Workshop | MSP AG

Aufbau einer zentralen Konfigurationsverwaltung

Von: Marcus Gaffke
Am: 28.05.2019

Inhaltsverzeichnis

1 Allgemeine Informationen

1.1 Vorwort

Diese Dokumentation ist Zwecks eines Workshops für den OpenSource Day's an der ITECH Wilhelmsburg entstanden. Dieser Workshop soll auch nach dem OpenSource Day weiter an der MSP AG angeboten werden. ((Dieser Workshop wurde auch den Auszubildenden der MSP AG zum Versuchen gegeben.))

Sollten zu irgendeinem Zeitpunkt, Daten aus dem Projektdokumenten benötigt werden, wird das durch Hinweise wie diese angezeigt!

In dieser Dokumentation wurde hauptsächlich mit Virtuellen Maschinen gearbeitet, welche Debian 9 (Stretch) betreiben.

1.2 Arbeitsumgebung

Für die Ausführung dieses Workshops werden…

- *Ein Git-basiertes Versionskontrollsystem wie GitHub oder Gitlab CE,*
- *3 Linux-Basierte VMs, welche miteinander kommunizieren können, gebraucht.*

Optional sollten die 3 VMs auch eine aktive Internetverbindung haben, um spätere Modul- und Paketinstallationen zu vereinfachen. Ebenso optional kann ein DNS-Server verwendet werden, um spätere Identifikation von Maschinen zu vereinfachen.

1.3 Ziel

Ziel dieses Workshops ist eine zentrale Konfigurationsverwaltung mithilfe der quelloffenen Version von Puppet 6.4 aufzubauen. Dazu wird ein Puppetserver aufgebaut, welcher mit einem Versionskontrollsystem (wünschenswert Gitlab CE) und mit mindestens 2 Agent-Maschinen kommunizieren kann.

Aus dem Versionskontrollsystem sollen Zustände von der Server- und den Agent-Maschinen bezogen und in den Puppetserver hinterlegt werden. Durch sogenannte Puppet-Agents auf der Server- und den Agent-Maschinen werden dann diese Zustände vom Puppetserver bezogen und auf den jeweiligen Maschinen abgebildet.

1.4 Vorteile

Durch diesen Aufbau ist es möglich mehrere Maschinen, aus einem zentralen Punkt zu verwalten, um zukünftige Softwareinstallationen, Konfigurationen und Wiederherstellungen von Maschinen deutlich zu beschleunigen.

Dazu kommt auch, dass durch die eingebundene Versionskontrollsystem eine Versionskontrolle sowie eine Benutzerverwaltung dieser Maschinenkonfigurationen gegeben ist.

2 Vorbereitung

2.1 Versionskontrollsystem

Um den Maschinenkonfigurationen einen Platz mit grafischer Oberfläche, Versionskontrolle und Benutzerverwaltung zu geben, werden diese mit einem Git-basierenden Versionskontrollsystem wie *GitHub* oder *Gitlab Community Edition* angebunden. Es ist für jedem Teilnehmer frei zu entscheiden, ob diese/r Gitlab CE oder lieber GitHub nutzen möchte.

Grundsätzlich gilt für beide Systeme:

- Es ist ein Projekt unter dem gewählten System zu erstellen (z.B. *Puppet* oder *Base*)
- Standard Repository zu „production“ ändern
- In diesem Projekt sind die SSH-Schlüssel (oder ähnliche Push-/Pull-Zugänge) zu hinterlegen:
  - Einer für den neuen Puppetserver, der die Zustände beziehen muss zur Weiterverteilung
  - Einer für jede Arbeitsmaschine, die diese Zustände beeinflussen möchte
- Testweise das neue Projekt auf alle Maschinen klonen, um die Zugänge zu testen

---

Für das Projekt ist die Ordnerstruktur „production“ in code/environments/ aus dem GitHub zu entnehmen und zu im neuen Projekt commiten! Dadurch wird die Puppet-Konforme Umgebungsstruktur erstellt.

2.2 Puppet/Virtuelle Maschinen

Da Puppet seine Nodes (inkl. Puppetserver) anhand von Hostnamen oder FQDNs identifiziert, wird die Hosts-Datei auf jeder Maschine editiert, sollte kein DNS-Server die Namens-Auflösung übernehmen.
Dazu wird der Hostname „puppet“ (welcher, standardmäßig von Puppet‘s Konfiguration verwendet wird,) zur IP-Adresse des Puppetservers aufgelöst:

```
#root@puppetmaster:/etc/hosts

127.0.0.1               localhost
127.0.0.1               puppetmaster.localdomain
192.168.0.1             puppet
```

Beispiel-Hosts-Datei auf einem Puppetserver

Anschließend muss nun das Paketverzeichnis passend zur Linux-Distribution eingebunden werden. Für Debian 9 (Stretch) wird das .deb Paket unter https://apt.puppet.com/puppet6-release-stretch.deb zum /tmp/-Ordner der jeweiligen Maschine bezogen und installiert:

```
root@puppetmaster:/tmp# wget https://apt.puppet.com/puppet6-release-
stretch.deb
root@puppetmaster:/tmp# dpkg -i puppet6-release-stretch.deb
```

Befehle zum Installieren der Paketquellenverweise unter Debian 9

- *Für andere Linux-Distributionen kann die „Puppet Platform“-Seite unter: https://puppet.com/docs/puppet/6.4/puppet_platform.html#task-383 konsultiert werden.*

Abschließend werden noch die Paketinformationen aller nun vorhandenen Paketquellen aktualisiert:

```
root@puppetmaster:/tmp# apt update
```

Befehl zur Aktualisierung der Paketquelleninformationen unter Debian 9

Dadurch sind nun die virtuellen Maschinen bereit für die eigentliche Installation der Puppet-Umgebung.

3 Installation

3.1 Puppetserver

Auf der Servermaschine müssen nun das „**Git**“- und „***puppetserver***“-Paket installiert werden. Anschließend muss die Root- sowie Intermediate-Zertifizierungsstelle für spätere SSL-Signierungsanfragen vom Puppetserver generiert werden:

```
root@puppetmaster:/tmp# apt install git
root@puppetmaster:/tmp# apt install puppetserver
…
root@puppetmaster:/tmp# puppetserver ca setup
```

Befehle zur Installation des Puppetserver-Paketes und der Erstellung der Root- und Intermediate-Zertifizierungsstelle

Nach der erfolgreichen Installation des Paketes können in der Datei **/etc/default/puppetserver** (Unter Debian 9) die Java-Startparameter des Puppetservers bearbeitet werden, um z.B. den zugewiesenen Arbeitsspeicher zu konfigurieren:

```
#/etc/default/puppetserver

…
# Modify this if you'd like to change the memory allocation, enable
JMX, etc
JAVA_ARGS="-Xms2g -Xmx2g"
```

Ausschnitt von /etc/default/puppetserver aus einem Beispielaufbau

Sobald der Puppetserver konfiguriert wurde, kann dessen Dienst über *systemctl/service* gestartet werden:

```
root@puppetmaster:/tmp# service puppetserver start
```

Befehl zum Starten des puppetserver-Dienstes unter Debian 9

» Vorsicht! Sollte der Dienst gestartet werden, bevor die Root- und Intermediate-Zertifizierungsstellen generiert wurden, wird der Puppetserver stattdessen nur eine Root-Zertifizierungsstelle generieren, welche dann von Puppet-Agents und entsprechenden Tests als fehlerhaft durch die fehlende Intermediate-Zertifizierungsstelle, gemeldet wird.

Sobald der Start des Dienstes erfolgreich war, kann über einen lokalen Puppetrun die Funktionalität getestet werden:

```
root@puppetmaster:/tmp# puppet agent -t
```

Befehl für einen Puppetrun auf aktueller Maschine. Das Argument -t liefert eine verbose Ausgabe.

Die Grundinstallation des Puppetservers wäre damit abgeschlossen, sobald der Puppetrun problemfrei abgelaufen ist.

3.2 Agent-Instanzen

Auf jeder Agent-Maschine muss nun das Paket „***puppet-agent***“ installiert werden:

```
root@agent01:/tmp# apt install puppet-agent
```

Befehl zur Installation des „puppet-agent“-Paketes

Sobald das Paket installiert wurde und die jeweilige Hosts-Datei bzw. der optionale DNS-Server korrekt den Hostnamen „puppet“ auf die IP-Adresse des Puppetserver auflöst, verbindet sich auch der Agent direkt mit dem Puppetserver und versucht Zustände zu beziehen und abzubilden.

Dennoch empfiehlt es sich einen verbosen Puppetrun zu starten, um die Funktionalität zu des Agents zum Puppetserver zu testen:

```
root@agent01:/tmp/# puppet agent -t
```

Befehl für einen Puppetrun auf aktueller Maschine. Das Argument -t liefert eine verbose Ausgabe.

Die Installation der Agent-Instanzen ist fertiggestellt, sobald alle Instanzen ihren Testlauf erfolgreich abgeschlossen haben und deren Zertifikatssignierungsanfragen an den Puppetserver signiert wurden:

```
root@puppetmaster:# puppetserver ca list –all
root@puppetmaster:# puppetserver ca sign AgentXY
```

Befehle zum Anzeigen aller Zertifikate (zum Erfassen von Namen) und zum Signieren der Anfrage von AgentXY

3.3 R10k – Anbindung an das Versionskontrollsystem

Puppet bietet nativ zunächst keine grafische Oberfläche sowie keinerlei Versionskontrolle über die gespeicherten Zustände. Allerdings hat dessen Entwickler, Puppet Labs, hierzu eine Lösung entwickelt, welche zusätzlich zum Puppetserver installiert werden kann.

R10k (, kurz für Robot10000,) ist ein Ruby Gem, welches einem Puppetserver erlaubt die lokal abgespeicherten Zustände durch bezogene Zustände, aus zuvor definierten Git- oder SVN-Quellen, zu überschreiben.

Um das Ruby Gem zu installieren muss unter **/opt/puppetlabs/puppet/bin/** beim Puppetserver das Programm „**gem**“ mit der Installationsanweisung gestartet werden.

R10k selbst muss dann noch dementsprechend über die PATH-Umgebungsvariable auffindbar sein, um über die Kommandozeile gestartet werden zu können, ohne komplett zu R10k navigieren zu müssen.

Dazu reicht es völlig aus R10k über einen symbolischen Link im Ordner **/opt/puppetlabs/bin** zu erwähnen, da sich der Pfad nach der Puppetserver-Installation in der PATH-Umgebungsvariable befindet:

```
root@puppetmaster:/opt/puppetlabs/puppet/bin# gem install r10k
…
root@puppetmaster:/opt/puppetlabs/bin# ln -s
/opt/puppetlabs/puppet/r10k r10k
```

Installationsanweisung für das Programm „gem“ sowie symbolischer Link von /opt/puppetlabs/bin/r10k zu /opt/puppetlabs/puppet/r10k

Ist die Installation erfolgreich gewesen, wird die Funktionalität mit folgendem Befehl getestet:

```
root@puppetmaster:/# r10k help
```

Sollte sich die Hilfeseite von R10k ohne Fehler öffnen, ist alles ordnungsgemäß installiert.

In /etc/puppetlabs/ muss nun ein Ordner r10k erstellt in der eine Datei r10k.yaml erstellt werden soll. Die r10k.yaml-Datei aus den Projektdokumenten dient hier lediglich als Beispiel, da hier die Git-Adressen aus den neuen Projekten aus Schritt 2.1 eingefügt werden müssen.

---

4 Gebrauchsanweisung

4.1 Deployment von Puppetcode in die Agent-Maschinen

4.1.1 R10k – Deployment des Puppetcodes an den Puppetserver

Damit die Projektdaten nun auf dem Masterserver bezogen werden, sollen muss nur noch dem R10k der Deployment-Befehl übergeben werden:

```
root@puppetmaster:/# r10k deploy environment -p
```

Dadurch liest R10k nun die gegebenen Konfigurationsdateien aus:

- Die zuvor erstellte **r10k.yml** in **/etc/puppetlabs/**
- Das **„Puppetfile“** in den Projektdaten, welches die Bezugsorte von externen Modulen beschreibt (näher beschrieben in Schritt 4.2.2)

…Und baut daraus die Puppet Umgebung in **/etc/puppetlabs/code/environments/** nach und bezieht die in der **„Puppetfile“** definierten externen Module nach **/etc/puppetlabs/code/environments/production/modules/**.

Dadurch ist der gesamte Puppetcode auf dem Puppetserver und wird automatisch alle 30 Minuten von allen signierten Agent-Instanzen bezogen.

Um den R10k-Deployment Prozess zu automatisieren, kann nun ein sog. Commit-Hook verwendet werden (für die, die sich eher mit Git auskennen), oder auch einfach ein Cronjob verwendet werden, um den neusten Zustand des Puppetcodes auf dem Puppetserver zu beziehen.

4.2 Arbeiten mit Puppet

4.2.1 Site.pp – Puppet‘s Umgebungsdefinition

Die **Site.pp** beschreibt die Umgebung für den Masterserver. In dieser Datei können **Maschinen** gruppiert und dann **Gruppen-weit** angepasst werden (z.B. durch Profile/Module mit Puppetcode) aber auch **Umgebungs-weit** angepasst werden, sollten z.B. SSH-Schlüssel von bestimmten Nutzern auf allen Maschinen eingerichtet werden, ungeachtet dessen, welche Rolle eine Maschine zu erfüllen hat.

In den Projektdaten haben wir, in Schritt 2.2, eine Beispielumgebung eingerichtet, die die Beispiele nochmal genauer darstellt.

4.2.2 Puppetfile – R10k’s Definitionsdatei für externe Module

Die **Puppetfile**, die sich in der Puppet Umgebung befindet ist eine weitere Datei, die von R10k in Betracht gezogen wird, wenn die Umgebung zum Puppetserver deployed wird. Sie besteht nur aus einer **Git-Quellen-Liste** von externen Modulen die zusätzlich in die Umgebung mit bezogen und verwendet werden können.

In Schritt 2.2 wurde aus den Projektdaten, eine Beispieldatei in der Umgebung hinterlegt, die natürlich angesehen werden kann. Dadurch wird ein Beispielmodul bezogen, welches dann in der Umgebung verwendet werden kann.

4.2.3 Hiera – Puppet‘s Wertedatenbank

Puppet bietet die Verwendung der sogenannten Hiera-Funktion. Die Hiera an sich ist eine Wertedatenbank, dessen Struktur, Dateiformat und Dateinamen durch die **hiera.yml** in der Puppet Umgebung definiert werden.

Durch die Arbeit in Schritt 2.2 wurde die Hiera so konfiguriert, dass die Dateien pro Host (deren Signaturname/FQDN) als **JSON-Datei** im **Hiera-Ordner** der Puppet Umgebung abgespeichert werden und als solche dementsprechend behandelt werden.

```
root@puppetmaster:/etc/puppetlabs/code/environments/hiera:# ls
Beispielmaschine.localdomain.json
Beispielmonitoringserver.localdomain.json
Beispielwebserver.localdomain.json
```

Beispieldateien eines Hiera-Ordners

Dadurch können zum Beispiel, Konfigurationsdateien als Schablonen verwendet werden können und dessen Werte durch die Hiera übergeben werden. Oder auch Module, die sich an Hiera-Werten bedienen ähnlich zu manipulieren.

Als Beispiel dient das sogenannte **Monitoring.pp** welches in Profile-Modul in **/etc/puppetlabs/code/environments/production/site/profile/manifests/monitoring.pp** Befindet.
Darin wird eine Testdatei erstellt, dessen Inhalt aus der Variable *$testcontent* kommt, welche in der Hiera des jew. Hosts zu finden sein sollte. Sollte der Host keine Variable *$testcontent* haben, schlägt das Profil dank der in Zeile 3 gesetzten Anforderung, einfach fehl.

4.2.4 Dynamische Environments durch Git-Branches

Wie schon sicher aufgefallen ist, ist unter **/etc/puppetlabs/code/environments**/ nur der Ordner **„production“** zu finden.

Über R10k besteht die Möglichkeit einfach neue Umgebungen für z.B. Test- und Stagingzwecke zu erstellen.

Vorsicht: Die Verwendung dieser Funktion kann Speicherplatzintensiv für den Puppetserver werden, da es sich hier um jew. vollständige Kopien aus dem Git handelt.

Um eine solche Umgebung zu erstellen, muss im Git-Projekt einfach eine neue Branch erstellt werden und dann wie gewohnt über R10k ausgerollt werden. Dadurch sollte nun ein Ordner mit dem Branch-Namen unter dem oben genannten Verzeichnis zu finden sein.

Um diese Umgebung für eine Agent-Instanz zu verwenden gibt es nun 2 Möglichkeiten:

A.) Nur für den aktuellen Puppetrun verwenden:

Um die neue Puppet Umgebung einmalig an einer Agent-Instanz zu testen, kann über den Agent-Befehl die Umgebung für den Puppetrun mitgeliefert werden:

```
root@puppetmaster:/tmp# puppet agent -t --environment <Branch-Name>
```

B.) Die aktuelle Puppet-Instanz auf ein Environment festlegen:

Ist es notwendig, eine Maschine konstant in einer anderen Puppet Umgebung als der „production“ zu halten (z.B. Test- oder Entwicklungsmaschinen), kann in der Konfigurationsdatei **/etc/puppetlabs/puppet/puppet.conf** des jew. Hosts unter dem Punkt **„[agent]“** der Parameter **„environment = <Branch-Name>“** gesetzt werden.

4.2.5 Module aus dem „Puppet Forge“ Installieren und Verwenden

Unter https://forge.puppet.com/ bietet eine Plattform an, in den Personen aus aller Welt deren Puppet-Module teilen können. Häufig sind gängige Lösungen wie Webserver-Installationen u.Ä. hier zu finden.

Um in diesem Aufbau, Module aus der Puppet Forge zu installieren muss nur das Modul auf dem Puppetserver über den entsprechenden Befehl installiert werden. Durch die Arbeit in Schritt 2.2, entstandene **„environment.conf“** in der Puppet Umgebung weiß Puppet, wohin es Module installieren muss, damit diese verwendet von der Umgebung werden können.

```
root@puppetmaster:/tmp# puppet module search apache
root@puppetmaster:/tmp# puppet module install puppetlabs-apache
```

Such- und Installationsbefehle der Module-Funktion von Puppet

4.2.6 Eigene Module schreiben

Natürlich kann man neben der Site.pp und auch eigene Module schreiben und diese in der **Site.pp (Oder sogar in andere Module!) einbinden**. Dazu muss für jedes Modul ein Ordner unter dem Ordner **„Site“** erstellt werden (ähnlich wie „Profile“ aus den Projektdaten).
Wie ein Modul nun erstellt wird, wird in dieser Dokumentation leider nicht weiter erläutert. Dazu wendet man sich am besten an die Dokumentation von Puppetlabs selbst (in der Quellenangabe zu finden) oder bezieht sich Beispiele aus anderen Modulen.

5 Quellenverzeichnis

Using Puppet Platform -- Puppet.com: Puppetlabs, Using the Puppet platform, 2019, https://puppet.com/docs/puppet/6.4/puppet_platform.html

Puppet Server -- Puppet.com: Puppetlabs, Puppet Server: Installing from Packages, 2019, https://puppet.com/docs/puppetserver/6.4/install_from_packages.html

Installing Agents -- Puppet.com: Puppetlabs, Installing Agents, 2019, https://puppet.com/docs/puppet/6.4/install_agents.html

R10k Github -- Github.com: Puppetlabs, GitHub - puppetlabs/r10k, 2019, https://github.com/puppetlabs/
r10k

Puppet modules – Puppet.com: Puppetlabs, Puppet modules, 2019, https://puppet.com/docs/puppet/6.4/modules.html