] == nil %>tcp<% else %>ping<% end %> |
<% if @mysql_role %>mysql_role_<%=mysql_role-%><%else
%>no_mysql_role<%end%> |
<%= mktags.join('|') %> ",
]
```

Überprüfung ob eine Variable gesetzt ist

Beispiel: all_hosts +=[

„db-host.example.de|CG_DEFAULT | CG_customer |
mysql_role_master|...

Puppet Tags im host.erb Template

```
all_hosts += [
    " <%= fqdn -%>|CG_DEFAULT | CG_customer | <%= architecture -%>|
<%= operatingsystem -%> |  processorcount_<%= processorcount -%> |
network_<%= scope.lookupvar('::ipaddress').split(".")[0..2].join('_')  -%> |
<% if hostname[/stage|dev/] == nil %>tcp<% else %>ping<% end %> |
<% if @mysql_role %>mysql_role_<%=mysql_role-%><%else%>no_mysql_role<%end%> |
<%= mktags.join('|') %> "
]
extra_host_conf['alias'] += [  ( '<%= hostname -%>', ['<%= fqdn -%>'], ), ]
extra_host_conf['notes_url'] += [
("/<%= omd_site -%>/wiki/doku.php?id=nagios_doku:hosts:<%= hostname -%>", ['<%= fqdn -%>'], ),
]
```

Puppet-Array

→ $OMD_ROOT/etc/check_mk/conf.d/$FQDN.mk
+ CMK -I $FQDN + CMK -O

Puppet Tags im host.erb Template

Custom Puppet-Fact, welcher alle Puppet-Klassen eines Hosts enthält.
->/var/lib/puppet/classes.txt

```
all_hosts += [
    " <%= fqdn -%>|CG_DEFAULT | CG
<%= operatingsystem -%> |  processorcount
network_<%= scope.lookupvar('::ipaddre )[0..2].join('_')  -%> |
<% if hostname[/stage|dev/] == nil %  else %>ping<% end %> |
<% if @mysql_role %>mysql_role_<%=mysql_role-%><%else%>no_mysql_role<%end%> |
<% if @puppet_classes_csv %> <%= puppet_classes_csv.split(",").join('|').split(":").join('_') -%>
<%end%>|<%= mktags.join('|') %> ",
]
extra_host_conf['alias'] += [  ( '<%= hostname -%>', ['<%= fqdn -%>'], ), ]
extra_host_conf['notes_url'] += [
("/<%= omd_site -%>/wiki/doku.php?id=nagios_doku:hosts:<%= hostname -%>", ['<%= fqdn -%>'], ),
]
```

→ $OMD_ROOT/etc/check_mk/conf.d/$FQDN.mk
+ CMK -I $FQDN + CMK -O

Puppet Tags im host.erb Template

extra_host_conf[

('<%= hostname -%>' , ['<%= fqdn -%>'],),]

Setzen von host-spezifischem Alias

Beispiel:

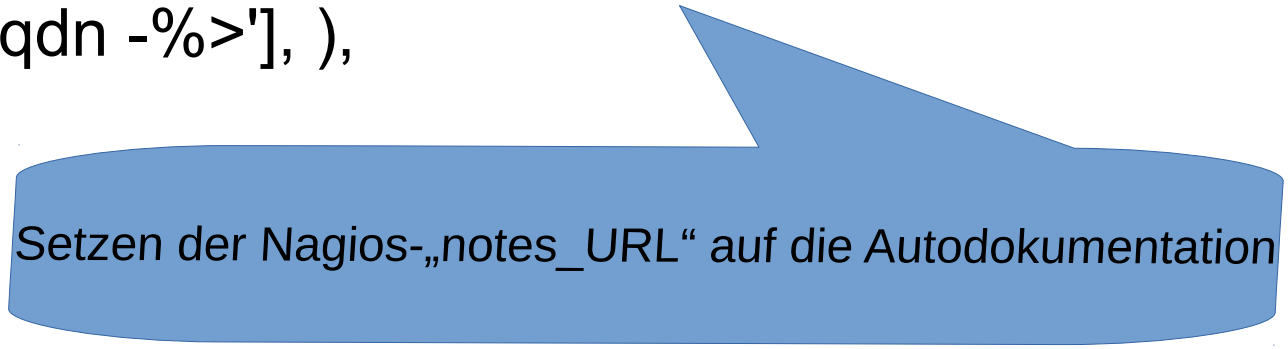
```
extra_host_conf['alias'] += [
  ('db-host', ['db-host.example.de'],),
]
```

Puppet Tags im host.erb Template

```
extra_host_conf['notes_url'] += [
("/<%= omd_site -%>/wiki/doku.php?id=nagios_doku:hosts:<%= hostname -%>", ['<%= fqdn -%>'], ),
]
```

Beispiel:

```
extra_host_conf['notes_url'] += [
("/omd/wiki/doku.php?id=nagios_doku:hosts:db-host", ['db-host.example.de'], ),
]
```



Automatische „Inventur“ + „Reload“ von Check_MK

```
all_hosts += [

    " <%= fqdn -%>|CG_DEFAULT | CG_customer | <%=
architecture -%>|
<%= mktags.join('|') %> ",

]
```

File: $CMK_CONF/$FQDN.mk
Exec: CMK -I $FQDN + CMK -O via „exportierter Resource“

Advance Agent-Usage

- Nutzung von Puppet-Concat um Agent-Konfig-Files aus Fragmenten zu erstellen
  - /etc/check_mk/mrpe.cfg
  - /etc/check_mk/logwatch.cfg
  - /etc/check_mk/fileinfo.cfg

Check_MK MRPE

Nagios NRPE Ersatz, sehr gut geeignet um lokale Nagios-Checks mit den Nagios-Plugins durch zuführen. Leider werden alle Checks mit dem gleichen Interval ausgeführt wie der Agent (default= 1min), es gibt aber einen Patch um dieses zu ändern

→ http://mathias-kettner.de/checkmk_mrpe.html

Puppet-Concat „mrpe.cfg“

```
# Build mrpe.cfg from Fragments !!! (TARGET Konfiguration)
  concat{'/etc/check_mk/mrpe.cfg':
    Mode    => '0644', owner   => root, group   => root,
    require => File['/etc/check_mk'],
  } # concat
  concat::fragment{'mrpe.cfg_header':
    target  => '/etc/check_mk/mrpe.cfg', TARGET muss vorher definiert sein !!!
    content => „###This file is Puppet managed !!!“,
    order   => 01,
  } #concat::fragment
  concat::fragment{'mrpe.cfg_User_Logged_in':  Fragmente können überall erstellt werden.
    target  => '/etc/check_mk/mrpe.cfg',                Diese werden dann zusammen gesetzt in einer Datei.
    content => "User_Logged_in /usr/lib64/nagios/plugins/check_users -w 4 -c 6",
    order   => 10,
  } #concat::fragment
```

Host_Parents

- Puppet kennt das Netz und das default Gateway
  - Fact „ipaddress“ + Custom-Fact „gateway“
  - check_mk TAG im Host.ERP Template

parent.mk:

```
parents = [
 ## Parent -> Hostname
 ( "Nagios-Parent-Name", [     "Nagios_Host_Name1" ] )
 ## Parent -> TAG
 ( "Nagios-Parent-Name2",        ["TAG1"], ALL_HOSTS ),
]
```

Hosts-Groups

- Erzeugen von dynamischen Host-Groups auf Basis des Custom-Facts „Puppet_classes_csv“
  - Alle Host, welche das Tag „mysql__server“ (das ist die Puppet-classe „mysql::server“ in umgewandelter Form) haben sollen in die Hostgrouppe MySQL-Server
- Erzeugen von dynamischen Host-Groups auf Basis der IP (Front-/Back-/Verwaltungs-LAN)
- Erzeugen von dynamischen Host-Groups auf Basis des Environment (DEV; Stage; QA; Prod)

Plugin_Checks

- Check_MK Systemtime:
  Interessant bei VMs, da dort kein NTP vorhanden sein muss.

/usr/lib/check_mk_agent/plugins/systemtime:

```
#!/bin/bash
 /bin/echo '<<<systemtime>>>'
 /bin/date '+%s'
```

Ein einfacher Check ob die Systemtime OK ist.

Weitere Plugin_Checks

- apache_status / nginx_status
- check_puppet
  (Status des Puppet-Agent)
- mk_logwatch
- mk_postgres / mk_mysql
- mk_job
  (Überwachung von Cronjobs)

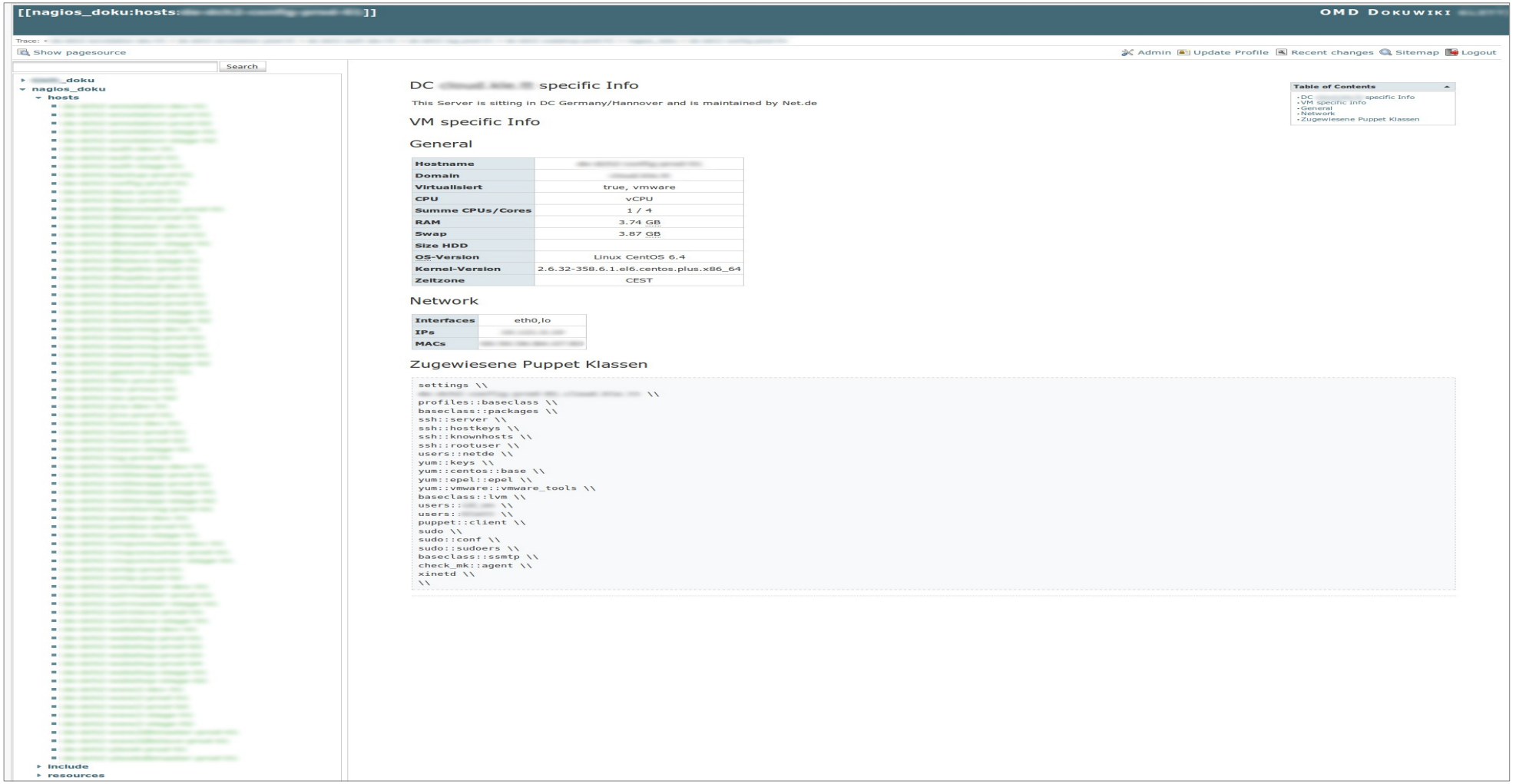
Weitere Plugin_Checks

- agent_vsphere
  (VMWare ESXi + VCenter)
- agent_emcvnx
  (EMC VNX Storage)
- Check_ipmi remote
  (iLO / iDRAC with own interface)
- check_bi_local
  (Alarmierung auf check_mk_bi Ergebnisse)

Konfigurationsmanagement mit Puppet

- Grundlagen des Konfigurationsmanagement
- Was kann ein Konfigurationsmanagement und was nicht?
- Wofür ist ein Konfigurationsmanagement und wofür besser nicht?
- Best Practices beim Umgang mit Puppet
- Übersicht der Konfigurations-Möglichkeiten
- Möglichkeiten der Organisation von Puppet-Konfiguration
- Praxis-Beispiele zur Nutzung von Puppet mit OMD/Nagios/Check_MK
- **Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation**

Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation



Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

```
==== General ====
^ Hostname |  <%= hostname -%>  | ^ Domain   |  <%= domain -%>   |
^ Virtualisiert |   <%= is_virtual -%>, <%= virtual -%>   |
^ CPU      |  vCPU  | ^ Summe CPUs/Cores |   <%= physicalprocessorcount -%> / <%= processorcount -%>   |
^ RAM      |   <%= memorysize -%>      | ^ Swap     |   <%= swapsize -%>     |
^ Size HDD  |  <% if defined? lvm_pv_size %><% then %><%= lvm_pv_size %><% end %>   |
^ OS-Version  |   <%= kernel -%> <%= operatingsystem -%> <%= operatingsystemrelease -%>      |
^ Kernel-Version   |   <%= kernelrelease -%>   |
^ Zeitzone |   <%= timezone -%>  |
==== Network ====
^ Interfaces |  <%= interfaces -%>    |
^ IPs |  <%= ipaddress  -%>  |
^ MACs |  <%= macaddress  -%>  |
==== Zugewiesene Puppet Klassen ====
<% classes.each do |klass| -%>
        <%= klass -%> \\
<% end -%>
```

```
check_mk::agent.pp:

@@file { "${wiki_confdir}/hosts/${::hostname}.txt":
  content => template('check_mk/wiki_hosts.txt.erb'),
} #file
```

http://www.nagios-wiki.de/nagios/howtos/wiki_hardware

Praxis-Beispiele zur Nutzung von Puppet für eine automatische Dokumentation

DC [redacted] specific Info

This Server is sitting in DC Germany/Hannover and is maintained by [redacted]

VM specific Info

General

Hostname	[redacted]
Domain	[redacted]
Virtualisiert	true, vmware
CPU	vCPU
Summe CPUs/Cores	1 / 4
RAM	3.74 GB
Swap	3.87 GB
Size HDD	
OS-Version	Linux CentOS 6.4
Kernel-Version	2.6.32-358.6.1.el6.centos.plus.x86_64
Zeitzone	CEST

Network

Interfaces	eth0,lo
IPs	[redacted]
MACs	[redacted]

Zugewiesene Puppet Klassen

```
settings \\
[redacted] \\
profiles::baseclass \\
baseclass::packages \\
ssh::server \\
ssh::hostkeys \\
ssh::knownhosts \\
ssh::rootuser \\
users::netdb \\
yum::keys \\
yum::centos::base \\
yum::epel::epel \\
yum::vmware::vmware_tools \\
baseclass::lvm \\
users::[redacted] \\
users::[redacted] \\
puppet::client \\
sudo \\
sudo::conf \\
sudo::sudoers \\
baseclass::ssmtp \\
check_mk::agent \\
xinetd \\
\\
```

ENDE

Fragen ???