Saltstack – Würzen Sie Ihre Oracle Umgebung

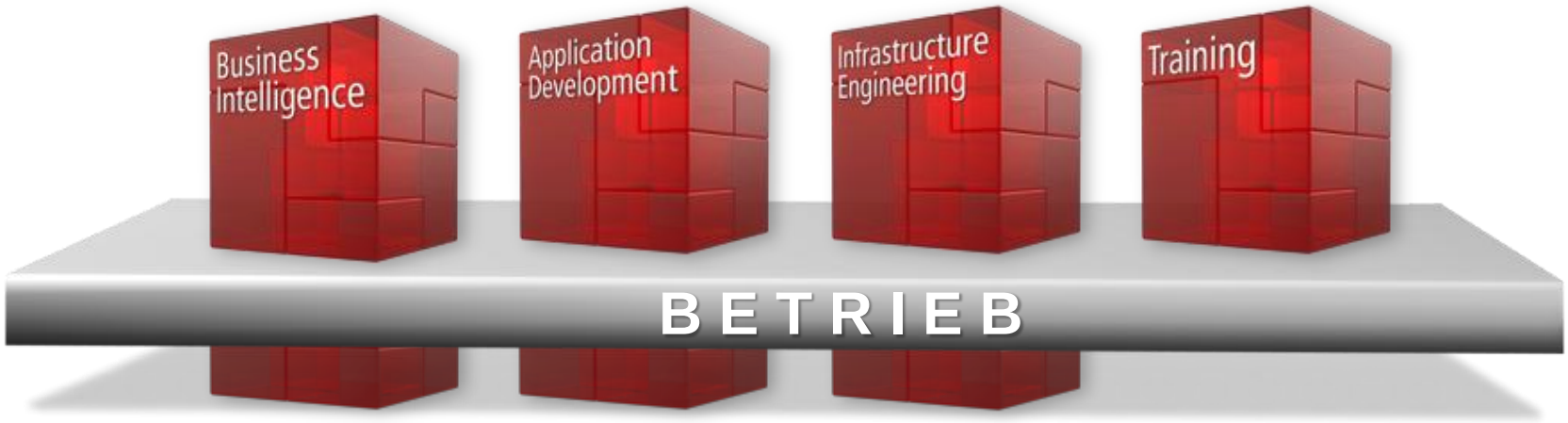
Installation von Oracle

Martin Bracher

November 2015

Unser Unternehmen.

Trivadis ist führend bei der IT-Beratung, der Systemintegration, dem Solution Engineering und der Erbringung von IT-Services mit Fokussierung auf Microsoft - und ORACLE -Technologien in der Schweiz, Deutschland, Österreich und Dänemark. Trivadis erbringt ihre Leistungen aus den strategischen Geschäftsfeldern:



Trivadis Services übernimmt den korrespondierenden Betrieb Ihrer IT Systeme.

Mit über 600 IT- und Fachexperten bei Ihnen vor Ort.

- 14 Trivadis Niederlassungen mit über 600 Mitarbeitenden.
- Über 200 Service Level Agreements.
- Mehr als 4'000 Trainingsteilnehmer.
- Forschungs- und Entwicklungsbudget: CHF 5.0 Mio.
- Finanziell unabhängig und nachhaltig profitabel.
- Erfahrung aus mehr als 1'900 Projekten pro Jahr bei über 800 Kunden.

trivadis
makes IT easier.

AGENDA

1. Introduction
   - What is Saltstack
   - Master, Minion
   - Variables
   - States
2. Setup of a RAC cluster

Saltstack

- Saltstack is
  - A configuration management system
    - Maintaining remote servers in defined states
      (e.g. specific software is installed and services are running)
  - A distributed remote execution system
    - Run commands on one or more remote hosts

  - http://salt.readthedocs.org/en/latest/contents.html
  - Opensource Software (Python)
  - Available for many platforms (also windows, but no GUI ;-) …)

  - Other similar tools:
    - puppet, chef
    - this presentation will NOT compare different tools

Saltstack

Situation @customer

- Unix-team already is using Saltstack for server configuration
- Goal for the Oracle team
  - Automatization of setup of new Oracle servers
    - Installation of Grid infrastructure and database software
    - Installation of single-servers or RAC-clusters
  - Standardization
    - All systems with 12c identically configured
  - What tools to use
    - Own shell-scripts
    - Using Saltstack framework of Unix team

Terms: Master, Minion

- Master
  - Central server
  - Has access to the Minions, can run commands
  - Scripts and global configuration is stored on the master
- Minion (client)
  - Server that is controlled by the Master

- Running commands on a Minion (e.g. host1)

```
# from the master
salt host1 cmd.run hostname

# from the minion host1
salt-call cmd.run hostname
```

Setup of minions

- Install the software
  - For Linux, avaliable as RPM packages
  - For Oracle Linux, use the YUM repository for RedHat. See: https://docs.saltstack.com/en/latest/topics/installation/rhel.html
  - Install the repo file in /etc/yum.repos.d and replace "rhel**$releasever**" with your current release, e.g. "rhel**6**"
  - Install the software

```
yum install salt-minion
```

    - On the master server, install salt-master

Setup of minions

- Configure and start the software

```
vi /etc/salt/minion
```

```
master: <your salt-master-server>
```

```
salt-minion -l debug  #initially start it interactively
chkconfig salt-minion on
service salt-minion start
```

- Accept the key on the master

```
root@master# salt-key -A
The following keys are going to be accepted:
Unaccepted Keys:
server01
Proceed? [n/Y] y
Key for minion server01 accepted.
```

Configuration of variables

- Pillars
  - Global Variables/Values, valid for the whole environment

```
~# salt-call pillar.items
```

- Grains
  - Local Variables/Values, valid on a specific minion
    - cpu-type
    - hostname
    - …

```
~# salt-call grains.setval ora_grid_version 12.1.0.2
~# salt-call grains.item ora_grid_version
local:
    ----------
    ora_grid_version:
        12.1.0.2
~# salt-call grains.items
… <shows all grains>
```

Errors with use of variables (grains/pillars)

- Caution: Lists and strings – do not confuse

```
{% set url = 'http://server/grid/' + grains['ora_grid_version'] + '/' %}
```

```
[CRITICAL] Rendering SLS 'base:oracle/copygrid' failed: Jinja error: cannot concatenate 'str' and
'list' objects
Traceback (most recent call last):
  File "/usr/lib64/python2.6/site-packages/salt/utils/templates.py", line 280, in render_jinja_tmpl
    output = template.render(**decoded_context)
...
TypeError: cannot concatenate 'str' and 'list' objects
```

```
{% set vers = grains['ora_grid_version'] %}
test-{{vers}}:
  cmd.run:
    - name: /bin/true
```

```
test-['12.1.0.2']    # ← it was a list
test-12.1.0.2        # ← it was a string
```

States

- A state is a specific setup of a server (software, configuration)
  - States are defined in **'sls' files**
    - Stored on the master below /srv/salt/
    - Example: Put Oracle stuff below /srv/salt**/oracle**
      - **create_grinf_rsp.sls**
        (creates the response file for grid-installation)
      - install_grindinfra.sls
      - …
    - Run a state-file

```
salt-call state.sls oracle/create_grinf_rsp
# -or run from master-
salt host1 state.sls oracle/create_grinf_rsp
```

States - YAML

- States are defined in ‘sls’ files (cont.)
  - SLS files are rendered by YAML (Yet Another Markup Language)
    - Use indention of 2 spaces

```
my_key:
  my_value:
```

    - Use dashes ‘-’ for list-values

```
my_dictionary:
  - list_value_one
  - list_value_two
  - list_value_three
```

    - Comments

```
{# comment that is not rendered {{var1}} #}
# comment that is rendered {{var1}}
```

      - {{var1}} is not interpreted in the 1st line, but in the 2nd line

States - YAML

- Learning by example
  - Variables

```
{% set db_version = grains['db_version'] %}          # error if undefined
{% set db_version = grains.get('db_version','') %}   # null if undef.

create_install_grinf_rsp:
  file.managed:
    - name:  /u00/app/grid/grid.{{grains['grid_version']}}.rsp

install_rdbms:
  cmd.run:
    - name: runInstaller -responseFile /tmp/rdbms.{{db_version}}.rsp -silent
```

  - rendered result:

```
create_install_grinf_rsp:
  file.managed:
    - name:  /u00/app/grid/grid.12.1.0.2.rsp

install_rdbms:
  cmd.run:
    - name: runInstaller -responseFile /tmp/rdbms.11.2.0.4.rsp -silent
```

States - YAML

- Learning by example
  - loop over a list

```
{% for node in grains.get('ora_cluster_nodes', [] ) %}
...
{% endfor %}
```

  - if-then-else

```
{% if grains.get('ora_cluster_name','') %}
  {% set asmdisks = salt['cmd.script']('salt://getasmdisk.pl').get('stdout') %}
{% endif %}
```

States - Modules

- State modules
  - Many modules are available to use in states.
    - http://docs.saltstack.com/en/latest/ref/states/all/
  - Some examples:
    - cmd: run commands on the minion
    - file: download/modify/delete files, create files from templates, etc.
    - ssh_auth: Control entries in ssh's authorized_keys file
    - sysctl: modify sysctl values
  - Do not confuse state modules with execution modules!
    - http://docs.saltstack.com/en/latest/ref/modules/all/
    - Often name is identical, but parameters are different!

Jinja Templates

- The "file" state module can work with templates
- Jinja is a template language
- state.sls file:

```
create_basenv_rsp:
  file.managed:
    - name:  /u00/app/oracle/local/dba/etc/basenv.rsp
    - source: salt://oracle/template/basenv.rsp
    - user: oracle
    - group: dba
    - template: jinja
    - context:
      grid_version: 12.1.0.2
```

- Template:

```
Oracle_Base="/u00/app/oracle"
Oracle_Home="/u00/app/grid/product/{{grid_version}}/grid"
```

Errorhandling in states

- Errorhandling is complex
  - It is not necessary to check every step, you can check for subsequent faults
  - If extraction of software fails, it is sufficient to check for successful installation

- Abort execution of StateB if previous state was unsuccessful

```
include:
  - oracle/StateA
  - oracle/StateB
```

```
# File: oracle/StateB
exit_if_StateA_failed:
  test.fail_without_changes:
    - failhard: True
    - onfail:
      -  sls: oracle/StateA
```

  - Disadvantage: StateB can only be run if StateA is run; no longer standalone

Errorhandling in states

- Abort execution of StateB if precondition is not fulfilled
  - Advantage: StateB can be run without running StateA
  - Check for existence of files, services, packages, …
    - StateA can generate a file if execution was successful (/tmp/StateA_ok)
    - To run standalone, create the file manually if precondition is fulfilled

```
# File: oracle/StateB
check_StateA_ok:
  file.exists:
    - name: /tmp/StateA_ok

exit_if_fail_StateA:
  test.fail_without_changes:
    - failhard: True
    - onfail:
      -  file: check_StateA_ok
```

Errorhandling in states

- Abort execution of StateB if precondition is not fulfilled
  - Example: install Oracle RDBMS

```
### check if responsefile is present
check_if_missing_rdbms_rsp:
  file.exists:
    - name: /u00/app/oracle/rdbms.{{ rdbms_version }}.rsp

exit_if_missing_rdbms_rsp:
  test.fail_without_changes:
    - failhard: True
    - onfail:
      -  file: check_if_missing_rdbms_rsp

### check if oracle binary is absent
check_if_existing_rdbms_bin_oracle:
  file.missing:
    - name: /u00/app/oracle/product/{{rdbms_version}}/db_1/bin/oracle

exit_if_existing_rdbms_bin_oracle:
  test.fail_without_changes:
    - failhard: True
    - name: rdbms home seems to be already installed
    - onfail:
      -  file: check_if_existing_rdbms_bin_oracle
```

AGENDA

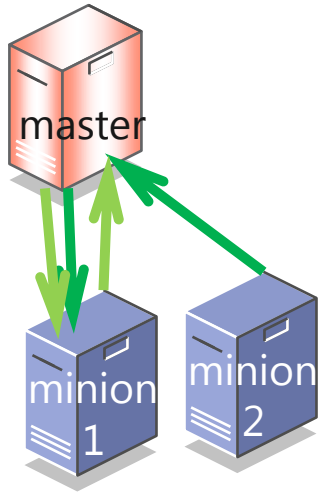
1. Introduction
   - What is Saltstack
   - Master, Minion
   - Variables
   - States
2. Setup of a RAC cluster

RAC installation

- Workflow
  - Prepare all the RAC hosts
    - Users, groups
    - Additional RPM packages
    - Shell- and Kernel-settings
    - Storage: Mountpoints, ASM-LUN's
    - **Specify Oracle versions, cluster-name and cluster-members**
    - **SSH-keys for non-interactive login between the hosts**
  - **Run the installer on the 1st host**
  - **Run the root.sh script sequentially on all hosts**
- The focus of this presentation is on the **bold** steps
  - the non-bold steps are usually done by the sysadmin team
  - the **bold steps** are usually done by the DBA team

RAC installation

- How to enforce that nodes 2 .. n are prepared before installation starts on 1st node?
  - Let the 1st node wait, until all other nodes have confirmed to be prepared
  - Poll existence of a file (/tmp/nodename.ok)
- Create a "reactor" on the master (file /etc/salt/master)
  - Gets messages from minions
  - The reaction is to write a file on the 1st node (minion)



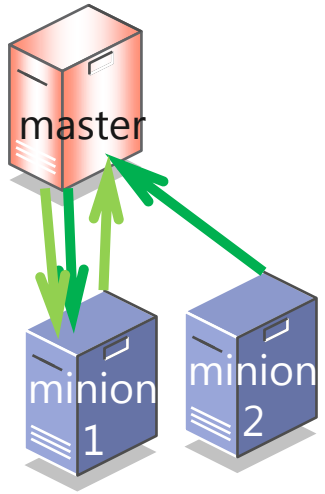

```
reactor:
  - 'oracle/ready':
    - /srv/salt/reactor/oracleready.sls
```

```
oracle_test_file:
  local.cmd.run:
    - tgt: {{ data['data']['grains']['ora_cluster_nodes'][0] }}
    - arg:
      - "echo {{ tag }} {{ data }} >> /tmp/{{ data['id'] }}.ok"
```

RAC installation

- Send a message from the minion to the master
  - If preparation on the minion is completed, send a message to the master

```
{% if grains.get('ora_cluster_nodes', False) %}
send_install_ready:
  event.send:
    - name: 'oracle/ready'
    - with_grains: True
{% endif %}
```



- Check for completion on the 1st node

```
{% if grains.get('ora_cluster_nodes', False) %}
check_if_rac_nodes_are_prepared:
  cmd.run:
    - name: while [ $(ls /tmp/*.ok 2>/dev/null|wc -l) -lt {{
grains['ora_cluster_nodes']|length() }} ] ; do sleep 10; done
{% endif %}
```

RAC installation

- Installation is done on the 1st host
  - Some scripts (root.sh) must be run sequentially on all host
    - Start it via SSH, loop over all nodes

```
{% for member in grains.get('ora_cluster_nodes', [ grains['host'] ]) %}
run_grinf_root.sh_on_{{member}}:
  cmd.run:
    - name: ssh -t {{member}} salt-call state.sls oles/oracle/run_gridinfra_root_sh
{% endfor %})
```

    - Unfortunately, this does not work on the 1st node ☹

```
The function "state.sls" is running as PID 18198 and was started at 2015, Apr 14
12:17:24.522014 with jid 20150414121724522201
```

    - So we have to create code for the first / standalone server and code for node 2..n

Non-interactive SSH login

- Initially distribute a well-known key on all hosts

```
install_temporary_authorized_keys_grid:
  ssh_auth.present:
    - source: salt://oracle/templates/tmp_id_rsa.pub
    - mode: 600
    - user: grid
    - group: dba
```

- On the 1st host
  - Add the temporary private key
  - Add all hosts to known_hosts file

```
{% for member in grains.get('ora_cluster_nodes', [ grains['host'] ]) %}
add_known_hosts_{{member}}_{{user}}:
  cmd.run:
    - name: ssh-keyscan -4 -t rsa {{member}} >> /home/{{user}}/.ssh/known_hosts
    - user: {{user}}
    - group: oinstall
{% endfor %}
```

Non-interactive SSH login

- Create the new key for the cluster (all nodes will use the same key-pair)

```
create_ssh_key_{{cluster_name}}_{{user}}:
  cmd.run:
    - name: ssh-keygen -t dsa -f /home/{{user}}/.ssh/id_rsa -P ''
    - user: {{user}}
    - group: oinstall
```

- Add the key to authorized_keys file

```
add_oracle_to_authorized_keys_{{user}}:
  ssh_auth.present:
    - user: {{user}}
    - source: /home/{{user}}/.ssh/id_rsa.pub
```

Non-interactive SSH login

- Remove the temporary key

```
remove_install_key_from_authorized_keys_{{user}}:
  ssh_auth.absent:
    - user: {{user}}
    - name: ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEA83....
```

- Distribute the files to the other nodes (all nodes will use the same files)

```
distribute_ssh_key_to_{{member}}_{{user}}:
  cmd.run:
    - name: scp -i tmp_id_rsa -p id_rsa* authorized_keys known_hosts
            {{member}}:/home/{{user}}/.ssh/
    - cwd: /home/{{user}}/.ssh
    - user: {{user}}
    - group: oinstall
```

Download of software

- Software is located on a webserver
  - To automatically download the required or latest software, an adequate naming concept is required, e.g. for PSU's:
    - <*version*>_PSU_GRID_<*yyyy-mm*>_Linux-x86-64.zip
  - Create a script that gets the filenames on the webserver
    - The script is dependent on the output of your webserver
    - Parameter 1: URL
    - Parameter 2: Filename pattern to look for
    - Optional parameter 3: return alphabetically first or last hit, default is all

```
salt['cmd.script']('salt://oracle/template/readdir.pl'
, args='"http://fileserver/Oracle/" \'(\S+x86-64_[12]of\S+\.zip)\''
, shell='/bin/bash' ).get('stdout').split()
```

Create a response file

- Take an existing response file and convert it to a template

```
ORACLE_HOSTNAME={{ORACLE_HOSTNAME}}
oracle.install.option={{install_options}}
ORACLE_HOME=/u00/app/grid/product/{{grid_version}}
oracle.install.crs.config.gpnp.scanName={{scanName}}
oracle.install.crs.config.gpnp.scanPort={{scanPort}}
oracle.install.crs.config.clusterName={{clusterName}}
oracle.install.crs.config.clusterNodes={{clusterNodes}}
oracle.install.crs.config.networkInterfaceList={{networkInterfaceList}}
oracle.install.asm.SYSASMPassword={{ASMPASSWORD}}
oracle.install.asm.diskGroup.disks={{asmdisks}}
oracle.install.asm.diskGroup.diskDiscoveryString={{ASMDISKSTRING}}
oracle.install.asm.monitorPassword={{ASMPASSWORD}}
```

- Create the response file from the template

```
create_install_grinf_rsp:
  file.managed:
    - name:  /u00/app/grid/grid.{{ grains['ora_grid_version'] }}.rsp
    - source: salt://oracle/templates/grid.{{grains['ora_grid_version']}}.rsp
    - user: grid
    - template: jinja
    - context:
      ORACLE_HOSTNAME: {{grains['fqdn']}}
      ...
```

Create a response file: Problems

- Generating passwords (1 alpha, 20 alphanumeric)
  - Accessing /dev/urandom hangs in states

```
{# set ASMPASSWORD = salt['cmd.run']("echo $(tr -cd '[:alpha:]' < /dev/urandom | fold -w1
| head -n1)$(tr -cd '[:alnum:]' < /dev/urandom | fold -w20 | head -n1)") #}
```

  - Workaround: use openssl random numbers

```
{% set ASMPASSWORD = salt['cmd.run_stdout']('echo $( openssl rand -base64 32|tr -cd
"[:alpha:]"| head -c1)$( openssl rand -base64 100|tr -cd "[:alnum:]" | head -c20)' ) %}
```

Installation

- Installation is straight forward, nothing special

```
install_gridinfra:
  cmd.run:
    - name: "unset DISPLAY; /u00/app/oracle/tmp/grid/runInstaller
            -responseFile /u00/app/grid/grid.{{grid_version}}.rsp
            -silent -ignoreSysPrereqs -force –waitforcompletion"
    - user: grid
    - group: oinstall
```

  - Run the root scripts

```
run_gridinfra_root_sh_node1:
  cmd.run:
    - name: "/u00/app/grid/product/{{grid_version}}/root.sh"

{% for member in grains.get('ora_cluster_nodes', [ grains['host'] ]) %}
{% if member != grains['host'] %}
run_grinf_root.sh_on_{{member}}:
  cmd.run:
    - name: ssh -t {{member}} salt-call state.sls oracle/run_grid_root
```

Installation in customer's environment

- Additional steps at customers site:
  - Install the database home similarly
  - Apply the latest Patchset Update to the Grid- and Database-home
  - Configure additional diskgroups
  - Install and configure Enterprise Manager agent
  - Configure the environment: Trivadis BasEnv and customer-specific scripts

Advantages

- Using the same standard software as the Unix team
  - Standardization
  - Saves a lot of time
- Works well, very flexible

Trivadis an der DOAG 2015

Ebene 3 - gleich neben der Rolltreppe

Wir freuen uns auf Ihren Besuch.

Denn mit Trivadis gewinnen Sie immer.

Exaday 13.4.2016 Hamburg (CFP)