|  |  |
| --- | --- |
| Informationstechnologie | |
| Dokumentenart | Arbeitsunterlage |
| Titel | **Apache-Schulung: Lektion 11** |
| Nummer | 1 |
| Autor/-in | Christian Folini |
| Kontaktangaben | +41 79 220 23 76  christian.folini@post.ch |
| Ausgabestelle | IT222 |
| Geltungsbereich | Internet / WWW |
| Klassifizierung | intern |
| Version | X01.00 |
| Ausgabedatum | 19. Januar 2016 |
| Ersetzt Ausgabe vom | 1. Januar 1970 |

Apache Schulungszentrale: <https://bullnix.post.ch/foswiki/bin/view/BasicServices/ApacheSchulungen>

Quellen: -

# Capi Deployment und Masterconf

# Was machen wir?

Reverse Proxies, White Proxies und ein Teil der Apache Applikationsserver bei Post IT werden mittels dem hauseigenen *Capi Deployment* installiert. Wir lernen den Deployment Mechanismus und die eng damit verwandte *Masterconf* kennen. Capi und Masterconf arbeiten eng zusammen und müssen auch gemeinsam betrachtet werden.

Capi und Masterconf für Apache 2.4 besitzen gegenüber der Apache 2.2 Version einige neue Features und ein leicht verbessertes Verhalten. Für diese Schulung wird dieses Verhalten als Grundlage genommen und das alte Verhalten ignoriert.

# Warum tun wir das?

Ein Standard Reverse Proxy Service besitzt bei Post IT 2 Produktions- und 2 Integrations-Instanzen. Die entsprechenden Konfigurationen sollen pro Paar mit Ausnahme der Serverspezifischen IP Adressen etc. identisch sein. Default-Werte sollen von einem Standard-Template übernommen und zentral geändert werden können.   
Diese verschiedenen Anforderungen bedingen ein Deployment-Verfahren. Mehrere Deploymentverfahren existieren bei Post IT parallel. Auf den Reverse Proxies hat sich mit Capi und Masterconf ein flexibles und sehr mächtiges Werkzeug durchgesetzt, das für die PSPP Plattform von Frédéric Tschannen (Capi) und Fränzi Bühler (Masterconf) entwickelt wurde.

# Voraussetzungen

Diese Lektion funktioniert nur im Zusammenspiel mit einem funktionierenden Deployment und Masterconf Server. Dies aufzubauen sprengt aber den Rahmen der Schulung, weshalb die Übung entweder theoretisch bleiben muss, oder ihr probiert die Konfiguration auf dem Server dev.post.ch.

# Schritt 1: Anforderungen im Detail

Das Capi Deployment ist zwar einfach anzuwenden – da haben wir drauf geachtet – aber nur schwer zu verstehen. Da haben wir zwar auch drauf geachtet, aber die mehrfachen Anforderungen führen zu einer hohen Komplexität. Zunächst also die Anforderungen im Detail:

1: Die Konfigurationen müssen serverübergreifend versioniert werden können und garantiert identisch sein.

2: Die Konfigurationen müssen zentral angepasst werden können, ohne sämtliche Services anzufassen.

3: Die Konfigurationen müssen vor der Aktivierung auf ihr Funktionieren und ihre Sicherheit hin geprüft werden.

Die erste Anforderung zwingt zu einem standardisierten Tag-Replacement und einem Deployment-Verfahren auf Basis von Files, die der Versionskontrolle unterliegen. Der Aufruf des Deployment soll dabei von zentraler Stelle aus erfolgen, wo auch das serverspezifische Tag-Replacement (Hostnamen, IP Adressen etc.) geschehen soll.

Die zweite Anforderung bedeutet, dass die Konfiguration asynchron generiert werden muss. Ein zentraler Update-Prozess muss dazu führen, dass sämtliche Instanzen eine neue Konfiguration mit neuen Werten erhalten. Gleichzeitig muss sichergestellt werden, dass einzelne Services gewisse Default-Werte durch eigene Werte mittels eines Tag-Replacements ersetzen können. Diese Anforderung verlangt einen Masterconf-Vorgang, der eine Konfiguration halbautomatisch oder vollautomatisch generiert.

Die dritte Anforderung kümmert sich um die Qualitätssicherung der generierten Konfiguration. Konfigurationen zu generieren ist eine heikle Angelegenheit. Um einen reibungslosen Betrieb zu gewährleisten muss eine generierte Konfiguration deshalb auf ihre syntaktische Korrektheit sowie die Voraussetzungen des Systems (OS, abhängige Files, Interfaces, User etc.) geprüft werden. Wirklich verlässlich ist diese Prüfung nur auf dem Server selbst. Zumal eine zentrale Generierung und anschliessende Verteilung weitere Fehlermöglichkeiten mit sich bringen würde. Aus diesem Grund geschieht die Generierung der Konfiguration besser dezentral auf dem Server selbst.

Die Anforderungen bringen also zwei Tag-Replacements mit sich. Das zentrale Deployment verlangt ein Tag-Replacement auf dem zentralen Deployment Server (hnix1a/2a). Die Überprüfung der generierten Konfiguration verlangt wiederum, dass die Masterconf mit ihrem Tag-Replacement auf dem Reverse Proxy Server selbst stattfindet. Das bedeutet, dass wir nacheinander zwei Tag-Replacements durchführen.

Der aus diesen Anforderungen abgeleitete grobe Deployment-Ablauf:

* Capi führt das Tag-Replacement durch und generiert ein Deployment-Paket pro Server.
* Capi kopiert das Deployment-Paket auf den Server.
* Capi packt das Deployment-Paket aus und verteilt die Files auf dem Server und setzt die Permissions
* Capi stösst Masterconf an.
* Masterconf stellt die verschiedenen generischen und service-spezifischen Snippets zu einer Konfiguration zusammen.
* Masterconf führt das Tag-Replacement auf Basis Default-Werte und service-spezifischen Werten durch.
* Masterconf überprüft die generierte Konfiguration auf Ihre Konformität und Lauffähigkeit.
* Masterconf installiert die generierte Konfiguration auf dem System.
* Capi startet den RP Service (neu).

# Schritt 2: Masterconf Generierung der Konfiguration

Masterconf ist der einfache Teil und deshalb nehmen wir ihn uns zuerst vor. Masterconf stellt eine Servicekonfiguration aus mehreren Konfigurations-Schnippseln (Snippets) zusammen. Die Snippets stammen aus zwei Quellen: Generische Schnippsel, die auf sämtlichen Services angewendet werden sowie service-spezifische Schnippsel, die durch Capi pro Service verteilt werden.

Die generischen Schnippsel befinden sich unter   
/data/project/apache-masterconf-24/etc/generic/snippets/

Die service-spezifischen Schnippsel befinden sich unter unter   
/data/project/apache-masterconf-24/etc/service-specific/<URL>/snippets/

Die Schnippsel sind im Filenamen durch einen Prefix nummeriert und werden in der Reihenfolge der Zahlen kombiniert. Hier eine Tabelle mit den standardmässig vorhandenen Snippets:

|  |  |
| --- | --- |
| **Generische Snippets** | **Service-spezifische Snippets** |
| 050\_metadata.txt  100\_general\_settings.txt  110\_listeners.txt  120\_listeners\_ipv6.txt  200\_load\_modules.txt  210\_load\_module\_cas.txt  300\_logfile\_geoip.txt  400\_docroot\_errorhandling.txt  420\_headers\_deflate.txt  440\_locations\_directories.txt  460\_rewrites.txt  500\_modsecurity\_basic.txt  510\_modsecurity\_config.txt |  |
|  | 520\_modsecurity\_service.txt |
| 530\_modsecurity\_corerules.txt |  |
|  | 540\_modsecurity\_service\_after\_corerules.txt |
| 550\_reqtimeout.txt  600\_ssl\_settings.txt  701\_virtual\_host\_http\_open.txt  710\_virtual\_host\_http\_general.txt  720\_virtual\_host\_http\_log.txt  730\_virtual\_host\_https\_header.txt  740\_virtual\_host\_http\_rewrite.txt |  |
|  | 760\_virtual\_host\_http\_service1.txt |
|  | 761\_virtual\_host\_http\_service2.txt |
|  | etc. bis 789 wenn nötig |
| 796\_virtual\_host\_http\_redirssl.txt  798\_virtual\_host\_http\_close.txt  801\_virtual\_host\_https\_open.txt  810\_virtual\_host\_https\_general.txt  820\_virtual\_host\_https\_log.txt  830\_virtual\_host\_https\_header.txt  850\_virtual\_host\_https\_cas.txt |  |
|  | 860\_virtual\_host\_https\_service1.txt |
|  | 861\_virtual\_host\_https\_service2.txt |
|  | etc. bis 889 wenn nötig |
| 898\_virtual\_host\_https\_close.txt |  |

Zusätzliche service-spezifische Snippets können beliebig hinzugefügt werden. Masterconf berücksichtigt sämtliche Snippets an den bezeichneten Standorten, sofern sie dem beschriebenen Namensmuster entsprechen. Dies war unter der Apache 2.2-Masterconf nicht der Fall und bedeutet eine wesentliche Vereinfachung.

Generische Snippets können einfach ersetzt werden, indem ein service-spezifisches Snippet mit demselben Namen ins Deployment eingefügt wird.

**Schritt 3: Masterconf Tag Replacement**

Masterconf arbeitet nur auf der Apache Konfiguration. Die übrigen Dateien eines Deployments bleiben unberührt. Auch das Tag-Replacement der Masterconf arbeitet nur auf der Apache Konfiguration. In diesem Vorgang werden Tags aus einem service-spezifischen File namens properties.rb gelesen und die entsprechenden Tag-Namen in der generierten Apache Konfiguration ersetzt. Kann ein Tag-Name im service-spezifischen properties.rb nicht aufgelöst werden, wird das generische properties.rb herangezogen. Kann es auch mit diesen Tags nicht aufgelöst werden, bricht die Masterconf mit einem Fehler ab. Hier die beiden Dateien nebeneinander:

/data/project/apache-masterconf-24/etc/service-specific/<service-url>/properties/properties.rb

/data/project/apache-masterconf-24/etc/generic/properties/properties.rb

Hier ein Beispiel für ein service-spezifisches Properties-File:

#Service-specific settings

#override generic definitions

@servername = "owa.swisspost.com"

@url = "owa.swisspost.com"

@address = "194.41.188.5"

@user = "www20075"

###

### Achtung, die properties werden uebersteuert vom snippet 100

###

#@timeout = "3600"

#@keepalivetimeout = "15"

#@keepaliverequests = "200"

#

#@maxclients = "512"

#@minspareservers = "10"

#@maxspareservers = "20"

#@startservers = "10"

#

#

##used by Snippet 100

#@limitrequestbody = "10485760"

#@limitrequestline = "2048"

@logicalhttp = "194.41.188.5:80"

@logicalhttps = "194.41.188.5:443"

#used by Snippet 300

@cookiedomain = ".swisspost.com"

#used by Snippet 400

@proxyerroroverride = "Off"

@sslcarevocationcheck = "none"

#used by Snippet 500

@secrequestbodylimit = "10485760"

#used by Snippet 510

#use regexp: ittrksessid|appcookie1|appcookie2

@httponlywhitelist = "ittrksessid|MOD\_AUTH\_CAS\_S|CASPRIVACY|CASTGC|JSESSIONID|UserContext|cadata|sessionid|owacsd|tzid|msExchEcpCanary|mkt"

@securewhitelist = "ittrksessid|CASPRIVACY|JSESSIONID|UserContext|MOD\_AUTH\_CAS\_S|cadata|sessionid|Cookie|msExchEcpCanary|mkt"

@pathwhitelist = "MOD\_AUTH\_CAS\_S|ittrksessid|UserContext|cadata|sessionid|owacsd|tzid|OutlookSession|msExchEcpCanary|mkt"

# ModSec Anomaly Scores (used by Snippet 510)

@inboundscore = "10"

@outboundscore = "10"

#used by Snippet 710 and 720

@proxytimeouthttps = "1200"

# used by Snippet 850

@casloginurl = "login.post.ch"

@casloginport = "443"

@authservice = "authservice-radius-eaas"

@casvalidateurl = "cas.post.ch"

@casvalidateport = "8001"

# Special extension to properties file, performed by cap deployment itself in task mc:package"

@hostname = "h058wv"

@documentation = "https://bullnix.post.ch/foswiki/bin/view/ITServices/OwaPSPP"

@deploymentpath = "https://repository.pnet.ch/svn/repos-itservices/owa/trunk/rp"

@deploymentrevision = "24119 (there were changes not yet committed to deployment repository)"

@deploymentdate = "2015-12-11 16:22:58"

@deploymentuser = "flueckigerl"

@documentationshort = "OwaPSPP"

# Special extensions to properties file, added by cap deployment itself in task mc:package

# Date updated by apache masterconfig on every run

@dynamicmasterconfgenerationdate = "2016-01-22 00:11:57"

# Physical IP linked to logical IP (needed for the restricted access to /post-sys-status)

@dynamicmasterconfphysicalip = "194.41.188.15"

Das Ziel bei der Masterconf ist eigentlich sämtliche Werte in der Apache Konfiguration zu taggen. Im Detail gibt es einzelne Teile in den Snippets, wo dies nicht der Fall ist. Wenn wir da anstossen wird bisweilen ein neues Tag im generischen Properties.rb erfasst.

# Schritt 4: Capi Deployment

Wir haben gesehen, wie die Dateien auf dem Server zu einer Apache Konfiguration kombiniert und wie die Tags mit den realen Konfigurationswerten gefüllt werden. Wie aber gelangen Sie auf den Server? Hier kommt Capi zum Zug. Capi ist eine Abkürzung für Capistrano. Dabei handelt es sich um ein Deployment-Framework in Ruby, welches das Tag-Replacement und das Verteilen von Dateien beherrscht. Capi ist eine Sammlung von Skripts, die für die Umgebung bei der Post zurecht gelegt wurden.

Apache Server, seien es Reverse Proxies, White Proxies oder Application Server, existieren meist in redundanten Paaren. Die Konfiguration der beiden Server soll identisch sein, mit Ausnahme von IP Adressen, je nachdem Backendservern etc. Die verschiedenen Konfigurations-Snippets, welche wir im letzten Kapitel angesehen haben, besitzen deshalb an den entscheidenden Stellen Platzhalter, sogenannte Tags, welche durch Capi ersetzt werden. Wir haben also gesehen, dass Masterconf gewisse Tags ersetzt, ein weiterer Teil der Tags wird durch Capi ersetzt.

Ein früher Design-Entscheid beim Aufbau der PSPP Plattform legte fest, dass sowohl die Capi-Tags wie auch die Masterconf-Tags der Form @tagname@ entsprechen. Im Rückblick betrachtet war dies ein Fehler, denn die Tags in den Snippet Files können nun nicht mehr unterschieden werden.

Capi ersetzt die Tags mit Hilfe von Props-Files. Diese stehen in einem eigenen Unterordner im Deploayment Tree und werden nach Umgebung, „prod“ oder „int“ etc. unterschieden. Beim Deployment wird also pro Server ein Tag Replacement durchgeführt.

Anders als die Masterconf arbeitet Capi auf sämtlichen Files eines Deployments und nicht nur auf den Apache Snippets, sondern auch auf dem Start-Stop-Skript etc.

In zahlreichen Fällen ist es erwünscht, Tags in der Masterconf Properties Datei durch Capi ersetzen zu lassen. Dabei kommt eine Eigenart von Capi ins Spiel: Die Tagnamen sind durch Punkte hierarchisch strukturiert. Ein Beispiel:

rp.h058wv.logical.address=194.41.188.117

Capi schneidet beim Durchgang nun die ersten beiden Teile ab. Denn Capi führt das Tag-Replacement für den „rp“ und der Serverspezifischen Teil, hier „h058wv“ durch. Das bedeutet, dass ein Props-File Tag namens rp.h058wv.logical.address im Masterconf Properties-File als "@logical.address@ referenziert werden muss:

@logicalhttp = "@logical.address@:80"

Links also der Tagname, wie er im Konfigurations-Snippet auftaucht. Rechts der Wert, der auf das Props-File verweist.

Dieser Zusammenhang und die Veränderung der Tagnamen ist sehr anstrengend. Es hängt am Capistrano Framework und kann von uns nicht beeinflusst werden.

Nach dem Durchlauf durch das Capi-Tag Replacement erhalten wir damit aber ein auf den Server angepasstes Properties-File, das für die Masterconf Generierung bereit ist. Hier die entsprechende Properties-Zeile auf dem Reverse Proxy:

@logicalhttp = "194.41.188.117:80"

Metadaten des Deployments werden zum Teil in eigenen Arbeitsschritten von Capistrano generiert. Diese Daten werden dann an das Properties-File angehängt, von wo sie später durch die Masterconf aufgenommen und vor allem zu Beginn der Apache Konfiguration im 050er Snippet eingefüllt werden. Wir haben das oben im Masterconf Properties-File Beispiel gesehen.

Neben dem Tag-Replacement kümmert sich Capi auch um die Verteilung der Dateien. Mit dem Tag-Replacement wird ein Archiv-Paket für jeden einzelnen Ziel-Server angelegt. Capi arbeitet dabei nicht auf sämtlichen Files des Deployment Trees, sondern nur auf denjenigen Files, die in einer Datei namens comp.list beschrieben sind. Es ist ein typischer Fehler besteht darin, ein neues File zu erstellen, den Eintrag in der Comp-List zu vergessen und sich dann zu wundern, weshalb die Datei auf dem Zielsystem nicht ankommt. Hier ein Auszug aus der besagten Comp-List:

Wie kopiert Capi nun die verschiedenen Files? Capi stützt sich auf die sogenannte Comp-Liste; ein File im Deployment Tree:

$> cat rp/comp.list

# f = copy file d = create directory

# T = tag destination path t = tag file content for destinantion

# p = tag source path F = touch file (generated through special task)

# x = do not remove on clean

### system

fFT root:root 0644 /etc/permissions.d/apache\_@service.url@

fTt root:root 0744 /etc/init.d/apache\_@service.url@

fTt root:root 0644 /etc/logical\_interfaces.d/@service.url@.nic

### config

dT root:root 0755 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/

dT root:root 0755 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/520\_modsecurity\_service.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/540\_modsecurity\_service\_after\_corerules.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/701\_virtual\_host\_http\_open.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/710\_virtual\_host\_http\_general.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/720\_virtual\_host\_http\_log.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/730\_virtual\_host\_https\_header.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/740\_virtual\_host\_http\_rewrite.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/760\_virtual\_host\_http\_service1.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/761\_virtual\_host\_http\_standorte.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/796\_virtual\_host\_http\_redirssl.txt

fTt root:root 0644 /data/project/apache-masterconf-24/etc/service-specific/@service.url@/snippets/798\_virtual\_host\_http\_close.txt

…

Neben den Permissions und dem Ownership kann man das Capi Verhalten mit den Flags in der ersten Kolonne beeinflussen. Damit lassen sich Files vom Tag-Replacement ausnehmen; das ist etwa für Binary-Files nötig, da sie sonst durch capi zerrissen werden.

Capi erstellt also mit Hilfe des Inhaltsverzeichnisses in der Comp-Liste ein Datei-Archiv-Paket und verschiebt es auf den Server. Dann packt es diese Dateien aus und legt sie dann an den richtigen Ort. Spielentscheidend sind hierbei die Permissions und das Ownership der Dateien. Sie werden in der eben beschriebenen Comp-List Datei definiert. Dies ist auch der Grund, weshalb Capi den Gebrauch der Comp-List zwingend vorschreibt, denn Capi muss die Permissions jeder einzelnen Datei auf dem Ziel-System kennen, um diesen Schritt durchführen zu können. Damit ist der Kernauftrag von Capi abgeschlossen. Es bleibt der über Capi gesteuerte Aufruf der Masterconf, der Neustart des Servers etc.

Hier ein fertiger Capi-Call:

$> cap int rp:deploy rp:master\_config\_restart

oder

$> cap int HOSTS=h054r9 rp:deploy rp:master\_config\_restart

Wir sehen, dass wir hinter der Definition der Umgebung und der Zielserver verschiedene Tasks auflisten können, die dann der Reihe nach abgearbeitet werden können. Es stehen eine Vielzahl von Tasks zur Verfügung. Hier die volle Liste (die aus meiner Sicht gebräuchlichen sind **fett** markiert; die Liste kann mittels cap -T selbst angezeigt werden.):

cap adm:add\_root\_privs # add root privs

cap adm:remove\_root\_privs # remove root privs

cap adm:show\_root\_privs # show root privs

**cap apache:configtest # Check the syntax of the apache config**

**cap apache:fullstatus # Show the full status of the apache service**

cap apache:graceful # Gracefully restart the apache service

cap apache:graceful\_stop # Gracefully stop the apache service

cap apache:restart # Restart the apache service

**cap apache:start # Start the apache service**

cap apache:status # Check the status of the apache service

**cap apache:stop # Stop the apache service**

cap cas:deploy\_appserver # install appserver

cap cas:integrity\_check # checks the integrity of the deloyment to en...

cap cas:pre\_deploy # prepare target server for deployment create...

cap cert:chain\_check # chain\_check

cap cert:expiration\_check # expiration\_check

cap int # Stage integration environment

cap invoke # Invoke a single command on the remote servers.

cap local\_clean # cleans ALL backups, packages and temporary ...

cap mc:backup # backup previous installed deployment create...

cap mc:clean # remove all previously deployed files exclud...

**cap mc:deploy # Full masterconf deployment without building...**

cap mc:diff # run a diff between the remote installation ...

cap mc:diff\_full # run a full diff between the remote installa...

cap mc:install # Install the package on target server unpack...

cap mc:integrity\_check # checks the integrity of the deloyment to en...

cap mc:package # Build a tagged package for each server By d...

cap mc:pre\_deploy # prepare target server for deployment create...

cap mc:push # push package to target servers upload previ...

cap mc:set\_file\_permissions # set the file permissions set file owner, gr...

cap mc:show\_vars # show vars important variables for debugging...

cap nic:check # check if all logical interfaces are deployed

cap nic:down # stops and removes logical interface

**cap nic:start # start logical interfaces**

cap nic:status # status logical interface

cap nic:stop # stop logical interfaces

cap prod # Stage production environment Engehalde

cap readme # HELP

cap rp:backup # backup previous installed deployment create...

**cap rp:clean # remove all service-specific apache24 and ma...**

**cap rp:create\_user # create apache user with uid as defined in p...**

**cap rp:deploy # Full deployment without building the new ap...**

**cap rp:diff # run a diff between the remote installation ...**

**cap rp:diff\_full # run a full diff between the remote installa...**

**cap rp:genpp # deploy gen\_pp file : asks for SSL key passp...**

cap rp:get\_free\_uid # Determine lowest available UID

cap rp:install # Install the package on target server unpack...

cap rp:integrity\_check # checks the integrity of the deployment to e...

cap rp:list\_targets # list targets

**cap rp:master\_config # build apache config from master config star...**

cap rp:master\_config\_copy # build apache config from master config star...

cap rp:master\_config\_graceful # build apache config form master config and ...

**cap rp:master\_config\_restart # build apache config form master config and ...**

cap rp:monitoring\_off # deactivate monitoring

cap rp:monitoring\_on # activate monitoring

cap rp:monitoring\_status # monitoring status

cap rp:monitoring\_status\_all # monitoring status all services

cap rp:package # Build a tagged package for each server By d...

cap rp:path\_check # checks the correct install path of apache

cap rp:post\_deploy\_check # checks if deployment standards are met

cap rp:pre\_deploy # prepare target server for deployment create...

cap rp:push # push package to target servers upload previ...

cap rp:service\_abnahme\_check # check ServiceAbnahme file for correct values

cap rp:set\_file\_permissions # set the file permissions set file owner, gr...

cap rp:show\_vars # show vars important variables for debugging...

**cap rp:superclean # remove all service-specific files - really ...**

**cap rp:undeploy # remove all previously deployed files**

cap shell # Begin an interactive Capistrano session.

cap tc:backup # backup previous installed deployment create...

cap tc:clean # remove all previously deployed files exclud...

cap tc:deploy # Full deployment without building the new ap...

cap tc:diff # run a diff between the remote installation ...

cap tc:diff\_full # run a full diff between the remote installa...

cap tc:install # Install the package on target server unpack...

cap tc:integrity\_check # checks the integrity of the deloyment to en...

cap tc:package # Build a tagged package for each server By d...

cap tc:pre\_deploy # prepare target server for deployment create...

cap tc:push # push package to target servers upload previ...

cap tc:set\_file\_permissions # set the file permissions set file owner, gr...

cap tc:show\_vars # show vars important variables for debugging...

cap testing # Stage test environment

cap util:fix\_comp\_list # fix componet list

Einige dieser Tasks sind Zusammenfassungen von mehreren Tasks. Das deutet bereits darauf hin, dass in der Praxis meist nur eine Auswahl der grossen Taskliste von Belang sind; eben die fett markierten.

**Schritt 5: Capi Setup Interface, create User, deploy Certs**

Beim erstmaligen Deployment einer Service-Instanz gibt es mehrere Tasks, welche nur ein einziges Mal durchgeführt werden müssen:

Das virtuelle Interface muss zunächst konfiguriert werden. Dies ist ein dynamischer Task in dem Sinn, dass Capi zunächst nicht weiss, unter welcher Interface-Nummer die im Props-File konfigurierte IP-Adresse geplumbt und hochgefahren werden soll. Das Suchen einer freien Nummer ist Teil des Capi-Tasks nic:start.

Der Create-User Task ist noch komplizierter, denn nicht nur muss dynamisch eine freie UID bestimmt werden, sondern die UID muss sogar auf sämtlichen Servern einer Umgebung frei sei. Der gefundene Wert muss also zurückgemeldet werden, damit er auf dem zweiten Server weiter geprüft werden kann etc. Final wird die gefundene UID dann im Props-File eingetragen. Das alles organisiert rp:create\_user.

Auch die Zertifikate werden mittels Capi verteilt. Bei Post IT verwenden wir Key-Files, welche mit einer Passphrase geschützt sind. Die Passphrase ist nicht Teil des Deployments. Vielmehr wird sie beim Deployment vom User interaktiv abgefragt und dann in ein gen-Passphrase Skript eingefügt.

Mittels SNI ist es heute möglich, mehrere SSL-/TLS-Zertifikate auf derselben IP Adresse zu deployen. Das bedeutet, dass man auch mehrere Passphrases eingeben muss. Capi lässt das mittels des rp:genpp Tasks zu. Hier der Aufruf und ein typischer Durchlauf des Tasks:

$> cap int HOSTS=h054r7 rp:genpp

CAPI deployment

\*\*\* 17:40:07 logger level set to 1

\*\* [GENPP] for h054r7dev.post.ch

We need to define if this is a service with a single certificate and a single

passphrase. Or a service with multiple certificates and thus multiple

passphrases in the gen\_pp file.

Do you need to enter more than one passphrase? Enter yes or no (default is no): yes

You will now be asked to enter the next fully qualified domain name

of this service instance plus the port number. (i.e. www-int.example.com:443)

Please do not enter any tags, but a real fqdn.

Enter fqdn of service instance with port number (leave empty when done): dev.post.ch:443

SSL passphrase for dev.post.ch:443: \*\*\*\*\*\*\*\*\*\*

You will now be asked to enter the next fully qualified domain name

of this service instance plus the port number. (i.e. www-int.example.com:443)

Please do not enter any tags, but a real fqdn.

Enter fqdn of service instance with port number (leave empty when done): www.example.com:444

SSL passphrase for www.example.com:444:

You will now be asked to enter the next fully qualified domain name

of this service instance plus the port number. (i.e. www-int.example.com:443)

Please do not enter any tags, but a real fqdn.

Enter fqdn of service instance with port number (leave empty when done):

\*\* [GENPP] saving in /opt/apache22w/bin/gen\_pp\_dev.post.ch.sh

\*\* 17:40:37 scp upload #<StringIO:0x7fb9ccfe58e8> -> /opt/apache22w/bin/gen\_pp\_dev.post.ch.sh

\*\* [SET FILE PERMISSION] for h054r7 dev.post.ch

# Schritt 6: Capi Deployment Remove

Neben dem Deployment eines Services ist es wichtig, einen Service auch spurlos entfernen zu können. Das hat in der Vergangenheit nicht ganz immer spurlos funktioniert. Mit dem Schritt auf Apache 2.4 haben wir diesen Task aber nochmals überarbeitet und sind jetzt zuversichtlich, dass er komplett funktioniert. Das Cleaning ist in verschiedene Stufen unterteilt, die jeweils immer stärker wirken, und in der höchsten Stufe auch Interfaces und Kopien von automatisch generierten Apache-Konfigurationen abräumen.

rp:undeploy # remove all previously deployed files

rp:clean # remove all service-specific apache24 and masterconf-24 files

rp:superclean # remove all service-specific files - really ...

# Schritt 7: Nächtliche Generierung unterdrücken

Masterconf wird ja nicht nur durch Capi angestossen. Vielmehr werden sämtliche Apache-Konfigurationen jede Nacht frisch generiert. Dies gibt uns die Möglichkeit, Default-Werte über die gesamte Plattform hinweg anzupassen, ohne dass die einzelnen Services einzeln angefasst werden müssen. Bisweilen ist es aber erwünscht, dass eine sich im Aufbau befindliche Konfiguration nicht in der Nacht wieder überschrieben wird.

Aus diesem Grund gibt es die Möglichkeit, das Generieren der Apache Konfiguration zu unterdrücken. Dies geschieht, indem man folgende Datei mit dem beschriebenen Inhalt anlegt (ein Beispiel):

$> cat /data/project/apache-masterconf/etc/service-specific/owaint.swisspost.com/do\_not\_generate

Wer: buehlerfr

Warum: Masterkonfig Tests

Bis wann: 15.06.2013

Diese Technik löst täglich eine OVO-Meldung aus, welche so lange versandt wird, bis die Datei wieder entfernt wird.

**Schritt 8: CAPI und Masterconf Fehlersuche**

Capi ist sehr gesprächig. Die Meldungen rauschen aber so rasch über den Bildschirm, dass man Ihnen oft nicht folgen kann. Um sinnvoll einen Fehler im Screen-Log suchen zu können, muss man die Ausgabe oft in eine Datei umleiten. Capi ist bei Fehlern auch etwas eigen, denn nicht immer gelang es, die Ruby-Fehlermeldungen abzufangen und durch sprechende Meldungen zu ersetzen. Vielmehr geschieht es immer noch, dass bei Capi-Fehlern ein Ruby-Stacktrace ausgegeben wird.

Die Masterconf ist etwas einfacher in den Griff zu bekommen. Die Masterconf schreibt auf dem Reverse Proxy ein Logfile namens /data/project/apache-masterconf-24/logs/ApacheGenerator.log.

Dieses File ist sehr detailliert und sauber strukturiert, so dass man damit bei der Fehlersuche gut zurecht kommt. Hier ein Auszug:

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheTemplateProperties: mitarbeiterportal.postauto.ch: Binding properties to snippet.

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheTemplate: mitarbeiterportal.postauto.ch: @-values in temporary Apache config file are replaced by ERB-tags.

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheTemplate: mitarbeiterportal.postauto.ch: ERB-tags in Apache config file snippet are replaced by valid apache values and snippet is now appended to resulting Apache config file /data/project/apache-masterconf-24/config/mitarbeiterportal.postauto.ch/httpd.conf\_mitarbeiterportal.postauto.ch\_20160122-000909.

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheTemplateProperties: mitarbeiterportal.postauto.ch: Binding properties to snippet.

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheTemplate: mitarbeiterportal.postauto.ch: @-values in temporary Apache config file are replaced by ERB-tags.

...

**Fri Jan 22 00:09:09 +0100 2016 INFO ApacheTemplate: mitarbeiterportal.postauto.ch: Apache config file has been generated and written: /data/project/apache-masterconf-24/config/mitarbeiterportal.postauto.ch/httpd.conf\_mitarbeiterportal.postauto.ch\_20160122-000909.**

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheDirTester: 10 entries found in /data/project/apache-masterconf-24/config/mitarbeiterportal.postauto.ch/.

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheDirTester: 10 entries found in /data/project/apache-masterconf-24/config/mitarbeiterportal.postauto.ch/. Nothing to do. OK

Fri Jan 22 00:09:09 +0100 2016 DEBUG ApacheControl: mitarbeiterportal.postauto.ch: Apache binary /opt/apache24/bin/httpd exists. OK

**Fri Jan 22 00:09:10 +0100 2016 INFO ApacheControl: mitarbeiterportal.postauto.ch: Generated Apache config /data/project/apache-masterconf-24/config/mitarbeiterportal.postauto.ch/httpd.conf\_mitarbeiterportal.postauto.ch\_20160122-000909 is valid.**

Fri Jan 22 00:09:10 +0100 2016 DEBUG ApacheCapcore: mitarbeiterportal.postauto.ch: Getting old score out of Serviceabnahme file.

Fri Jan 22 00:09:10 +0100 2016 DEBUG ApacheCapcore: mitarbeiterportal.postauto.ch: Opening Serviceabnahme file.

Fri Jan 22 00:09:10 +0100 2016 DEBUG ApacheCapcore: mitarbeiterportal.postauto.ch: Got Capcore score out of Serviceabnahme file: 7.79.

**Fri Jan 22 00:09:10 +0100 2016 INFO ApacheCapcore: mitarbeiterportal.postauto.ch: Capcore testing of Apache Config started.**

**Fri Jan 22 00:09:13 +0100 2016 INFO ApacheCapcore: mitarbeiterportal.postauto.ch: Current Capcore score is calculated:9.41.**

Fri Jan 22 00:09:13 +0100 2016 DEBUG ApacheCapcore: mitarbeiterportal.postauto.ch: New Capcore Score is: 9.41. Old and signed Capcore score was: 7.79. Capcore difference is: 20.795892169448 percent.

Fri Jan 22 00:09:13 +0100 2016 INFO ApacheCapcore: mitarbeiterportal.postauto.ch: Capcore testing of Apache Config was successful. Apache config ist Capcore-valid.

Fri Jan 22 00:09:13 +0100 2016 DEBUG ApacheControl: mitarbeiterportal.postauto.ch: Old config directory /opt/apache24/conf/old/ exists. OK.

**Fri Jan 22 00:09:13 +0100 2016 INFO ApacheControl: mitarbeiterportal.postauto.ch: Save old Apache config file /opt/apache24/conf/httpd.conf\_mitarbeiterportal.postauto.ch to /opt/apache24/conf/old/httpd.conf\_mitarbeiterportal.postauto.ch\_pre20160122\_000909.**

**Fri Jan 22 00:09:13 +0100 2016 INFO ApacheControl: mitarbeiterportal.postauto.ch: Copy generated Apache config file /data/project/apache-masterconf-24/config/mitarbeiterportal.postauto.ch/httpd.conf\_mitarbeiterportal.postauto.ch\_20160122-000909 to /opt/apache24/conf/httpd.conf\_mitarbeiterportal.postauto.ch.**

**Fri Jan 22 00:09:13 +0100 2016 INFO ApacheControl: mitarbeiterportal.postauto.ch: Apache will do the command: graceful.**

**Fri Jan 22 00:09:13 +0100 2016 INFO ApacheControl: mitarbeiterportal.postauto.ch: Apache has done the command: /etc/init.d/apache\_mitarbeiterportal.postauto.ch graceful. Checking Apache status.**

Fri Jan 22 00:09:15 +0100 2016 DEBUG ApacheControl: mitarbeiterportal.postauto.ch: httpd PID /var/run/apache/pid/httpd\_mitarbeiterportal.postauto.ch.pid does exist, Apache is running. OK.

**Fri Jan 22 00:09:15 +0100 2016 INFO ApacheControl: mitarbeiterportal.postauto.ch: Apache is still running. OK.**

**Fri Jan 22 00:09:15 +0100 2016 INFO ApacheGeneratorControl: all: Config generation for service mitarbeiterportal.postauto.ch was successful.**

# Verweise

* CAPI <https://bullnix.post.ch/foswiki/bin/view/BasicServices/CAPI>
* Masterconf <https://bullnix.post.ch/foswiki/bin/view/BasicServices/MasterKonfig>
* Post IT PSPP <https://bullnix.post.ch/foswiki/bin/view/BasicServices/PSPP>
* Post IT PSPP Projekt (mit zusätzlicher Dok) <https://bullnix.post.ch/foswiki/bin/view/BasicServices/ApacheNSPPStart>