ORACLE
DATABASE APPLIANCE

Oracle Database Appliance: Automated Virtual Machine Provisioning

ORACLE WHITE PAPER | JANUARY 2017

ORACLE

Table of Contents

Introduction

Oracle VM Templates provide an innovative approach to deploying a fully configured software stack by offering pre-installed and pre-configured software images. Use of Oracle VM Templates eliminates the installation and configuration costs, and reduces the ongoing maintenance costs helping organizations achieve faster time to market and lower cost of operations. Oracle VM Templates are part of many key Oracle products available for download, including Oracle Linux, Oracle Solaris, Oracle Database, Fusion Middleware, and many more. Simply download an Oracle VM Template from Oracle Software Delivery Cloud, import it into Oracle Database Appliance and then deploy the Template as a virtual machine in order to use the pre-configured software.

Oracle VM Guest Additions allow the guest software to pass information through Oracle OAK (Oracle Appliance Kit) Manager to the virtual machine, and thus provide direct integration between guest software and the virtualization layer, to assist in orchestration of complex, multi-VM deployments.

The rest of this technical white paper focuses on how to automate virtual machine provisioning based on Oracle VM Templates with the Oracle VM Guest Additions providing an example ("*DBCString - A SendKey script to configure DB Connect String*") on how to build an action addition script.

Concepts: Oracle VM Guest Additions

Oracle VM Guest Additions is a set of packages that can be installed on the guest operating system of a virtual machine running in the Oracle Database Appliance VP (Virtualized Platform) environment. These packages provide the tools to allow communication directly between OAK Manager and the operating system running within the virtual machine. This is a powerful tool that provides administrators fine-grained control over the configuration and behavior of components running within the virtual machine directly from ODA_BASE. Features of the Oracle VM Guest Additions include the option to send messages directly to a virtual machine from ODA_BASE to trigger programmed events, and the ability to use the template configuration facility to automatically configure virtual machines as they are first started.

These Guest Additions are available for Oracle Linux 5, Oracle Linux 6, and Oracle Linux 7 from Oracle's Public YUM repository, and can be installed in the guest with the following command:

```
# yum install ovmd xenstoreprovider python-simplejson ovm-template-config
```

This installs the basic necessary packages to support the Oracle VM Guest Additions:

- **ovmd** is a daemon that handles configuration and re-configuration events and provides a mechanism to send messages between the virtual machine and Oracle Appliance Kit Manager from ODA_BASE.
- **xenstoreprovider** is an information storage space shared between domains. It looks for specific messages (key-value pairs or events) and passes those to ovmd, or the other way around.
- **python-simplejson** is a simple, fast, extensible JSON encoder/decoder for Python.
- **ovm-template-config** is a collection of OS configuration system scripts used to (re)configure an Oracle VM template when booted up the first time.
- **libovmapi** is the library which communicates with the ovmapi kernel infrastructure. This package will be automatically installed because it is a dependency.
- **libovmapi-devel** is an optional package to be installed when creating additional extensions to ovmd.

There is an extra kernel module required to make this work, the `ovmapi` kernel module that provides the ability to communicate messages between the ODA VP and the VM and as such between OAK Manager and the VM. Since UEK (2.6.39) this kernel module is shipped with the kernel.

Messaging Channel between Guest and OAK Manager

The Oracle VM Guest Additions daemon: ovmd, facilitates a messaging channel between Oracle Appliance Kit Manager and the guest. It allows first-boot installation configuration, and is capable of receiving messages consisting of key-value pairs.

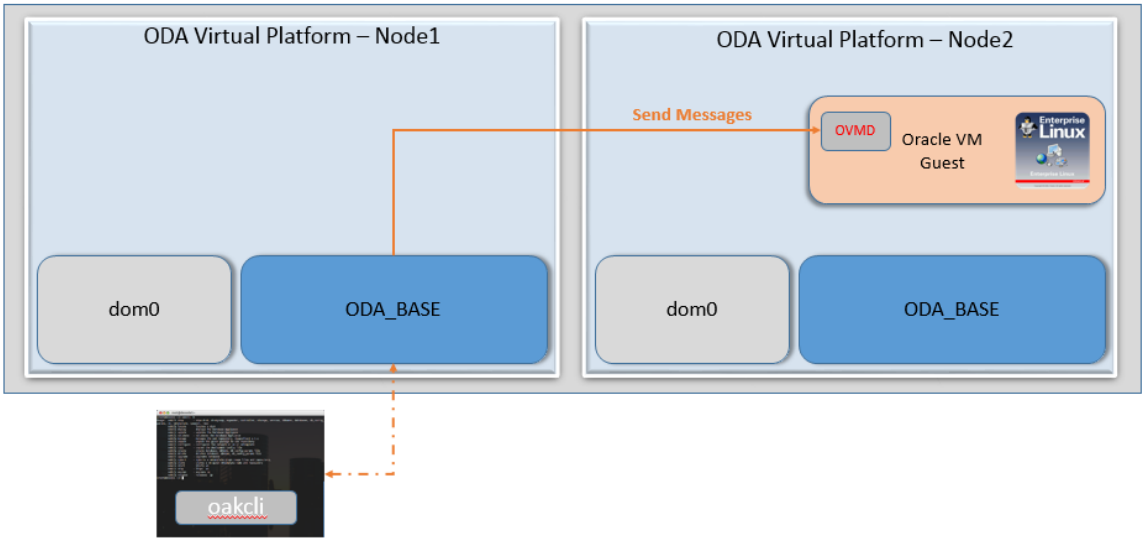


Figure 1. Messaging Chanel between Guest and ODA_BASE

Sending messages via Oracle Appliance Kit (OAK) can be done by using the oakcli command line Interface. Inside the guest, ovmd is responsible for receiving messages.

Send Messages to VM

To send a message via the oakcli command line interface, use the following syntax:

```
# oakcli modify vm <vmname> -s <key1:value1;key2:value2....>
```

Example:

```
# oakcli modify vm OL6U8 -s com.oracle.db.network.servicename:ORCL
Sent Pairs
```

Message Handling inside the Guest

Inside the guest, the ovmd executable can be used to receive messages. ovmd has the following options:

```
# ovmd -l                   lists all currently set key/value pairs
# ovmd -p key=value         sets a key/value pair inside the VM
# ovmd -g key               gets a value from inside the VM
# ovmd -r key               removes a key out of the current cache
# ovmd -x                   deletes the key/value values currently set in the cache
```

Example:

```
# ovmd -p servicename=ORCL          <-- (sets a key "servicename" with the value "ORCL")
# ovmd -l                           <-- (lists all keys)
{"servicename":"ORCL"}              <-- (there is only one key/value pair)

# ovmd -g servicename               <-- (gets the value of key "servicename")
ORCL

# ovmd -r servicename               <-- (removes the key "servicename")
# ovmd -l                           <-- (lists all keys)
                                    <-- (there are no more key/value pairs)
```

With these simple tools, it's possible to set up a model to send messages from an application outside of a virtual machine to a virtual machine through the Oracle VM Guest Additions. A recommendation would be to create a naming convention for this product. For instance, for the Oracle VM Template configuration `com.oracle.linux.[values]` is used. Something similar could be considered or just something like `[application].[key]`. The maximum size of the total message is 8Kb

Under the hood: Oracle VM Template Configuration

As earlier mentioned, ovmd is a utility (daemon) that handles configuration and re-configuration events, and provides a mechanism to receive messages between a virtual machine and ODA_BASE. By enabling the initial-configuration option in the Template, this utility is used to perform first-boot installation configuration either locally from the virtual machine console or remotely through the messaging interface provided by this utility. During startup of the VM, '`ovmd -s configure`' will be executed which waits till all "required" parameters are received and then all the configuration scripts are executed. By default, the scripts have only one parameter that is "required," which is the root-password to configure the system root password; all other parameters are optional. Required parameters need to be sent at the end of the configuration because once they are received, the actual configuration will be executed. Optional parameters sent afterwards will be ignored.

To verify the root-password is a required parameter, the following command can be used:

```
# ovm-template-config --human-readable --enumerate --script authentication configure
[('90',
  'authentication',
  [{u'description': u'System root password.',
    u'key': u'com.oracle.linux.root-password',
    u'password': True,
    u'required': True}])]
```

and to verify that is the only required parameter (by default):

```
# grep "required" /etc/template.d/scripts/*       <-- (location of configuration scripts)
/etc/template.d/scripts/authentication:                  'required': True}]
```

To use another "trigger" (required parameter), other than the root-password, modify e.g. `/etc/template.d/scripts/network` to make another parameter (e.g., the IP address) a required parameter. As mentioned at least one required parameter is needed and if there are multiple, the trigger will happen once all are received. Once the above message gets sent, the `ovm-template-config` scripts will set up all the values and the virtual machine will end up in a configured state.

Example 1: Automated Virtual Machine Provisioning

Demonstrating how virtual machines can be automatically provisioned by using Oracle VM Templates and Oracle VM Guest Additions will be done by means of an example where the hostname, network settings and root user password will be automatically configured when the virtual machine boots for the first time after creation.

Step by step: Creating an Oracle VM Template with Guest Additions

This chapter is a step-by-step guide explaining how to create an Oracle VM Template from scratch and how to install and configure the Oracle VM Guest Additions. The latest available Oracle Linux 6 will be used, which is OL 6.8 at the time of writing. A similar approach can be followed for Oracle Linux 5. Where there are significant differences in the installation procedure they will be mentioned. Typically, users will skip this part because ready-to-go Oracle VM Templates can be downloaded from Oracle Software Delivery Cloud, e.g. Oracle Linux 6.8 Oracle VM Template (V776594-01). As in all recent Oracle VM Templates, the Oracle VM Guest Additions are already installed.

Steps to follow if your Oracle Linux Guest VM is coming from Oracle VM Template

- Step 1: Import template

```
oakcli import vmtemplate <name> -assembly http://.../OVM OL6U8 x86 64 PVHVM.ova -repo <name> -node 0
```

Example:

```
# oakcli import vmtemplate OL6U8 -assembly http://oda1/OVM_OL6U8_x86_64_PVHVM.ova -repo srepo1 -node 0
```

- Step 2: Clone VM

```
oakcli clone vm <vm_name> -vmtemplate <src_vmtempl_name> -snap
```

Example:

```
oakcli clone vm OL6U8 -vmtemplate OL6U8 -snap
```

- Step 3: Setup the VM network

```
oakcli configure vm <name> -network "['type=netfront,bridge=net1']"
```

Example:

```
oakcli configure vm OL6U8 -network "['type=netfront,bridge=net1']"
```

- Step 4: Start the VM

```
oakcli start vm <name>
```

Example:

```
oakcli start vm OL6U8
```

Steps to follow if your Oracle Linux Guest VM is coming from an ISO image:

- Step 1: Create a virtual machine with a default installation of Oracle Linux 6.8

- Step 2: As a best practice update the virtual machine to the latest version of UEK and to the latest patches. Once done reboot the virtual machine.

```
# yum update
# reboot
```

Oracle Linux updates are freely available on Oracle's Public YUM repository and the default install of Oracle Linux 6.8 already points to this location for updates. Note as the network connectivity for the VM needs to be established in order to use "`yum update`".

Guest VM network setup

In order to setup the network you need to add a virtual network interface "`vif`" as following:

```
oakcli configure vm <name> -network "['type=netfront,bridge=net1']"
```

Example:

```
oakcli configure vm OL6U8 -network "['type=netfront,bridge=net1']"
```

After the Guest VM reboot you need to configure the network making the related changes into "`/etc/sysconfig/network-scripts/ifcfg-eth0`", example:

```
[root@OL6U8 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
NM_CONTROLLED=no
USERCTL=no
BOOTPROTO=none
IPADDR=10.168.1.62
NETMASK=255.255.0.0
GATEWAY=10.168.1.1
DEVICE=eth1
TYPE=ETHERNET
```

- Step 3: Enable the Oracle Linux add-on channel. Download the latest public-yum repository file from Oracle's Public YUM repository which contains more repositories and enable the add-on channel which contains the Oracle VM Guest Additions package:

```
# cd /etc/yum.repos.d
# rm public-yum-ol6.repo           <-- (replace the original version with this newer version)
# wget http://public-yum.oracle.com/public-yum-ol6.repo
```

Edit the `public-yum-ol6.repo` file to enable the ol6_addons channel. Find the ol6_addons section and change `enabled=0` to `enabled=1`.

```
[ol6_addons]
name=Oracle Linux $releasever Add ons ($basearch)
baseurl=http://public-yum.oracle.com/repo/OracleLinux/OL6/addons/$basearch/
gpgkey=http://public-yum.oracle.com/RPM-GPG-KEY-oracle-ol6
gpgcheck=1
enabled=1
```

Note: For Oracle Linux 5 the public-yum repo file is located at http://public-yum.oracle.com/public-yum-el5.repo

- Step 4: Install the Oracle VM Guest Additions package. These are available for Oracle Linux 5, 6 and 7.

```
# yum install ovmd xenstoreprovider python-simplejson ovm-template-config
```

- Step 5: Install additional Oracle VM Template configuration packages:

```
ovm-template-config-authentication : Oracle VM template authentication configuration script
ovm-template-config-datetime       : Oracle VM template datetime configuration script
ovm-template-config-firewall       : Oracle VM template firewall configuration script
ovm-template-config-network        : Oracle VM template network configuration script
ovm-template-config-selinux        : Oracle VM template selinux configuration script
ovm-template-config-ssh            : Oracle VM template ssh configuration script
ovm-template-config-system         : Oracle VM template system configuration script
ovm-template-config-user           : Oracle VM template user configuration script
```

For demo purposes all these additional packages can be installed, although strictly speaking not all are needed.

```
# yum install ovm-template-config-*
```

Guest VM from ISO and Guest VM from Oracle Template common steps

- Step 1: Enable ovmd to be able to send and receive messages between the virtual machine and Oracle Appliance Kit Client (OAKCLI) from OAD_BASE.

```
# chkconfig ovmd on
# /etc/init.d/ovmd start
```

- Step 2: Now that all configuration packages are installed, scripts can be selectively enabled and disabled. This works very similar to the `chkconfig` command. To check which scripts/modules are registered and whether they are enabled to run at configure time and/or cleanup time, execute following command:

```
# ovm-chkconfig --list
name                 configure   unconfigure reconfigure cleanup     suspend     resume      migrate
shutdown
authentication       on:90       off         off         on:10       off         off         off         off
datetime             on:50       off         off         on:50       off         off         off         off
dbcstring            on:80       off         off         on:20       off         off         off         off
firewall             on:41       off         off         off         off         off         off         off
network              on:50       off         off         on:50       off         off         off         off
selinux              on:30       off         off         off         off         off         off         off
ssh                  on:70       off         off         on:30       off         off         off         off
system               on:60       off         off         on:60       off         off         off         off
user                 on:60       off         off         on:40       off         off         off         off

# ovm-chkconfig --add authentication      <-- (to enable all targets supported by a module)
# ovm-chkconfig --del datetime            <-- (to disable all targets supported by a module)
# ovm-chkconfig --target=cleanup user off <-- (to enable or disable particular targets for a module)
```

The two main targets are `configure` and `cleanup`. There are other targets available but they are not yet implemented at the time of writing. For example, to configure the hostname, the network settings and the root password when the virtual machine boots for the first time, the network module needs to be enabled when the virtual machine boots, i.e. at configure time. Make sure authentication is also enabled at configuration time to configure the root password, this to have a required parameter. To enable the authentication and network module, in case this isn't done already, execute following command and verify they are enabled:

```
# ovm-chkconfig --target configure authentication on
# ovm-chkconfig --target configure,cleanup network on
# ovm-chkconfig --list
```

- Step 3: For the Template configuration that is provided, and depending on optional scripts that are installed by the user, there is a well-defined set of variables (keys) that can be set. To get the list of configuration keys in a readable format execute following command:

```
# ovm-template-config --human-readable --enumerate configure
```

or, to get only a subset belonging to the network configure script:

```
# ovm-template-config --human-readable --enumerate --script network configure
```

For this example, the following keys are interesting:

```
com.oracle.linux.hostname                 : System host name, e.g. "localhost.localdomain"
com.oracle.linux.network.host.0           : Hostname entry for /etc/hosts, e.g.,
                                                 "127.0.0.1 localhost.localdomain localhost".
com.oracle.linux.network.device.0         : Network device to configure, e.g. "eth0"
com.oracle.linux.network.onboot.0         : Activate interface on system boot: yes or no
com.oracle.linux.network.bootproto.0      : Boot protocol: dhcp or static
com.oracle.linux.network.ipaddr.0         : IP address of the interface
com.oracle.linux.network.netmask.0        : Netmask of the interface.
com.oracle.linux.network.gateway.0        : Gateway IP address
com.oracle.linux.network.dns-servers.0    : DNS servers separated by comma, e.g., "8.8.8.8,8.8.4.4"
```

- Step 4: Now that the Template configuration is done, the virtual machine needs to be configured for first-boot. Run these commands on the virtual machine console because we enabled the network configuration to be cleaned, meaning network connectivity to the virtual machine will be lost.

```
# ovmd -s cleanup                         <-- (reinitializes/cleans up the Template)
# service ovmd enable-initial-config      <-- (enables the first-boot configuration)
# shutdown -h now
```

After cloning this virtual machine or starting it, it will act as a first time boot virtual machine and it will require configuration input on the virtual machine console or through the Oracle VM API (more details below). To create multiple instances of the virtual machine prepared as Template, it can be cloned to a new virtual machine or Template.

Note: starting with ODA v. 12.1.2.0.0 we can clone a VM from another VM

```
# oakcli clone vm <vm_name> -vm <src_vm_name> -snap
```

Local Configuration via Virtual Machine Console

Oracle VM Template configuration can be done locally by entering the parameter values via the virtual machine console when the virtual machine boots the first time.

- Step 1: After starting the virtual machine for the first time the Oracle VM Template configuration wizard will be started and the user will be able to specify values for hostname, network settings and the system root password. As mentioned before, it's only the root-password that is required and that needs to be filled in; all other parameters are optional and can be skipped by pressing "enter".

- Step 2: Once the root-password has been filled in the actual configuration will be started and the boot process completed.

Note-1: You can connect the Guest VM console using VNC client if the virtual frame buffer "vfb" is configured on your vm.cfg

- The backend listens on IP 127.0.0.1 port 5900+N by default where N is the domain ID. Both, address and N can be changed (IP address 0.0.0.0 means dom0 IP)

```
vfb = [ 'type=vnc,vnclisten=0.0.0.0,vncdisplay=10' ]
```

You can now connect to this system with `vncviewer oda_dom0:<VNCDisplayID>`, example is: `vncviewer dom0-hostname:5910`

Or you can bind the first unused port above 5900:

```
vfb = [ 'type=vnc,vnclisten=0.0.0.0,vncunused=1' ]
```

Note-2: You can connect the Guest VM console within ODA graphical environment issuing: "`oakcli show vmconsole <vm-name>`"

Remote Configuration via OAKCLI

Oracle VM Template configuration can also be done from ODA_BASE through the Oracle VM API. In the following steps, the same information (hostname, network settings, root password) will be entered by using the Oracle Appliance Kit Client Interface (OAKCLI) without any manual intervention via the virtual machine console.

- Step 1: Create and boot a virtual machine that is cloned from the Template. To follow the progress of the boot process the virtual machine console can be used. The configuration wizard will be launched but unlike in the previous chapter nothing will be inputted.

```
# oakcli clone vm OL6U8 -vmtemplate OL6U8 -snap
# oakcli start vm OL6U8
```

- Step 2: Send the network parameters and lastly the (required) root-password

```
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.device.0:eth0'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.onboot.0:yes'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.bootproto.0:static'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.ipaddr.0:10.214.120.23'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.netmask.0:255.255.252.0'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.gateway.0:10.214.120.1'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.dns-servers.0:130.35.249.52'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.hostname:rwsoda309c4n1'
oakcli modify vm OL6U8 -s 'com.oracle.linux.root-password:q2w3e4r5'
```

After the root-password has been send the actual configuration will be executed and the boot process continued. Once completed the virtual machine is ready to go and can be accessed through ssh.

Example 2: DBCString OVM guest addition script to configure a DB connect string

In the following example, your application, pre-installed into the Guest VM, needs to connect a remote database using a connect string. We want demonstrate how we could automate the configuration of such “DB connect string” once the guest VM is started up. The dbcstring (ovm template config script) will generate under the guest VM '/tmp' folder, two files:

```
# ls
dbcstring.ora tnsnames.ora
```

With the following content (example):

```
# cat dbcstring.ora
oda2:3042/ORCL

# cat tnsnames.ora
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = oda2 )(PORT = 3042))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ORCL )
    )
```

Assume the application has been written to look at this location for connect string information, and process it when found.

Developing DBCScript Oracle VM Template Configuration Modules

The provided module scripts are developed in Python. The script consists of 2 main parts:

- the script header which contains information like script name, targets, priorities and description.
- the actual script, which handles a small set of parameters.

Module Script Header

Module script headers require a very specific comment block in order for ovm-chkconfig to handle enabling and disabling your script functionality. The format for the script header is as follows:

```
### BEGIN PLUGIN INFO
# name: [script name]
# [target]: [priority]
# [target]: [priority]
# description: a description that can
#   cross multiple lines.
### END PLUGIN INFO
```

In case of DBCString:

```
### BEGIN PLUGIN INFO
# name: dbcstring
# configure: 80
# cleanup: 20
# description: Script to configure template authentication.
### END PLUGIN INFO
```

Module Script Body

The main requirement for the module script body is that it accepts at least one target parameter. Target parameters that might get presented by the ovm-template-configure script include:

- configure
- unconfigure
- reconfigure
- cleanup
- suspend
- resume
- migrate
- shutdown

Your script can handle any other arguments that you require. There is one optional parameter which is useful to implement and this is -e or --enumerate. ovm-template-config uses this to be able to enumerate or list the parameters for a target supported by your script.

A very basic template to use for your script body follows:

```
try:
    import json
except ImportError:
    import simplejson as json
from templateconfig.cli import main


def do_enumerate(target):
    param = []
    if target == 'configure':
        param += []
    elif target == 'cleanup':
        param += []
    return json.dumps(param)


def do_configure(param):
    param = json.loads(param)
    return json.dumps(param)


def do_cleanup(param):
    param = json.loads(param)
    return json.dumps(param)


if __name__ == '__main__':
    main(do_enumerate, {'configure': do_configure, 'cleanup': do_cleanup})
```

This script supports the configure and cleanup targets. You can fill out the script with your own code. For instance, for the `do_enumerate` function, you would populate the parameters that are supported for each target in the script. An example from DBCString module is presented below:

```
def do_enumerate(target):
    param = []
    if target == 'configure':
        param += [ {'key': 'com.oracle.db.network.scanname',
                    'description': 'Database Scan name, e.g. "oda1-scan.us.oracle.com".',
                    'required': True},
                    {'key': 'com.oracle.db.network.port',
                    'description': 'Listener port, e.g. "1521".',
                    'hidden': False},
                    {'key': 'com.oracle.db.network.servicename',
                    'description': 'Database Service name, e.g. "ORCL".',
                    'required': True}]
    return json.dumps(param)
```

'required' means ovm-template-config will wait for the parameter before calling scripts. ovm-template-config will call scripts immediately when all the 'required' parameters are provided. It will ignore all further parameters. So you may want to send a required parameter at last.
'hidden' : False --> show the parameter in the guest VM console screen for input
'hidden' : True --> don't show the parameter in the guest VM console screen.

Each target function begins by reading the JSON parameters passed to the script, using the `param = json.loads(param)` statement. From this point, code can be written to perform actions based on the values of the keys that the script expects to receive.

```
def do_configure(param):
    param = json.loads(param)
    service_name = param.get('com.oracle.db.network.servicename')
    if not service_name:
        raise Exception('DB Service name is required.')
    scan_name = param.get('com.oracle.db.network.scanname')
    if not scan_name:
        raise Exception('DB scan name is required.')
    port = param.get('com.oracle.db.network.port')
    if not port:
        port = "1521"
    tnsnames = open('/tmp/tnsnames.ora', 'w')
    tnsnames.write(service_name + ' =\n')
    tnsnames.write('  (DESCRIPTION =\n')
    tnsnames.write('    (ADDRESS = (PROTOCOL = TCP)(HOST = ' + scan_name + ' )(PORT = ' + port + '))\n')
    tnsnames.write('    (CONNECT_DATA =\n')
    tnsnames.write('      (SERVER = DEDICATED)\n')
    tnsnames.write('      (SERVICE_NAME = %s )\n' % service_name)
    tnsnames.write('    )\n')
    tnsnames.write('  )\n')
    tnsnames.close()
    dbcstring = open('/tmp/dbcstring.ora', 'w')
    dbcstring.write(scan_name + ':' + port + '/' + service_name + '\n')
    dbcstring.close()
    return json.dumps(param)
```

DBCString:

```
### BEGIN PLUGIN INFO
# name: dbcstring
# configure: 80
# cleanup: 20
# description: Script to configure template authentication.
### END PLUGIN INFO

try:
    import json
except ImportError:
    import simplejson as json
from templateconfig.cli import main
from templateconfig.common import run_cmd

def do_enumerate(target):
    param = []
    if target == 'configure':
        param += [ {'key': 'com.oracle.db.network.scanname',
                    'description': 'Database Scan name, e.g. "oda1-scan.us.oracle.com".',
                    'required': True},
                    {'key': 'com.oracle.db.network.port',
                    'description': 'Listener port, e.g. "1521".',
                    'hidden': False},
                    {'key': 'com.oracle.db.network.servicename',
                    'description': 'Database Service name, e.g. "ORCL".',
                    'required': True}]
    return json.dumps(param)

def do_configure(param):
    param = json.loads(param)
    service_name = param.get('com.oracle.db.network.servicename')
    if not service_name:
        raise Exception('DB Service name is required.')
    scan_name = param.get('com.oracle.db.network.scanname')
    if not scan_name:
        raise Exception('DB scan name is required.')
    port = param.get('com.oracle.db.network.port')
    if not port:
        port = "1521"
    tnsnames = open('/tmp/tnsnames.ora', 'w')
    tnsnames.write(service_name + ' =\n')
    tnsnames.write('  (DESCRIPTION =\n')
    tnsnames.write('    (ADDRESS = (PROTOCOL = TCP)(HOST = ' + scan_name + ' )(PORT = ' + port + '))\n')
    tnsnames.write('    (CONNECT_DATA =\n')
    tnsnames.write('      (SERVER = DEDICATED)\n')
    tnsnames.write('      (SERVICE_NAME = %s )\n' % service_name)
    tnsnames.write('    )\n')
    tnsnames.write('  )\n')
    tnsnames.close()
    dbcstring = open('/tmp/dbcstring.ora', 'w')
    dbcstring.write(scan_name + ':' + port + '/' + service_name + '\n')
    dbcstring.close()
    return json.dumps(param)


def do_cleanup(param):
    param = json.loads(param)
    run_cmd(['rm', '-f', '/tmp/tnsnames.ora'])
    run_cmd(['rm', '-f', '/tmp/dbcstring.ora'])
    return json.dumps(param)


if __name__ == '__main__':
    main(do_enumerate, {'configure': do_configure, 'cleanup': do_cleanup})
```

Module Script Packaging

Once you have written one or more configuration module scripts, you may want to package them as RPMs that can be deployed on other systems. In order to install and configure template configure scripts, they have to be packaged in an RPM, with a specific naming convention. Package the script as `ovm-template-config-[scriptname]`. Ideally in the post install of the RPM you should add the script automatically by executing `/usr/sbin/ovm-chkconfig --add [scriptname]`. When de-installing a script/RPM, remove it at un-install time using `/usr/sbin/ovm-chkconfig --del [scriptname]`.This is illustrated in the following example of an RPM spec file that can be used (`dbcstring.spec`):

```
Name: ovm-template-config-dbcstring
Version: 1.0
Release: 1%{?dist}
Summary: SendKey DB Connect String configuration script.
Group: Applications/System
License: GPL
URL: http://www.oracle.com/virtualization
Source0: %{name}-%{version}.tar.gz
BuildRoot: %(mktemp -ud %{_tmppath}/%{name}-%{version}-%{release}-XXXXXX)
BuildArch: noarch
Requires: ovm-template-config


%description
SendKey DB Connect String configuration script.

%prep
%setup -q

%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT

%clean
rm -rf $RPM_BUILD_ROOT

%post
if [ $1 = 1 ]; then
    /usr/sbin/ovm-chkconfig --add dbcstring
fi

%preun
if [ $1 = 0 ]; then
    /usr/sbin/ovm-chkconfig --del dbcstring
fi

%files
%defattr(-,root,root,-)
%{_sysconfdir}/template.d/scripts/dbcstring

%changelog
* Wed Sep 18 2014 Ruggero Citton - 1.0-1
- Initial build.
```

In order to create RPMs, you must install rpmbuild:

```
# yum install rpm-build
```

The following example Makefile may help to automate the build process:

```
DESTDIR=
PACKAGE=ovm-template-config-dbcstring
VERSION=1.0

help:
	@echo 'Commonly used make targets:'
	@echo '  install    - install program'
	@echo '  dist       - create a source tarball'
	@echo '  rpm        - build RPM packages'
	@echo '  clean      - remove files created by other targets'


install:
	install -D dbcstring $(DESTDIR)/etc/template.d/scripts/dbcstring

dist: clean
	mkdir $(PACKAGE)-$(VERSION)
	tar -cSp --to-stdout --exclude .svn --exclude .hg --exclude .hgignore --exclude
$(PACKAGE)-$(VERSION) * | tar -x -C $(PACKAGE)-$(VERSION)
	tar -czSpf $(PACKAGE)-$(VERSION).tar.gz $(PACKAGE)-$(VERSION)
	rm -rf $(PACKAGE)-$(VERSION)

rpm: dist
	rpmbuild -ta $(PACKAGE)-$(VERSION).tar.gz

clean:
	rm -fr $(PACKAGE)-$(VERSION)
	find . -name '*.py[cdo]' -exec rm -f '{}' ';'
	rm -f *.tar.gz
	rm -fr ./../rpmbuild

.PHONY: dist install rpm clean
```

Remember to edit this Makefile to reference your own script.

Create a working directory, copy over your script, the spec file and the Makefile. Run:

```
# make dist
```

to create a src tarball of your code and then run:

```
# make rpm
```

This will generate an RPM in the RPMS/noarch directory within your working directory. For example:

RPMS/noarch/ovm-template-config-dbcstring-1.0-1.noarch.rpm

Using DBCString Oracle VM Template Configuration Modules

Guest VM Import/clone

» Import template

```
oakcli import vmtemplate <name> -assembly http://.../OVM_OL6U8_x86_64_PVHVM.ova -repo <name> -node 0
```

Example:

```
# oakcli import vmtemplate OL6U8 -assembly http://oda1/OVM_OL6U8_x86_64_ PVHVM.ova -repo srepo1 -node
```

» Clone VM

```
oakcli clone vm <vm_name> -vmtemplate <src_vmtempl_name> -snap
```

Example:

```
oakcli clone vm OL6U8 -vmtemplate OL6U8 -snap
```

» Setup the VM network

```
oakcli configure vm <name> -network "['type=netfront,bridge=net1']"
```

Example:

```
oakcli configure vm OL6U8 -network "['type=netfront,bridge=net1']"
```

» Start the VM

```
oakcli start vm <name>
```

Example:

```
oakcli start vm OL6U8
```

DBCString setup

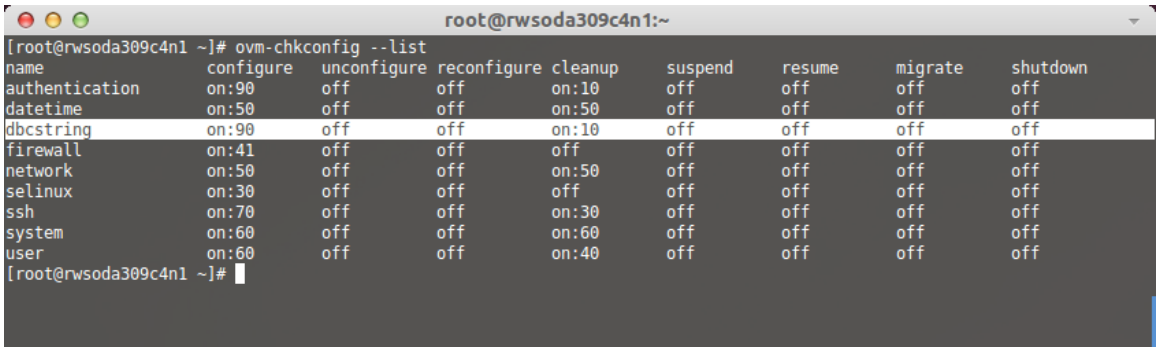
Into the new Guest VM (OL6U8) issue the following commands:

1. Install the dbcstring RPM:

```
# rpm -Uvh ovm-template-config-dbcstring-1.0-1.noarch.rpm
```

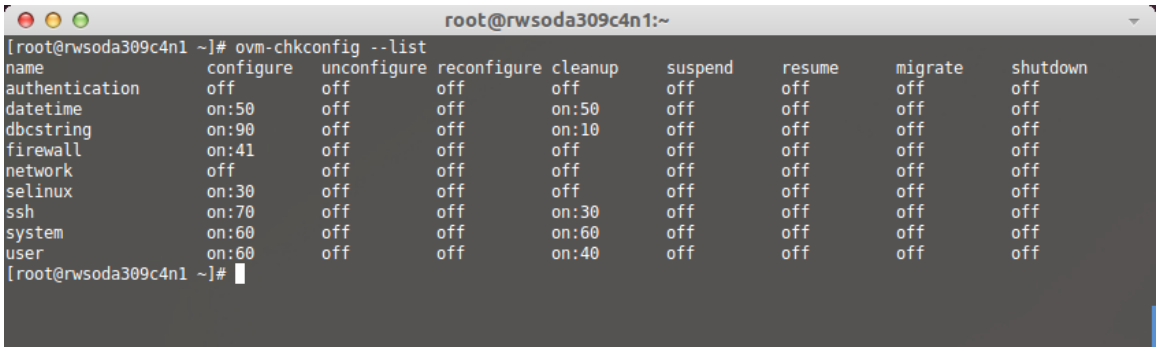
2. You can check if dbcstring is armed:

```
# ovm-chkconfig --list
```



3. Disable authentication & network setup, if you want keep root password and network setup without change at boot time:

```
# ovm-chkconfig --del authentication
# ovm-chkconfig --del network
```



4. Enable the OVMD initial config:

```
# service ovmd enable-initial-config
```

5. Reboot the Guest VM:

```
# reboot
```

DBCString - SendKeys

From ODA_BASE, with authentication & network disabled (see step 3 above), you need to send the following three keys:

```
# oakcli modify vm <vm_name> -s 'com.oracle.db.network.scanname:oda1-scan'
# oakcli modify vm <vm_name> -s 'com.oracle.db.network.port:1521'          ←(optional, hidden:false)
# oakcli modify vm <vm_name> -s 'com.oracle.db.network.servicename:ORCL'
```

Example:

```
oakcli modify vm OL6U8 -s 'com.oracle.db.network.scanname:oda1-scan'
oakcli modify vm OL6U8 -s 'com.oracle.db.network.port:1521'               ←(optional, hidden:false)
oakcli modify vm OL6U8 -s 'com.oracle.db.network.servicename:ORCL'
```

If you keep authentication & network enabled, you must provide all following keys (change the values with your own):

```
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.device.0:eth0'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.onboot.0:yes'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.bootproto.0:static'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.ipaddr.0:10.214.120.23'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.netmask.0:255.255.252.0'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.gateway.0:10.214.120.1'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.dns-servers.0:130.35.249.52'
oakcli modify vm OL6U8 -s 'com.oracle.linux.network.hostname:rwsoda309c4n1'
oakcli modify vm OL6U8 -s 'com.oracle.db.network.scanname:oda1-scan'
oakcli modify vm OL6U8 -s 'com.oracle.db.network.port:1521'               ←(optional, hidden:false)
oakcli modify vm OL6U8 -s 'com.oracle.db.network.servicename:ORCL'
oakcli modify vm OL6U8 -s 'com.oracle.linux.root-password:q2w3e4r5'
```

Summary

Oracle VM Guest Additions provides the tools to allow communication directly between ODA_BASE and the operating system running within the guest virtual machine. This is a powerful tool that provides administrators fine-grained control over the configuration and behavior of components running within the virtual machine directly from OAK Manager. Together with Oracle VM Templates, first-boot installation configuration can be performed either manually, locally from the virtual machine console, or in an automated way, remotely through the messaging interface provided by the Oracle VM API. Leveraging this feature you can automate the applications deployment inside the VM Guest.

Appendix: Testing of Template Configuration Scripts

Testing Template configuration scripts on a per script basis can be done in several ways. Just keep in mind at least 1 required key (e.g. the root password, if authentication module is 'on') is needed to trigger some action.

» Option 1

Configuration can be done directly from the virtual machine console by running the script with the `--console-input` option. This will prompt for values for each of the keys that need to be defined for any enabled modules:

```
# ovm-template-config -s authentication --console-input configure
authentication: System root password.:
```

or in case of dbcstring (note as it's using required keys):

```
# ovm-template-config -s dbcstring --console-input configure
dbcstring: Database Scan name, e.g. "oda1-scan.us.oracle.com".: oda2-scan.it.oracle.com
dbcstring: Listener port, e.g. "1521".: 3042
dbcstring: Database Service name, e.g. "ORCL".: JDE
```

» Option 2

After defining key-value pairs they can be passed to a Template configuration script with the following command:

```
ovmd -l | ovm-template-config --script authentication --stdin configure
```

Example (using authentication module):

```
# ovmd -l | ovm-template-config --script authentication --stdin configure
missing value for key "com.oracle.linux.root-password" of script "authentication"
                                                        ←(value for the required key is missing)

# ovmd -p com.oracle.linux.root-password=password123          ←(root password is set)
# ovmd -l
{"com.oracle.linux.root-password":"password123"}

# ovmd -l | ovm-template-config --script authentication --stdin configure
#                                     ←(authentication module is executed and root password is set)
```

Example (using dbcstring module):

```
# ls /tmp/*.ora
ls: cannot access /tmp/*.ora: No such file or directory                    ←(dbcstring not created yet)

# ovmd -l | ovm-template-config --script dbcstring --stdin configure
missing value for key "com.oracle.db.network.scanname" of script "dbcstring"      ←(required scan_name)

# ovmd -p com.oracle.db.network.scanname=oda1-scan                        ←(required scan_name provided)

# ovmd -l | ovm-template-config --script dbcstring --stdin configure
missing value for key "com.oracle.db.network.servicename" of script "dbcstring"  ←(required service_name)

# ovmd -p com.oracle.db.network.port=1521                                 ←(optional parameter provided)

# ovmd -l | ovm-template-config --script dbcstring --stdin configure
missing value for key "com.oracle.db.network.servicename" of script "dbcstring"  ←(required service_name)

# ovmd -p com.oracle.db.network.servicename=ORCL                          ←(all required keys are provided)

# ovmd -l | ovm-template-config --script dbcstring --stdin configure              ←(dbcstring execute)

# ls /tmp/*.ora
/tmp/dbcstring.ora  /tmp/tnsnames.ora                                      ←(dbcstring created!)
```

» Option 3

Key-value pairs are actually passed to ovm-template-config in JSON format. They can be passed to a Template configuration script by creating a file e.g. `/tmp/dbcstring.json` with the following content:

```
{
"com.oracle.db.network.scanname":"oda1-scan",
"com.oracle.db.network.port":"1521",                                       ←(optional, hidden:false)
"com.oracle.db.network.servicename":"ORCL"
}
```

To run the dbcstring configure script with dbcstring.json as input:

```
# /etc/template.d/scripts/dbcstring configure < /tmp/dbcstring.json
```

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

ORACLE

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Oracle Database Appliance: Automated Virtual Machine Provisioning
January 2017
Author: Ruggero Citton
Contributing Authors: RACPack Team