Contents

Puppet walkthrough

Inhaltsverzeichnis

Best Practices für das Testen von Komponenten

Was ist ein Puppet Modul

- Ein normales Puppet Modul kann man auch als Komponentenmodul bezeichnen
- Es bündelt Puppet Klassen, welche eine einzelne Komponente verwalten (z.B. nginx oder Apache oder HAProxy)
  - Eine Klasse bündelt Puppet Ressourcen in einem Namespace. Dies hat wenig Klassen aus einem OOP Kontext zu sein
  - Man sollte eine Klasse in Puppet nicht mit einer Klasse in Java oder C# vergleichen
- Jedes Komponentenmodul sollte ein eigenes Git Repository haben
- Jedes Komponentenmodul ist nicht nur ein Puppet Projekt, sondern auch ein Ruby Projekt (weil `puppet` in Ruby geschrieben ist)
  - Ruby Projekte haben immer ein Gemfile, dies listet Ruby Abhängigkeiten für die Laufzeit, für Tests und für die Entwicklung
  - Ruby Projekte haben meistens ein Rakefile, dort sind Tasks definiert (ähnlich einem Makefile)
  - Ruby Projekte testet man (unter anderem) mit Ruby Tools

Näheres dazu unter puppetlabs_spec_helper.

Warum will man Linter

- In Programmiersprachen, egal ob Puppet DSL, Ruby, Rspec DSL oder andere, gibt es viele Schreibweisen für das selbe Ergebniss
- Onboarding neuer Entwickler ist einfacher, wenn Code einheitlich ist
- Updates sind mit einheitlichem Code einfacher
  - Code auf eine neue Ruby Version portieren
  - Code auf eine neue Puppet Version portieren
- In jedem Ökosystem gibt es eine Community welche sich Style Guides ausdenkt, diesen kann man mit Lintern anwenden/erzwingen
- In einer CI Pipeline sind Linter gut, da sie sehr schnell arbeiten und direkt Feedback liefern
  - Optional auch als pre-commit Hook lokal ausführen
- Ein Linter prüft nicht zwingend den Code auf Syntax Fehler

rspec

- rspec ist ein unit testing framework in Ruby. Es bringt eine eigene DSL mit
- rspec-puppet ist eine Erweiterung für rpsec, um Puppet code testen zu können
- `Unit Tests` sind Tests, welche einzelne Funktionen / minimale Codeblöcke testen
- `rspec-puppet` validiert, ob bestimmte Puppet Resourcen so im Katalog sind, wie man sie erwartet
  - korrekte Parameter, Reihenfolge der Resourcen, Prüfen ob der Katalog kompiliert

rspec Beispiele

Minimales Beispiel

```
# import a helper
require 'spec_helper'

# class we want to test
describe 'borg' do
  # mock an FQDN
  let :node do
    'rspec.puppet.com'
  end

  let :facts do
    {
      "operatingsystem": "CentOS",
      "operatingsystemmajversion": 7
    }
  end

  # mock class params if required
  # bad practice, a module should work with default data
  let :params do
```

```
    {
      backupserver: 'localhost'
    }
  end

  context 'with all defaults' do
    it { is_expected.to compile.with_all_deps }
  end
end
```

- Gefühlt 80% der Probleme kann man damit lösen. Die meisten Fehler führen dazu, dass der Code nichtmal kompiliert

Prüfen der einzelnen Resources:

```
require 'spec_helper'

describe 'borg' do
  let :node do
    'rspec.puppet.com'
  end

  on_supported_os.each do |os, facts|
    context "on #{os} " do
      let :facts do
        facts
      end

      let :params do
        {
          backupserver: 'localhost'
        }
      end

      context 'with all defaults' do
        it { is_expected.to compile.with_all_deps }

        it { is_expected.to contain_file('/etc/backup-sh-conf.sh') }
        it { is_expected.to contain_file('/etc/borg') }
        it { is_expected.to contain_file('/etc/profile.d/borg.sh') }
        it { is_expected.to contain_file('/usr/local/bin/borg-backup') }
        it { is_expected.to contain_file('/usr/local/bin/borg_exporter') }
        it { is_expected.to contain_file('/etc/borg-restore.cfg') }
        it { is_expected.to contain_class('borg::install') }
        it { is_expected.to contain_class('borg::config') }
        it { is_expected.to contain_class('borg::service') }
        it { is_expected.to contain_ssh__client__config__user('root') }
        it { is_expected.to contain_borg__ssh_keygen('root_borg') }
        it { is_expected.to contain_exec('ssh_keygen-root_borg') }
      end

      case facts[:os]['name']
      when 'Archlinux'
        context 'on Archlinux' do
          it { is_expected.to contain_package('borg') }
          it { is_expected.to contain_package('perl-app-borgrestore') }
        end
      when 'Ubuntu'
        context 'on Ubuntu' do
          it { is_expected.to contain_package('borgbackup') }
          it { is_expected.to contain_package('borgbackup-doc') }
          it { is_expected.to contain_package('gcc') }
          it { is_expected.to contain_package('make') }
```

```
          it { is_expected.to contain_package('cpanminus') }
          it { is_expected.to contain_package('libdbd-sqlite3-perl') }
          if facts[:os]['release']['major'] == '16.04'
            it { is_expected.to contain_apt__ppa('ppa:costamagnagianfranco/borgbackup') }
          end
        end
      when 'RedHat', 'CentOS'
        context 'on osfamily Redhat' do
          it { is_expected.to contain_package('perl-local-lib') }
          it { is_expected.to contain_package('perl-Test-Simple') }
          it { is_expected.to contain_package('perl-App-cpanminus') }
          it { is_expected.to contain_package('gcc') }
          it { is_expected.to contain_exec('install_borg_restore') }
          it { is_expected.to contain_file('/opt/BorgRestore') }
          it { is_expected.to contain_file('/usr/local/bin/borg-restore.pl') }
        end
      when 'Gentoo'
        context 'on osfamily Gentoo' do
          it { is_expected.to contain_package('App-cpanminus') }
        end
      when 'Fedora'
        context 'on osfamily Fedora' do
          it { is_expected.to contain_package('perl-Path-Tiny') }
          it { is_expected.to contain_package('perl-Test-MockObject') }
          it { is_expected.to contain_package('perl-Test') }
          it { is_expected.to contain_package('perl-autodie') }
        end
      end
    end
  end
end
```

Quelle ist das puppet-borg modul

- rspec Homepage
- rspec-puppet Homepage

rspec-puppet-facts

- Puppet Module nutzen fast immer den `$facts` Hash
- In Tests muss man die Daten also mocken
- Das ganze sollte konsistent (z.B. `CentOS` vs `centos`) in allen Modulen sein
- Am liebsten automatisiert durch alle Betriebssysteme in der metadata.json iterieren und für jedes OS die Facts mocken

rspec-puppet-facts macht genau das!

```
require 'spec_helper'

describe 'borg' do
  let :node do
    'rspec.puppet.com'
  end

  on_supported_os.each do |os, facts|
    context "on #{os} " do
      let :facts do
        facts
      end

      let :params do
        {
          backupserver: 'localhost'
        }
```

```
      end

      context 'with all defaults' do
        it { is_expected.to compile.with_all_deps }
      end
    end
  end
end
```

Es ist wichtig das alle Betriebssysteme, auf denen ein Modul genutzt wird, auch in der metadata.json stehen!

- rspec-puppet-facts Homepage

facterdb

- rspec-puppet-facts ermöglicht das iterieren über die metadata.json in rspec tests
- die gemockten factsets kommen aus dme facterdb Projekt
- Sammlung an Scripten + Vagrant configs um VMs zu starten und darin facter Versionen zu installieren
- Gesammelten Daten werden als Ruby gem released

Für alle genutzten Betriebssysteme sollten in dem Projekt factsets sein!

- FacterDB Homepage
- Facter documentation
- Bugreport: Facter 3.14 docs are broken

Beaker

- Acceptance testing framework für Puppet
- Startet Virtualbox/Docker Instanz, führt darin das Puppet Modul aus
- Idempotenz kann getestet werden
  - Puppet Modul zwei mal ausführen, zweiter run darf keine Änderungen zeigen
- Mit rspec/serverspec kann das System inspiziert werden
  - Prüfen ob Packete installiert sind
  - Ob Services gestartet sind und laufen
  - Ob TCP ports offen sind
  - Und vieles mehr

```
require 'spec_helper_acceptance'

describe 'zabbix::server class' do
  context 'default parameters' do
    # Using puppet_apply as a helper
    it 'works idempotently with no errors' do
      # this will actually deploy apache + postgres + zabbix-server + zabbix-web
      pp = <<-EOS
        class { 'postgresql::server': } ->
        class { 'zabbix::database': } ->
        class { 'zabbix::server': }
      EOS

      shell('yum clean metadata') if fact('os.family') == 'RedHat'

      # Run it twice and test for idempotency
      apply_manifest(pp, catch_failures: true)
      apply_manifest(pp, catch_changes: true)
    end

    # do some basic checks
    describe package('zabbix-server-pgsql') do
      it { is_expected.to be_installed }
    end

    describe service('zabbix-server') do
```

```
      it { is_expected.to be_running }
      it { is_expected.to be_enabled }
    end
  end
end
```

Quelle der Tests ist das voxpupuli/zabbix Modul.

- Langfristig wird OnyxPoint die Entwicklung von beaker übernehmen
- Puppet Inc. arbeitet an Litmus, einer Ruby Bibliothek um VMs/Container zu starten
  - Tests sollen hier mit Bolt ausgeführt werden
- Beaker Homepage
- Serverspec Homepage
- Beaker-Puppet Homepage
- Beaker-Docker Homepage
- Bolt Dokumentation

puppet-lint

- Ruby Projekt um verschiedene Styles in der Puppet DSL zu checken / erzwingen
- Bietet ein Plugin-System
  - Jeder Check ist ein Plugin / eigenständiges Ruby gem
  - Die meisten Plugins haben eine Autokorrekturfunktion

Liste der aktuell empfohlenen plugins:

Gemfile:

```
  gem 'puppet-lint-leading_zero-check',                          :require => false
  gem 'puppet-lint-trailing_comma-check',                        :require => false
  gem 'puppet-lint-version_comparison-check',                    :require => false
  gem 'puppet-lint-classes_and_types_beginning_with_digits-check', :require => false
  gem 'puppet-lint-unquoted_string-check',                       :require => false
  gem 'puppet-lint-variable_contains_upcase',                    :require => false
  gem 'puppet-lint-absolute_classname-check', '>= 2.0.0',        :require => false
  gem 'puppet-lint-topscope-variable-check',                     :require => false
  gem 'puppet-lint-legacy_facts-check',                          :require => false
  gem 'puppet-lint-anchor-check',                                :require => false
```

man kann in der Puppet DSL gezielt einzelne Linter deaktivieren, sofern sie ein False/Positive liefern:

```
# lint:ignore:case_without_default
```

Für Fehlermeldungen in einem sinnvollen Format kann man folgendes in seinem `Rakefile` eintragen:

```
PuppetLint.configuration.log_format = '%{path}:%{line}:%{check}:%{KIND}:%{message}'
```

Für neue Komponentenmodule sollte man auch puppet-lint-param-docs nutzen. Dies erzwingt Dokumentation für jeden Parameter.

Ausgabe mit Fehler:

```
$ bundle exec rake lint
manifests/init.pp:106:arrow_on_right_operand_line:WARNING:arrow should be on the right operand's line
```

Ausgabe mit Autokorrektur:

```
$ bundle exec rake lint:auto_correct
manifests/init.pp:106:arrow_on_right_operand_line:FIXED:arrow should be on the right operand's line
```

- puppet-lint Webseite
- Liste bekannter Community Plugins
- Liste der Vox Pupuli Plugins
- Vox Pupuli hat weitere Style Guides, wofür es noch keine linter gibt

puppet-syntax

- Checkt die Syntax der Puppet DSL mit `puppet parser`
- Checkt die Syntax von erb Templates mit Ruby

- Checkt die Syntax von EPP Templates mit `puppet`
- Checkt die Syntax von Hiera YAML Datein
  - Prüft auf valides YAML
  - Prüft auf korrekte Syntax der Keys, damit Puppet damit umgehen kann

Beispielhafte Ausgabe vom Rake Task wenn es keine Fehler gibt:

```
$ bundle exec rake syntax
  ---> syntax:manifests
  ---> syntax:templates
  ---> syntax:hiera:yaml
```

Ausgabe mit Syntaxfehlern:

```
$ bundle exec rake syntax
  ---> syntax:manifests
  rake aborted!
  Could not parse for environment production: Syntax error at end of file at demo.pp:2
  Tasks: TOP => syntax => syntax:manifests
  (See full trace by running task with --trace)
```

- puppet-syntax Homepage
- `puppet-parser` Dokumentation
- Dokumentation über ERB Templates in Puppet

RuboCop

- RuboCop ist ein Linter für Ruby Datein
- Er lintet keinen Puppet DSL Code, aber eigenes Types und Provider
- Er lintet alle Datein mit Tests
- RuboCop ist sehr schmerzhaft...
  - Hat aber eine autofix Option die relativ gut funktioniert

Ausgabe mit Fehlern:

```
$ bundle exec rake rubocop
Running RuboCop...
Inspecting 4 files
..C.

Offenses:

spec/spec_helper.rb:8:1: C: Layout/EmptyLines: Extra blank line detected. (https://github.com/bbatsov/ruby-style

4 files inspected, 1 offense detected
RuboCop failed!
```

Mit Autokorrektur:

```
 bundle exec rake rubocop:auto_correct
Running RuboCop...
Inspecting 4 files
Parser::Source::Rewriter is deprecated.
Please update your code to use Parser::Source::TreeRewriter instead
..C.

Offenses:

spec/spec_helper.rb:8:1: C: [Corrected] Layout/EmptyLines: Extra blank line detected. (https://github.com/bbatsc

4 files inspected, 1 offense detected, 1 offense corrected
```

- RuboCop Documentation

puppetlabs_spec_helper

- gem von Puppet Inc.

- Hat sehr viele der oben genannten Tools als Abhängigkeit
- Puppet kümmert sich um korrekte Versionen die untereinander kompatibel sind. Man muss nur noch `puppetlabs_spec_helper` einbinden
- Viele Rake Tasks sind bereits mit dabei

Gemfile:

```
source ENV['GEM_SOURCE'] || "https://rubygems.org"
gem 'puppetlabs_spec_helper'
```

Rakefile:

```
require 'puppetlabs_spec_helper/rake_tasks'
```

Alle Rake Tasks ausgeben:

```
bundle exec rake -T
```

```
rake beaker                    # Run beaker acceptance tests
rake beaker:sets               # List available beaker nodesets
rake beaker:ssh[set,node]      # Try to use vagrant to login to the Beaker node
rake build                     # Build puppet module package
rake build:pdk                 # Build Puppet module with PDK
rake build:pmt                 # Build Puppet module package with PMT (Puppet < 6.0.0 only)
rake check:dot_underscore      # Fails if any ._ files are present in directory
rake check:git_ignore          # Fails if directories contain the files specified in .gitignore
rake check:symlinks            # Fails if symlinks are present in directory
rake check:test_file           # Fails if .pp files present in tests folder
rake clean                     # Clean a built module package
rake compute_dev_version       # Print development version of module
rake help                      # Display the list of available rake tasks
rake lint                      # Run puppet-lint
rake lint_fix                  # Run puppet-lint
rake parallel_spec             # Run spec tests in parallel and clean the fixtures directory if successful
rake parallel_spec_standalone  # Parallel spec tests
rake release_checks            # Runs all necessary checks on a module in preparation for a release
rake rubocop                   # rubocop is not available in this installation
rake spec                      # Run spec tests and clean the fixtures directory if successful
rake spec:simplecov            # Run spec tests with ruby simplecov code coverage
rake spec_clean                # Clean up the fixtures directory
rake spec_clean_symlinks       # Clean up any fixture symlinks
rake spec_list_json            # List spec tests in a JSON document
rake spec_prep                 # Create the fixtures directory
rake spec_standalone           # Run RSpec code examples
rake syntax                    # Syntax check Puppet manifests and templates
rake syntax:hiera              # Syntax check Hiera config files
rake syntax:manifests          # Syntax check Puppet manifests
rake syntax:templates          # Syntax check Puppet templates
rake validate                  # Check syntax of Ruby files and call :syntax and :metadata_lint
```

Wie veröffentlicht man erfolgreich Module und welchen Mehrwert bietet es?

Wie strukturiert man in der Firma seinen Code sinnvoll?

Anforderungen:

- Irgendwie muss man die ganzen Komponentenmodule seinen Systemen zuweisen
- Man möchte öffentliche Komponentenmodule parallel zu selbst entwickelten nutzen
- Einige Systeme sind sehr ähnlich, aber doch anders
  - Jeder server hat SSH Passwort Authentifizierung deaktiviert, nur einer nicht
- Leute möchten Daten schreiben, aber sollen keinen Puppet Code modifizieren müssen/dürfen
  - Z.b. festlegen welche Version von interner Software wo läuft

Dies lässt sich mit dem Roles and Profiles Pattern lösen

- Ein Komponentenmodul kapselt Funktionalität um eine einzige Komponente zu verwalten

- Ein Profil enthält “Business-Logik”
  - Es kann die Reihenfolge von Komponentenmodulen setzen
    - z.B. epel Komponentenmodul vor Bird Komponentenmodul ausführen
  - Es holt sich Business Daten via Hiera
    - explizite `lookup()` Aufrufe oder Automatic Class Parameter Lookup Pattern
  - Mehrere solche Profile werden in einer Rolle gekapselt
  - Ein Profil kann in mehreren Rollen sein
  - Einem System wird immer nur eine Rolle zugewiesen
  - Profile und Rollen sind jeweils eigene Klassen

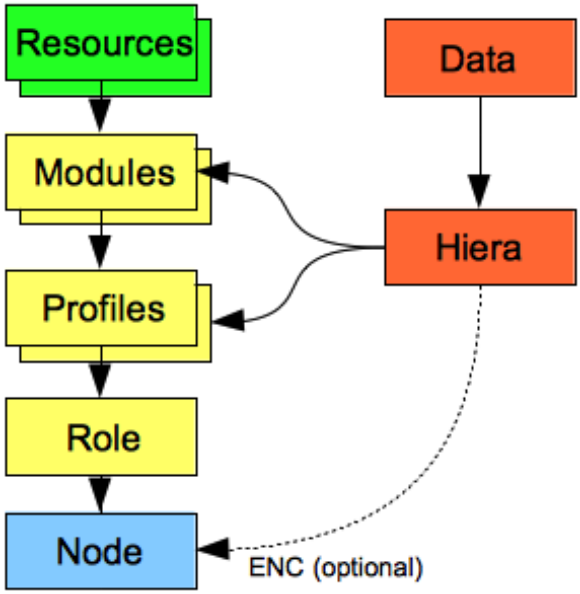

Figure 1: roles-and-profiles-and-hiera

(Copyright Craig Dunn)

Wie sieht ein gutes Komponentenmodul aus

- `include modul` funktioniert und installiert mir eine Komponente mit sinnvollen und sicheren Standardwerten
- Es gibt eine README.md
- Es gibt eine Lizenz
- Es enthält keine Business-Logik
  - Irgendetwas, was firmenspezifisch ist
  - Das Modul sollte in jeder Umgebung funktionieren
  - Ansonsten wird das Roles and Profiles Pattern nicht eingehalten
- Es gibt Unit- und Acceptance Tests
- Aktuelle Style Guidelines werden eingehalten
- Parameter mit `puppet-strings` dokumentiert

Mehrwert durch die Veröffentlichung von Komponentenmodulen

- Ein internes Modul nachträglich veröffentlichen kann Security Probleme mitbringen
  - Es kann potentiell Business-Daten enthalten (Passwörter, IP-Adressen. . .)
- Ist an Modul von Anfang an öffentlich:
  - achtet man auf eine sichere Implementierung
  - Enthält es nie Business-Logik. Diese wird dann im Profil implementiert. Dadurch is das Modul wiederverwendbar
- Sofern Dokumentation vorhanden ist, wird das Modul von anderen benutzt
  - Man bekommt plötzlich Bug Reports und Feature Requests, die potentiell mehr / unvorgergesehene Arbeit erzeugen
  - Man bekommt auch Pull Requests mit neuen Features und auch Bug Fixes

Es kommt vor, dass man ein öffentliches Modul patchen muss. Man muss versuchen diese Patches Upstream gemerged zu bekommen. Andernfalls difft der Fork immer mehr von Upstream ab und man landet in der Abhängigkeitshölle. Bei Modulen von Vox Pupuli / Puppet wird tendenziell schneller reagiert als bei Modulen mit einzelnen Maintainern. Puppet hat viele Mitarbeiter mit Commit Rechten für Ihre Module, Vox Pupuli hat > 140 (unregelmäßige) Maintainer. Außerdem gibt es hier Esakalationsmöglichkeiten.

Mehrwert durch Communities

- Vox Pupuli, Foreman, CERN, CamptoCamp, Puppet Inc und weitere Gruppen / Organisationen / Firmen pflegen viele Komponentenmodule
- Ein Team kümmert sich um alle generischen Datein in den Modulen
  - Einheitliche Testmatrix
  - Einheitliches Testsetup
  - Einheitliches Gemfile
- Die ganze Community hilft beim beantworten von Issues / Pull Requests
- Oftmals findet man hier schon ein fertiges Modul für seinen Usecase

modulesync

- modulesync wurde von Puppet Angestellten entwickelt
- Von Anfang an Open Source, mittlerweile von Vox Pupuli betreut
- In einem Git Repository liegen Templates, diese werden auf alle verwalteten Module angewand
- Änderungen kann man auf Modul Ebene überschreiben
- Eine Hand voll Leute können damit >120 Module verwalten
- Modulesync definiert auch die komplette Testmatrix für alle Module
- Viele Firmen nutzen die Vox Pupuli Konfigurationen für interne Module / man kann modulesync aber auch unabhängig benutzen

Weiterführende Dokumentation

- Erfindung des Roles and Profiles Pattern (Craig Dunn)
- Deepdive von Gary Larizza
- Roles and Profiles Dokumentation (Puppet Inc.)
- puppet-strings Dokumentation
- Vox Pupuli Module auf forge.puppet.com
- Modulesync Dokumentation
- Modulesync Konfiguration für Vox Pupuli

Grundlagen für Continuous Integration

- Jede Änderung sollte getestet werden
- Testergebnisse sollten möglichst schnell sichbar sein
- Tests müssen reproduzierbar sein
- Tests müssen in einer sauberen Umgebung ausgeführt werden, z.B. nspawn/Docker Container oder einer VM
- Tests müssen auf einer zentralen Platform ausgeführt werden
  - Jenkins
  - Gitlab-CI
  - Travis
- Je mehr öffentliche Module man nutzt, desto weniger muss man lokal testen
  - Die einzelnen Communities haben eine eigene Testinfrastruktur mit passender testmatrix
- Acceptance Tests für Controlrepos sind komplex zu bauen. Onceover hilft dabei
  - Ordentliche Tests in allen Komponentenmodulen ist wesentlich einfacher
  - Acceptance Tests in jedem Komponentenmodul sollte Daten aus den eigenen Profilen enthalten
- Prüft jede metadata.json:
  - Alle benötigten Module müssen als Dependencies gelistet sein
  - Alle Betriebssysteme müssen dort gelistet sein
- Spec Tests müssen rspec-puppet-facts nutzen
- Alle genutzten Betriebsysteme müssen in facterdb sein
- Irgendwo muss noch ein Peer Review vorkommen

Grundlagen für interne CI

- Man benötigt eine Testmatrix. Für Unit Tests (im Optimalfall):
  - Gegen alle Ruby Versionen die man nutzt
  - Gegen alle Puppet Versionen die man nutzt
  - Gegen alle Betriebssysteme die man nutzt
- Automatische Lizenzprüfung aller Abhängigkeiten
- CPU Leistung: Viel hilft viel!
  - Wenn genug Leistung zur Verfügung steht, kann man für jeden Test eine Dockerinstanz parallel starten

- Abhängigkeiten: Fehler früh erkennen
  - In der Regel nutzt man für Unit Tests die Master Branches der Abhängigkeiten, um noch nicht releaste Inkompatibilitäten früh zu erkennen
  - Für Acceptance Tests nutzt man die neusten Releases der Abhängigkeiten
- Bei Neuen Modulversionen:
  - Die Abhängigkeiten in der ganzen Environment prüfen mit `puppet module list --environment production --tree`
  - In `/var/log/puppetlabs/puppetserver/puppetserver.log` nach Warnungen / Fehlern schauen

Vorteile einer CI Pipeline

- Steigerung der Codequalität
- Erzwingen einer möglichst einheitlichen Codebasis
- Potentielle Fehler/Bugs/unerwünschtes Verhalten erkennen bevor es passiert
- Inkompatibilitäten in der Zukunft erkennen und vorbeugen, damit diese nicht eintreten
- Gibt Mitarbeitern mehr Selbstbewusstsein Änderungen zu machen
  - Verringert auch die Einstiegshürde für andere Administratoren

Anforderungen an eine lokale CI/CD/CD Platform

- Ergebnisse müssen schnell und einfach ersichtlich sein
- Testumgebungen müssen schnell verfügbar sein
  - R10k nutzen zum ausrollen von Environments
  - tar.gz Artefakte oder git tags/commits deployen, keine branches
  - Git Server Webhooks nehmen und Testenvironments zu erstellen sobald Commits / Branches gepusht werden
    - Puppet Webhook Server + Choria / mcollective / Bolt zum ausrolen

SSH Multiplexing aktivieren um `git clone`s zu beschleunigen:

```
class profiles::ssh2git {
 # /root/.ssh/config entry to access git.internal
 ssh::client::config::user { 'gitaccess':
   ensure        => present,
   user          => 'root',
   user_home_dir => '/root',
   options       => {
     'Host git.internal' => {
       'User'           => 'git',
       'IdentityFile'   => '~/.ssh/id_ed25519...',
       'ControlMaster'  => 'auto',
       'ControlPath'    => '~/.ssh/ssh-%r@%h:%p',
       'ControlPersist' => '10m'
     },
   },
 }
 sshkey { 'git.internal':
   ensure       => 'present',
   target       => '/root/.ssh/known_hosts',
   type         => 'ecdsa-sha2-nistp256',
   key          => 'DATA',
 }
}
```

r10k mit mehreren Workern starten:

```
# puppet/r10k from Vox Pupuli
class { 'r10k':
  version   => '3.5.0',
  pool_size => 10,
}
```

CI Dokumentation

- Puppet Catalog Diff Viewer von CamptoCamp

- octocatalog-diff Homepage
- Installation von octocatalog-dif (Example42)
- Onceover Dokumentation
- Puppet Webhook Server Dokumentation
- Choria Webseite
- Vortrag von Kevin Paulisse (GitHub Inc.) über CI

Grundlagen Continuous Delivery und Continuous Deployment

Continuous Delivery

- Continuous Delivery - nach einem erfolgreichen CI Durchlauf Buildartefakte bauen und in einem Repository speichern
- Artefakte können verschiedenste Formen haben
  - Ruby gem / Python Package
  - Docker Image
  - rpm / deb Paket
  - Puppet Modul
  - Ein generisches .tar.gz Archiv

Continuous Delivery für Puppet Module

Es gibt viele Möglichkeiten Puppet Module zu speichern.

Folgende Projekte unterstützen Puppet Modul Releases(.tar.gz Archive):

- Artifactory
- Pulp
- Sonatype Nexus
- kindredgroup/puppet-forge-server)

Alternativ kann man mit git tags / commit ids im Puppetfile arbeiten.

Continuous Deployment

- Continuous Deployment - Build Artefakte möglichst automatisiert in die Produktivumgebung bringen

Mit Puppet eine große Menge an Scripten als File Resources verteilen ist ein Antipattern. Diese sollte man als Artefakte Paketieren. Puppet richtet dann das Repository auf den Zielsystemen ein. Dies gillt für alle Artefakte, welche Puppet deployen soll.

Bei vielen Projekten bietet sich ein gitflow-workflow an:

- Develop Umgebung läuft auf einem individuellen Dev Branch
- Test Umgebung läuft auf dem Develop Branch
- Staging/Integreation Umgebung läuft auf Master Branch
- Git Tags basieren auf Master Branch. Diese werden nach Produktion deployed

Dies funktioniert sehr gut für intern entwickelte Anwendungen die auf anderen internen Tools aufsetzen und wiederum von anderen Teams genutzt werden. Für jede Umgebung benötigt man eigene Repositories und Maschinen/Server/Docker Images. Puppet kann solche Umgebungen deployen. Puppet Code selbst mit so einem Workflow zu verwalten funktioniert in der Regel nicht / bringt viel zu viel Overhead mit sich.

Für Puppet hat sich in den meisten Organisationen ein Git Feature Branch Workflow als beste Lösung etabliert.

- Im Control-Repository ist der Standardbranch `production`. Dieser wird als Environment auf normalen Nodes genutzt
- Zum testen / entwickeln erzeugt man einen Feature Branch im Control-Repository + ggf im entsprechenden Modul
- Push eines Branches im Control-Repo erzeugt via hooks die Environment auf den Puppetservern
- Bei Merges wird Production neu deployed

Review der Puppetumgebung

Dedizierter Vortrag über Puppetserver Skalierung

Grundsätzliches zum Tuning

- PuppetDB / Puppetserver CA / Compile Puppetserver / Foreman / PostgreSQL / Choria haben verschiedene Anforderungen an Hardware und kann man gut auf einzelnen Systemen installieren

- Ein verteiltes System ist immer nur so stark wie ihr schwächstes Glied
- Tuning Optionen sind immer stark abhängig von der Anzahl der Resources pro Node / Runinterval / Funktionen
  - Die Werte sollten deshalb nicht blind übernommen werden
  - Es sollte immer nur ein Wert gleichzeitig geändert werden um die Auswirkungen besser zu verstehen

PuppetDB

Hiera Tuningoptionen für puppetlabs/puppetdb:

```
---
puppetdb::server::java_args:
  '-Xmx': '8192m'
  '-Xms': '2048m'
puppetdb::server::node_ttl: '14d',
puppetdb::server::node_purge_ttl: '14d',
puppetdb::server::report_ttl: '999d'
# default is 50
puppetdb::server::max_threads: 100
# default is processorcount / 2
puppetdb::server::command_threads: %{facts.processors.count}
# default is 4, have your database in mind
puppetdb::server::concurrent_writes: 8
puppetdb::server::automatic_dlo_cleanup: true
```

PostgreSQL

Optionen für puppetlabs/postgresql:

Anzahl der Verbindungen erhöhen:

```
postgresql::server::config_entry{'max_connections':
  value => 400,
}
```

Laufenden PostgreSQL Server analysieren und eine optimierte Konfiguration generieren:

```
pgtune -i /var/lib/pgsql/10/data/postgresql.conf -o postgresql.conf
```

Für PostgreSQL empfiehlt sich deren eigenes Yum Repository. Die Upstream Pakete erlauben es beliebige Versionen parallel zu installieren. Außerdem bekommt das Yum Repository am schnellsten Sicherheitsupdates. PostgreSQL hat in jeder neuen Version viele Geschwindigkeitsoptimierungen. Es sollte immer geprüft werden mit welcher neusten PostgreSQL Version PuppetDB funktioniert und diese dann genutzt werden. PuppetDB 6 und 7 in Puppet Enterprise nutzen PostgreSQL 11. Nutzer von PuppetDB 7 haben auch berichtet, dass sie erfolgreich PostgreSQL 13 einsetzen.

Neben dem autovacuum, welches als gelöscht markierte Tuples entfernt, bietet sich auch die Nutzung von pg_repack an. Für Puppet Enterprise wird dies mit dem pe_databases Modul konfiguriert. `pg_repack` kann Tabellen und Indexe aufräumen sowie die PostgreSQL Daten im Dateisystem neu ordnen. Somit benötigen Queries weniger IO.

Hier die 3 Timer + Services aus einer PE Umgebung extrahiert.

```
# /etc/systemd/system/pe_databases-catalogs.service
[Unit]
Description=Service to repack PE database tables
Wants=pe_databases-catalogs.timer

[Service]
User=pe-postgres
Group=pe-postgres
Type=oneshot
ExecStart=/opt/puppetlabs/server/apps/postgresql/11/bin/pg_repack -d pe-puppetdb --jobs 3 -t catalogs -t catalog

[Install]
WantedBy=multi-user.target

# /etc/systemd/system/pe_databases-catalogs.timer
[Unit]
```

```
Description=Timer to trigger repacking PE database tables

[Timer]
OnCalendar=Sun,Thu *-*-* 04:30:00
Persistent=true

[Install]
WantedBy=timers.target

# /etc/systemd/system/pe_databases-facts.service
[Unit]
Description=Service to repack PE database tables
Wants=pe_databases-facts.timer

[Service]
User=pe-postgres
Group=pe-postgres
Type=oneshot
ExecStart=/opt/puppetlabs/server/apps/postgresql/11/bin/pg_repack -d pe-puppetdb --jobs 3 -t factsets -t fact_pa

[Install]
WantedBy=multi-user.target

# /etc/systemd/system/pe_databases-facts.timer
[Unit]
Description=Timer to trigger repacking PE database tables

[Timer]
OnCalendar=Tue,Sat *-*-* 04:30:00
Persistent=true

[Install]
WantedBy=timers.target

# /etc/systemd/system/pe_databases-other.service
[Unit]
Description=Service to repack PE database tables
Wants=pe_databases-other.timer

[Service]
User=pe-postgres
Group=pe-postgres
Type=oneshot
ExecStart=/opt/puppetlabs/server/apps/postgresql/11/bin/pg_repack -d pe-puppetdb --jobs 3 -t producers -t resour

[Install]
WantedBy=multi-user.target

# /etc/systemd/system/pe_databases-other.timer
[Unit]
Description=Timer to trigger repacking PE database tables

[Timer]
OnCalendar=*-*-20 05:30:00
Persistent=true

[Install]
WantedBy=timers.target
```

Foreman

Hiera Optionen für theforeman/foreman:

```
---
# default is 5
foreman::db_pool: 20
foreman::keepalive: true
foreman::max_keepalive_requests: 1000
foreman::keepalive_timeout: 180
```

Foreman läuft via Passenger. Dieser wird mit puppetlabs/apache aufgesetzt. Multithreading hochdrehen:

```
---
apache::mod::passenger::passenger_max_pool_size: %{facts.processors.count}
apache::mod::passenger::passenger_min_instances: %{facts.processors.count}
```

Foreman kann memcached als Cache nutzen:

Via Puppet installieren:

```
# https://forge.puppet.com/saz/memcached
include memcached
include foreman::plugin::memcache
```

Via Hiera einstellen:

```
---
# 50GB of cache
memcached::max_memory: 51200
foreman::plugin::memcache::hosts:
  - 127.0.0.1
```

Puppetserver

Puppetlabs bietet im Modul puppetlabs/puppet_metrics_dashboard einen Monitoring Stack für Puppetserver. Dieser bietet JVM Metriken via JMX und Puppetserver Metriken via Graphite.

- Eine JVM Instanz pro (logischen) CPU Kern funktioniert gut
  - Auf modernen CPUs auch gern 2 Instanzen pro Kern
- ~800-1500 Resources pro katalog vertragen 2GB Heap pro JVM Instanz (server_jvm_max_heap_size)
- Puppetserver kann zum starten weniger Ram allokieren, dies verkürzt die Bootstrap Zeit
- Puppetserver hat einen Class Cache, welchen man aktivieren kann
- Jede JVM Instanz hat einen Cache, dieser wird besser je länger er läuft
  - Serverseitige Funktionen mit memory Leaks führen zu Problemen
  - Jeder Restart tötet den Cache
  - server_max_requests_per_instance auf unendlich setzen oder mindestens 500.000
    - Die Option startet die JVM Instanz neu nachdem die Anzahl der Requests bearbeitet wurde, dies resettet den Cache
    - Man sollte die Option also nur setzen wenn man irgendwo im Code Memory leaks einer Puppet Funktion vermutet
- Mehrere Puppetserver kann man via DNS Round Robin skalieren
- HAProxy/Nginx als Proxy davor funktioniert besser als Round Robin
  - Während eines Agent runs sollte der Agent immer mit dem gleichen Backend sprechen, der Loadbalancer/Proxy muss dies beachten
- nginx wurde gern zum terminieren von TLS Verbindungen direkt auf Puppetservern genutzt
  - Puppetserver kann dies seit 6 selbst gut
  - Legacy Vortrag: bastelfreak.de/scalingpuppetserver (Repo)
- `file` Reports sind unnötig und benötigen oft IO/s und Inodes (`grep ^reports /etc/puppetlabs/puppet/puppet.conf`)
- Jede `file` Ressource mit einer Puppetserver URL (`puppet:///`) kann zu einem eigenen HTTP Request vom Agent zum Puppetserver führen
  - `source => puppet:///modules/...` durch `content => file(${module_name}/)` ersetzen
  - Damit liest der Puppetserver die Datei ein während der Katalog kompiliert wird und bindet diese dort ein
  - Nachteil: Katalog wird größer; Vorteil: Weniger HTTP Verbindungen. Gerade bei Textdatein lohnt sich dies
  - Große Binärdatein sollten damit nicht verteilt werden. Diese sollten immer anders übertragen werden (z.b. sauber paketiert)

Hiera Optionen für theforeman/puppet:

```
$cpu_count_twice = $facts['processors']['count'] * 2
$cpu_count = $facts['processors']['count'] * 1
class{'puppet':
  server_jvm_min_heap_size               => "${cpu_count}G",
  server_jvm_max_heap_size               => "${cpu_count_twice}G",
  server_max_requests_per_instance       => 500000,
  server_max_queued_requests             => $cpu_count,
  server_environment_class_cache_enabled => true,
  server_jvm_extra_args                  => ['-XX:ReservedCodeCacheSize=2G'],
}
```

Upgrades und kompatible Versionen

Folgende Puppet Komponenten sind untereinander Kompatibel

Puppet Agent	Puppetserver	PuppetDB
6.x	6.x	6.x
5.x	6.x	6.x
4.x	6.x	6.x
7.0.0	6.14.1	6.13.1
7.0.0	6.14.1	7.0.0
6.19.1	6.14.1	7.0.0

Puppet 6 zu 7 Upgrade

- Puppet Agent 7 funktioniert mit Puppetserver 6 / PuppetDB 6
- Puppet Agent 7 funktioniert mit Puppetserver 6 / PuppetDB 7

Es empfiehlt sich folgender Upgrade Pfad:

- Puppet 7 zur CI Pipeline hinzufügen und sicherstellen, dass die Module mit Puppet 7 funktionieren
- Die einzelnen Agents auf die neusten Puppet 6 Version updaten
  - Wenn dies erfolgreich war, die Agents auf Version 7 updaten
- PuppetDB von 6 auf 7 Updaten
  - Während des Updates erfolgen Datenbank Migrationen. Je nach Anzahl der Reports kann dies mehrere Stunden dauern
  - Während der Migration ist PuppetDB nicht nutzbar
  - Man kann vor der Migration die TTL für Reports heruntersetzen und diese damit löschen. Dies reduziert die Downtime
  - Puppetserver können danach geupdated werden

Puppet 7 Bugs

Liste an bekannten Bugs, welche große Auswirkungen haben könnten bei einem Upgrade (Stand 2020-11-24):

- https://tickets.puppetlabs.com/browse/PUP-10793 - Fact is undef with Puppet 6 but empty string with Puppet 7
- https://tickets.puppetlabs.com/browse/FACT-2880 - facter 4 differing output from facter 3 for service_provider fact
- https://tickets.puppetlabs.com/browse/PUP-10790 - user provider with uid/gid as Integer raises warning

Seit Q2 2021 sind keine offenen Bugs bekannt die ein Upgrade von Puppet 6 zu Puppet 7 behindern

Puppet Tipps und Tricks

crl_refresh_interval aktivieren

Auf sehr vielen System wird das Puppet Agent Zertifikat für einen lokalen Webserver zweckentfremdet. Oft ebenfalls mit Klientzertifikatsvalidierung. Dies funktioniert nur sinnvoll mit einer aktuellen CRL (Certificate Revocation List - Liste der von der Puppetserver CA zurückgezogenen Zertifikaten). Beim initialen Puppet Lauf wird die CRL vom Puppetserver heruntergeladen, danach aber nie aktualisiert. Viele Leute nutzen dafür einen systemd timer um die Datei periodisch herunterzuladen. Puppet kann dies seit Version 6.5.0 selbst periodisch aktualisieren.

ReservedCodeCacheSize erhöhen

Es gibt viele Mythen über die korrekten Optionen für die JVM. Die Kurzfassung: `-XX:ReservedCodeCacheSize=2G` sollte immer der Puppetserver JVM mitgegeben werden. Details dazu gibt es unter PUP-10225 sowie SERVER-2771. Dies behebt 99% der Performanceprobleme die Leute nach dem Upgrade von Puppetserver 5 haben. Langfristig soll die Option standardmäßig im Puppetserver gesetzt sein. Puppet Inc ist leider etwas träge was das updaten der Doku/Pakete angeht. Nähere Informationen zu der Option gibt es in der offiziellen Java Dokumentation. Außerdem hat Baeldung dazu noch einen Artikel Der Blog ist eine sehr gute Anlaufstelle für Artikel über Java, JVM und Spring.

Der Default Wert beträgt, je nach Java Version, 48MB. Dies ist für Java Applikationen oftmals vollkommen ausreichend. Für Jruby wird allerdings wesentlich mehr benötigt.

Code Cache Nutzung protokollieren

Um die Ramnutzung etwas transparenter zu gestalten kann man die JVM Option `-XX:+PrintCodeCache` setzen. Damit loggt diese die Nutzung/Auslastung des Code Caches.

Garbage Collection Logging aktivieren

Puppetserver und PuppetDB werden innerhalb der JVM ausgeführt. Diese führt regelmäßig eine Garbage Collection durch. Hierbei werden nicht mehr benötigte Objekte aus dem Arbeitspeicher entfernt. Je mehr Aktivit innerhalb der JVM existiert und je näher der genutzte Ram an der Heap Grenze ist, desto agressiver und öfter arbeitet der Garbage Collector. In der Regel läuft dieser periodisch zum Puppetserver/PuppetDB. Sollte er Ram nicht freigeben können kann es vorkommen, dass er die Applikation kurzzeitiig pausiert. All diese Informationen kann die JVM in eine Logdatei schreiben. Dies ist nützlich um die Arbeitspeichernutzung zu prüfen und ggf den Heap zu vergrößern/verringern. Aktiviert wird es mit folgender Option: `-Xlog:gc*:file=/gc.log::filecount=16,filesize=20m` Dies aktiviert das Logging in die Datei gc.log. Sobald die Datei 20MB Größe erreicht wird die rotiert. Es werden 16 Datein vorgehalten. Für Puppetserver empfiehlt sich der Pfad `/var/log/puppetlabs/puppetserver/puppetserver_gc.log`, für PuppetDB `/var/log/puppetlabs/puppetdb/puppetdb_gc.log`.

Es gibt verschiedene Tools, welche die Logs auswerten können. Eine simple Version ist der Upload auf https://gceasy.io/.

JVM Heap Abbilder schreiben

Sobald die JVM versucht mehr Arbeitsspeicher zu nutzen als verfügbar/Heap erlaubt, läuft die JVM in einen Out of Memory Fehler und wird beendet. Sofern dies passiert kann die JVM ein Abbild des Heaps auf das Dateisystem schreiben. Out of Memory Fehler werden provoziert wenn Heap massiv zu klein konfiguriert ist oder ein Puppet Modul irgendeine Funktion bereitstellt die Memory Leaks verursacht. Wenn `max_requests_per_instance` im Puppetserver gesetzt ist werden Memory Leaks eventuell verschleiert weil die einzelnen JVM Instanzen regelmäßig neugestartet werden. Die erzeugten Abbilder können mit verschiedenen Tools analysiert und visualisiert werden.

dns_alt_names beim initialen Puppet run setzen

Damit mindestens Webbrowser TLS Zertifikate als valide empfinden muss ein FQDN als Subject Alternative Name im Zertifikat gesetzt werden. Chrome erfordert dies seit Version 58. Puppet setzt bis Puppetserver 6.15.3 ausschließlich den alten Common Name. Als Workaround kann man den initialen Puppet run starten mit: `puppet agent -t --dns_alt_names=$(hostname -f)`. Sofern ein Server mehrere FQDNs hat die auf ihn zeigen kann man der Option auch eine Kommaseparierte Liste mitgeben. Seit Puppetserver 6.15.3 bzw. 7.1.0 setzt die Puppetserver CA den Common Name ebenfalls als Subject Alternative Name.

Java 11 und nicht Java 8 nutzen

Je nach Version vom Puppetserver und PuppetDB werden diese mit Java 8 ausgeführt. Diese Version ist sehr sehr alte. Puppetserver 6 und PuppetDB 6 unterstützen diese auch Java 11. Dies sollte unbedingt genutzt werden. Es empfiehlt sich regelmäßig die Release Notes zu prüfen. Sobald eine neuere Java Version unterstützt wird sollte auf diese gewechselt werden. Beide Dienste funktionieren sehr gut mit OpenJDK/AdoptJDK. Es wird kein Java von Oracle benötigt.

`$trusted['certname']` und nicht `$facts['networking']['fqdn']` nutzen

Oftmals wird innerhalb der Puppet DSL der FQDN vom Agent benötigt. Hierzu gibt es viele Optionen. Veraltet sind:

- `$fqdn`
- `$::fqdn`
- `$facts['fqdn']`

Diese drei optionen sind veraltet. Alternativ soll `$facts['networking']['fqdn']` genutzt werden. Dies liefert den FQDN zurück. Dieser kann aber auf dem Quellsystem sehr einfach geändert und manipuliert werden. In den meisten Fällen ist `$trusted['certname']` die bessere Option. Dies ist der Common Name aus dem Zertifikat des Agents. In der Regel entspricht dies dem FQDN des Agents *während des ersten* Puppet runs.

trusted_external_command nutzen um Inventarsysteme einzubinden

Oft hat man Innerhalb einer Firma ein zentrales Inventarsystem. Leider meistens sogar mehrere. Oft enthalten diese Informationen die man auch innerhalb einer Puppet Umgebung nutzen möchte. Oft werden z.b. die Teamzugehörigkeit oder offene Problemtickets in der motd eines Servers verlinkt. Mit der trusted_external_command Option kann man auf dem Puppetserver ein Script hinterlegen. Diesem wird als erster Parameter der Common Name des anfragenden Agents übergeben. Das Script kann sich dann zu externen Systemen verbinden und dort die Informationen ermitteln. Puppet erwartet als Antwort einen JSON Hash. Dieser wird in den Hash `$trusted['externa']` eingefügt und ist innerhalb des Puppet Codes verfügbar. Puppet Code kann `$trusted` nicht überschreiben, nur lesen.

Ein gutes Beispiel hierfür ist das servicenow Modul von Puppet Inc. ServiceNow ist ein cloudbasiertes Inventarsystem. Das Script für den `trusted_external_command` gibt es hier.

Achtung: Die Lizenz erlaubt nur eine Nutzung mit Puppet Enterprise.

server_max_open_files erhöhen

Jeder Prozess darf unter Linux eine bestimmte Anzahl offener Dateideskriptoren haben. Dies sind zum großen Teil geöffnete Datein und TCP Sockets. Das Limit lässt sich mit `ulimit -Sn` auslesen und beträgt auf den meisten Systemen 1024 für normale Benutzer (Soft Limit). Diese können es manuell auf den Wert von `ulimit -Hn` erhöhen (Hard Limit). Ein Puppetserver mit vielen JVM Instanzen, der eventuell mit mehreren externen Diensten (Report Prozessor, trusted_external_command...) kommuniziert, benötigt oftmals mehr offene Dateideskriptoren. Die aktuell offenne Dateideskriptoren lassen sich wie folgt ermitteln: `find /proc/$(pgrep -f puppetserver)/fd | wc -l` Mit dem puppetserver Puppet Modul lässt sich das limit erhöhen:

```
puppet::server_max_open_files: 16384
```

Es gibt noch ein globales Limit welches der Linux Kernel vorgibt. Dies erhält man mit `sysctl fs.file-max`. Es kann ebenfalls über Puppet erhöht werden:

```
sysctl { 'fs.file-max':
  ensure => 'present',
  value => '9923372036854775807',
  target => '/etc/sysctl.d/99-puppet.conf',
```

Sofern dies nötig ist deuted es immer auf ein Problem auf dem Server hin! Weitere Informationen findet man in der Linux Kernel Dokumentation.

digest_algorithm von md5 auf sha256 ändern

digest_algorithm bestimmt das Hashverfahren mit dem Datein vergleichen werden. Puppetserver erzeugt Prüfsummen für Datein auf dem Puppetserver. Der Agent führt das gleiche lokal durch und geänderte Datein zu erkennen. Bis einschließlich Puppet 6 wird hier das veraltete md5 genutzt. Seit Puppet 7 ist der Standardwert sha256. Ältere Puppet 4/5/6 Umgebungen sollten ebenfalls auf sha256 geändert werden. Sofern benötigt, kann man dies auch auf sha512 ändern. Dies ist eventuell für einige Zertifizierungen/Audits notwendig. Die Option wird in der puppet.conf gesetzt.

key_type von rsa auf ec wechseln

Standardmäßig nutzt Puppet Agent veraltete Zertifikate mit dem RSA Verfahren. Seit Puppet 6.5.0 kann man aber aber Zertifikate auf Basis von Elliptischen Kurven nutzen. Dafür muss in der puppet.conf vor dem ersten Puppet Run (Bevor ein Zertifikat vorhanden ist) die Option `key_type` auf `ec` gesetzt werden. Zertifikate auf Basis Elliptischer Kurven gelten als Zukunftssicher und benötigen weniger Speicherplatz.

graph Option aktivieren

Der Puppet agent wendet bei jedem Lauf einen Katalog an, den zuvor der Puppetserver kompiliert hat. Dieser besteht aus einem gerichteten Graphen. Alle Ressources die angewendet werden sollen werden untereinander auf implizite und explizite Abhängigkeiten geprüft und in eine Reihenfolge gebracht. Manchmal gibt es Probleme den Katalog auf einem Node anzuwenden. Dazu kann der Agent den Graph lokal speichern in verschiedenen Formaten. Die Option ist standardmäßig deaktiviert, sollte aber aktiviert sein um auch einmalig aufgetretene Fehler im nachhinein noch zu analysieren.

Directed acyclic Graph bei Wikipedia

show_diff aktivieren

Wenn Puppet eine Datei ändern, sei es mit der Concat Resource oder weil die ganze Datei vom Puppetserver kommt oder aus einem Template generiert wird, kann Puppet einen diff anzeigen, wenn show_diff=true gesetzt ist. In dem Fall wird der diff aber auch im Report gespeichert und damit zum Puppetserver zurückgeschickt. Wenn man Reports verarbeitet, z.b. mit Foreman, kann man sich hier die diffs anzeigen lassen. ist show_diff nicht gesetzt oder `false` kann man im Report nur erkennen das eine Datei geändert wurde, aber nicht wie. Es empfiehlt sich daher die Option zu aktivieren. Reports sollten dann allerdings nicht unverschlüsselt übertragen werden da diffs potentiell sensitive Informationen enthalten können.

resubmit_facts aktivieren

In einer Agent/Server Umgebung werden Facts auf dem Agent ermittelt und zum Server geschickt. Im Gegenzug bekommt der Agent den kompilierten Katalog zurück und wendet ihn an. Am Ende wird ein Report zum Server geschickt. Es ist möglich, dass der Katalog Einfluss auf Werte irgendwelcher Facts hat (z.B. die Anzahl offener Updates wenn der Katalog Updates bestimmter Pakete erzwingt). Somit sind Facts in der PuppetDB eventuell nicht aktuell nachdem ein Katalog angewand wurde. Sofern man viel mit PuppetDB Queries innerhalb der Puppet Codebasis agiert und Facts anderer Node ausliest, oder sich viele Reports mit eigenen PuppetDB Queries erzeugt ist es sinnvoll dem Agent mitzuteilen, dass er Facts nach dem anwenden des Katalogs nochmal zum Server schicken soll. Dies kann man mit der Option resubmit_facts aktivieren. Somit verdoppelt sich allerdings die Last durch Facts auf der Puppetdb.

PDF

`pandoc` ermöglicht es aus dieser markdown Datei eine pdf zu generieren. Die gängigen Linux Distributionen haben pandoc in ihren Repositories. Der simpelste Aufruf lautet wie folgt:

```
pandoc --from markdown README.md --output puppet.pdf
```

```
pandoc README.md ---output puppet.pdf "-fmarkdown-implicit_figures -o" --from=markdown -V geometry:margin=.4in -
```

Lizenz

Anmerkungen

- Dieses Dokument wurde ursprünglich als gist gepflegt: https://gist.github.com/bastelfreak/33a9d1510448e31e7d8139a80d16e44a
- Weitere Vorträge und Dokumente gibt es in meinem Git Repository
- Für Consulting Anfragen, schicken sie bitte eine Email an tim@bastelfreak.de