Documentation applicable to VM image atg11_1p1_v14

Overview

This software will assist in the installation and configuration of the Oracle ATG Commerce stack.
It will also assist in the provisioning, and automation of setting up instances in the Oracle Cloud with the Oracle ATG Commerce stack.

The code is broken into 3 distinct features, which can be used together to create an automated provision, install and configure of the Oracle ATG Commerce Stack in the cloud. The pieces can also be used independently if desired; for example – if you want to quickly setup a local machine/VM with the Oracle ATG Commerce Stack.

The provisioning process installs and configures the following products:

- ATG 11.1
    - ATG patches
- Endeca 11.1
- WebLogic 12.1.2
    - WebLogic Patches
- Oracle RDBMS 11.2.0.4
- Java 1.7.0_80
- Oracle Traffic Director 11.1.1.9

The installation process works by utilizing configuration data in JSON format.

When the VM is started the first time, the opc-init script calls the custom provisioning scripts inside the VM image. These scripts read the JSON data and perform the installations and configurations defined in that data.

The installation process is started by calling opt/oracle/install/11.1/pywrapper.sh

This is the script you must call, and pass configuration flags to from your provisioning process.

You call the script using the opc-init pre-bootstrap and script parameters in OPC.

```
{
   "pre-bootstrap": {
      "script": "/opt/oracle/install/11.1/pywrapper.sh  <config options here>"
   }
}
```

In OpenStack, you pass JSON as userdata. This is picked up by a process similar to opc-init.

```
{ "script": "/opt/oracle/install/11.1/pywrapper.sh  <config options here>"}
```

The different is no pre-bootstrap top level field.

Feature overview

- Install ATG
    - Add environment variables to users .bashrc
- Optionally apply ATG 11.1 patch 1
- Install Java
    - Add environment variables to users .bashrc
- Install Endeca
    - Install either all Endeca components, or just an MDEX server
    - Add environment variables to users .bashrc
- Install and configure Oracle Traffic Director
    - Create start/stop scripts
    - Configures server pools and virtual servers
- Install and configure Weblogic
    - Create start/stop scripts
    - Create managed servers
    - Create instances
        - Create and base configure ATG server layers for each instance
    - Set host and port binding for instances
    - Create and bind datasources
    - Change domain settings
- Optionally apply Weblogic patches
- Install database, and create sample database
    - Create start/stop scripts
- Generate configuration JSON
- Generate Ansible playbooks
- Generate Oracle Public Cloud orchestrations

Prerequisites

Use of this code outside an Oracle provided VM image requires the end user to make sure certain prerequisites have been met.

The following are required to use this code:

- Python 2.6 or newer for all tools except Ansible OpenStack extensions.
- Python 2.7 or newer if using Ansible OpenStack extensions.
- If Ansible is to be used for provisioning:
    - Ansible
    - OpenStack Ansible Python extensions
    - Oracle Compute Cloud Ansible Python Extensions
- OPC-INIT to provision VM's at boot time in OPC
- If provisioning VM's at boot time with OpenStack, you must add the provided opc-solaris-init script to your VM's boot execution sequence
- Product binaries

Operating system setup

If you are not using an image provided by Oracle, configured for this provisioning process, you will need to properly configure your own operating system prior to using this setup process.

This process is designed to work with Solaris SPARC and Oracle Linux x86. It may work with other operating systems, but has not been tested with them.

OS packages

Optionally on Oracle Linux:

yum install oracle-rdbms-server-11gR2-preinstall

This will change some OS tuning settings, and create the oracle user and oinstall group.
You can also do this manually.

You may need to install additional packages to use Ansible. What packages you need are beyond the scope of this document as they will vary based on how your OS was installed/configured.

OS user setup

- Create group oinstall if you didn't use the preinstall package.
- Create user oracle if you didn't use the preinstall package.
- Create user opc if you plan to create a private image for Oracle Public Cloud
- Create the wlinstall user.
- Create a ssh key pair for the wlinstall user
    - Add the pub key generated to the wlinstall users authorized_keys file

opc-init and opc-solaris-init

If you are creating a private image for the Oracle Cloud, and wish to automatically use this provisioning setup, you must install libraries that allow the VM to talk to the meta-data service at boot time.

opc-init

For opc-init, used with Oracle Public Cloud, follow the instructions here:

https://docs.oracle.com/cloud/latest/stcomputecs/STCSG/toc.htm

Refer to the section in using opc-init.

opc-solaris-init

For opc-solaris-init, which is for OpenStack/Solaris images, add the provided shell script to your boot images boot process.

In the openstack-metadata folder is a script called opc-solaris-init. Make sure this has execute permissions. Then add this script to your Solaris startup process. This can be accomplished multiple ways, you decide which best meets your needs. Examples:

- Copy opc-solaris-init to /etc/init.d
    - Create start/stop links in /etc/rc3.d pointing to the /etc/init.d/opc-solaris-init
    - OR Create a Solaris SMF service that calls opc-solaris-init

Edit opc-solaris-init and set the python interpreter to match your environment.

Product binaries

Product binaries are not included with this code, unless you have an Oracle provided VM image.

You will need to place the correct product binaries in the correct location for this code to execute correctly. You also must use the exact versions of each product described in this document, and the exact binary name listed below. If you want to try using other versions of products, you will need to modify the python code that installs/configured each product.

Product binaries must be in /opt/oracle/install/11.1/binaries

Under binaries, a subfolder for each product with its binaries should exist as follows:

For x86

binaries/atg11.1/linux/OCPlatform11.1.bin
binaries/atg11.1/patches/<any patches you want to install>
binaries/atg11.1/crs/OCReferenceStore11.1.bin
binaries/endeca11.1/Platform_Install/OCplatformservices11.1.0-Linux64.bin
binaries/endeca11.1/ToolsAndFrameworkInstall/Disk1
binaries/endeca11.1/ToolsAndFrameworkInstall/Translations
binaries/endeca11.1/CAS_Install/OCcas11.1.0-Linux64.sh
binaries/endeca11.1/MDEX_Install/OCmdex6.5.1-Linux64_829811.sh
binaries/OTD/Disk1
binaries/oracleRDBMS/database
binaries/java1.7/jdk-7u80-linux-x64.tar.gz
binaries/wls-12.1.2/patches/p22505331_121200_Generic.zip
binaries/wls-12.1.2/wls_121200.jar

For SPARC

binaries/atg11.1/linux/OCPlatform11.1.bin
binaries/atg11.1/patches/<any patches you want to install>
binaries/atg11.1/crs/OCReferenceStore11.1.bin
binaries/wls-12.1.2/patches/p22505331_121200_Generic.zip
binaries/wls-12.1.2/wls_121200.jar
binaries/endeca11.1/solaris/MDEX_Install/OCmdex6.5.1-Solaris_829811.sh
binaries/endeca11.1/solaris/Platform_Install/OCplatformservices11.1.0-Solaris.bin
binaries/endeca11.1/solaris/CAS_Install/OCcas11.1.0-Solaris.sh
binaries/endeca11.1/solaris/ToolsAndFrameworkInstall/Disk1
binaries/java1.7/solaris/jdk-7u80-solaris-sparcv9.tar.gz (NOTE – 32 and 64 bit installs are separate files)
binaries/java1.7/solaris/jdk-7u80-solaris-sparc.tar.gz (NOTE – 32 and 64 bit installs are separate files)
binaries/OTD/solaris/Disk1
binaries/oracleRDBMS/solaris/database

Make sure all .bin and .sh files have execute permissions

For the endeca11.1/ToolsAndFrameworkInstall tree, this is the 11.1 Tools and Framework installer unzipped. Drop the zip archive in this tree, and unzip.

For the OTD tree, this is the Oracle Traffic Director 11.1.1.9 installer unzipped. Drop the zip archive in this tree, and unzip.

For the oracleRDBMS tree, this is the Oracle database 11g (11.2.0.4) installer unzipped. Drop the zip archive in this tree, and unzip.

Python setup

The following python libraries are required

- PyCrypto, abi, ansible
- at least python 2.6

If using OpenStack, you must also have

- shade, importlib
- at least python 2.7

Install python OC wrappers

- cd to the python-oc directory
- tar -xvf python-oc.tar
- cd oc
- python setup.py install
    - make sure you are using the correct version of python. For example, if you manually installed a different version of python, you should have a different executable: i.e. python2.7

Adding the setup process

This provisioning code must exist in /opt/oracle/install/11.1

Copy the code/directory structure exactly as it was provided.

The tree oracle/install/11.1 and all files under it should be owned by oracle:oinstall

Edit commerce_setup.py and set the python interpreter to match your environment.

Script Commands

The following are possible arguments that can be passed to pywrapper.sh

--configSource=	Specify the source of the JSON configuration data
--java	Install java. Add entries to users .bashrc
--weblogic	Install Weblogic. Add entries to users .bashrc
--weblogicDomain	Creates a new Weblogic Domain. Optionally creates managed servers and machines. Creates start/stop scripts for the Weblogic Admin instance and nodemanager. Optionally adds startup scripts to the VM boot process.
--weblogicMachines	Called by –weblogicDomain. Creates machines in the domain if appropriate JSON exists. Do not call directly.
--weblogicServers	Called by –weblogicDomain. Creates server instances in the domain if appropriate JSON exists. Do not call directly.
--weblogicSettings	Wrapper to enable domain settings. Currently sets JTA timeout.
--weblogicDatasources	Create WebLogic datasources
--weblogicManagedServer	Used with –weblogic. Tells the installer to get domain information from the admin server, join the domain and add this box as a managed server. Creates start/stop script for the nodemanager. Optionally adds startup script to the VM boot process.
--atg	Install ATG 11.1
--atgpatch	Install ATG patches
--endeca	Installs all Endeca components (MDEX, PlatformServices, ToolsAndFramework, CAS). Creates start/stop scripts for PlatformServices, ToolsAndFramework, CAS. Optionally adds startup scripts to the VM boot process. Add entries to users .bashrc
--dgraph	Installs only MDEX and PlatformServices Endeca components. Meant for adding an additional dgraph server. Creates start/stop script for PlatformServices. Optionally adds startup script to the VM boot process. Add entries to users .bashrc
--otdInstall	Install Oracle Traffic Director. Create start/stop scripts for admin server. Optionally adds startup script to the VM boot process.
--otdConfig	Configure OTD. Create server pools and virtual servers.
--db	Install Oracle database. Create start/stop scripts for database an dbconsole. Optionally adds startup scripts to the VM boot process. Add entries to users .bashrc
--copy-ssh-keys	Copy ssh authorized_keys from one user to another. The default provisioning process only gives keys to the OPC user. Useful if, for example, you want to login as the Oracle user directly.
--addstorage	Format and attach block storage to a single mount point. Disk is formatted as ext4, and added to /etc/fstab
--advancedStorage	Format and attach multiple block storage to multiple mount points. Disks are formatted as ext4, and added to /etc/fstab
--openstack	If an instance is being created with user-data on openstack, this flag is required.
--debug	Additional data is output to the install log

--configSource=

This tells the installation process where to read JSON data from.

Possible data sources are:

- Default JSON in the VM image
  - Used by default is no other source is specified. Default JSON configuration data embedded in the VM image will be used.
- File based JSON data
  - Pass file:<filename> to read from file system.
    - Example –configSource=file:defaultConfig.json
- External URL
  - External URL's must contain the http:// prefix, and should point directly to a json response.
    - Example: http://192.168.1.100/testData.json
- JSON added to the custom user attributes field during provisioning. commerceSetup must be the top level property. Example:

```
{
   "commerceSetup":
     {
        "JAVA_install":
          {
             "javaHome": "/usr/java",
             "installOwner": "oracle",
             "installGroup": "oinstall"
          }
     }
}
```

Do not pass configDatasource option	Use default JSON in VM image
--configDatasource=file:<filename>	Read <filename> from the filesystem. Relative path to base script location.
--configDatasource=user-data	Use user data added to custom user attributes
--configDatasource=http://<url>	Get JSON data from external source

All JSON data – formatting

All JSON data must be well formatted, and must be nested under the top level commerceSetup property. Any JSON outside of the commerceSetup property will be ignored by this installation process.

It is recommended you test your JSON for proper formatting prior to use.

All start/stop scripts

All start/stop scripts are installed to /etc/init.d

--java

Install java. Add JAVA_HOME to installOwner .bashrc, and add JAVA_HOME/bin directory to path

JSON options are:

Field name	Field Description	Required/Optional
javaHome	Installation path for java. This will be your JAVA_HOME area.	Required
installOwner	Linux user that will own installation	Required
installGroup	Linux group that will own installation	Required

```
"JAVA_install":
  {
    "javaHome": "/usr/java",
    "installOwner": "oracle",
    "installGroup": "oinstall"
  }
}
```

--weblogic

Install WebLogic. This only installs WebLogic. No domain creation or configuration is performed here.

Adds MW_HOME to installOwner .bashrc

Note – all WebLogic commands share the WEBLOGIC_common JSON data

JSON options are:

Field name	Field Description	Required/Optional
middlewareHome	Installation path for weblogic. This will be your MW_HOME area.	Required
installOwner	Linux user that will own installation	Required
installGroup	Linux group that will own installation	Required
oraInventoryDir	Oracle inventory directory	Required
wl_adminUser	WebLogic admin user name	Required
wl_adminPassword	WebLogic admin user password	Required
wl_adminHost	The host the WebLogic admin instance is running on	Required
wl_adminHttpPort	The http Port the WebLogic admin instance is listening on	Required
wl_adminHttpsPort	The http Port the WebLogic admin instance is listening on	Required
wl_domain	The name of the WebLogic domain to be used	Required
wl_patches	Comma separated list of WebLogic patch zip files. Files should be in the exact format as provided by Oracle.	Optional

```
"WEBLOGIC_common":
            {
               "middlewareHome": "/u01/middleware",
               "installOwner": "oracle",
               "installGroup": "oinstall",
               "oraInventoryDir": "/u01/oraInventory",
               "wl_adminUser": "weblogic",
               "wl_adminPassword": "password1",
               "wl_adminHost": "atgsupport",
               "wl_adminHttpPort": "7001",
               "wl_adminHttpsPort": "7002",
               "wl_domain": "base_domain",
              "wl_patches": "p22505331_121200_Generic.zip"
            }
```

--weblogicDomain

Creates a new WebLogic domain. Creates start/stop scripts for the Admin server and nodemanager. Optionally adds the admin server and nodemanager to the VM boot process. Packs the newly created domain for distribution to managed servers added later. Use in conjunction with –weblogic as the product must be installed before a domain can be created.

When the –weblogicDomain flag is passed in, the installation process looks for the WEBLOGIC_domain_setup JSON property. If found, it also looks for WEBLOGIC_common JSON data. Both are required for this option to work.

Optionally, this command will create WebLogic machines and servers. If JSON data for WEBLOGIC_machines and WEBLOGIC_managed_servers is found, it will use that data. See details of –weblogicMachines and –weblogicServers below for JSON structure details

JSON options are:

Field name	Field Description	Required/Optional
wl_startAdmin_onBoot	Can be true or false. Determines if the admin server should start on VM boot	Required
wl_startNodemgr_onBoot	Can be true or false. Determines if the node manager should start on VM boot	Required

```
"WEBLOGIC_domain_setup":
        {
                "wl_startAdmin_onBoot": "true",
                "wl_startNodemgr_onBoot": "true"
        }
```

--weblogicMachines

Used with –weblogicDomain to create WebLogic machines in the domain. This JSON field is optional. If it is present, it will be used.

JSON is an array of machines to create

JSON options are:

Field name	Field Description	Required/Optional
machineName	Name of machine to create. This is any name you want to refer to this machine by.	Required
machineAddress	The hostname or IP of the machine	Required

```
"WEBLOGIC_machines":
  [
  {
    "machineName": "atgsupport",
    "machineAddress": "atgsupport"
  },
  {
    "machineName": "atg1",
    "machineAddress": "atg1"
  },
  {
    "machineName": "atg2",
    "machineAddress": "atg2"
  }
  ]
```

--weblogicServers

Used with –weblogicDomain to create WebLogic server instances in the domain. This JSON field is optional. If it is present, it will be used.

JSON is an array of instances to create

WARNING – The WebLogic admin server will attempt to communicate with any instance you define here. If you specify hosts that do not exist, or do not have a node manager running on them, the WebLogic admin console will be slow.

JSON options are:

Field name	Field Description	Required/Optional
managedServerName	Name of instance to create. This is any name you want to refer to this instance by.	Required
managedServerHttpPort	HTTP port this instance will listen on.	Required
managedServerHttpsPort	HTTPS port this instance will listen on.	Required
managedServerHost	The hostname or IP the instance will run on.	Required
atgRmiPort	ATG RMI port for this instance	
atgFdPort	ATG File Distribution port for this instance (For BCC deploys)	Required
atgDrpPort	ATG DRP port for this instance. Note – DRP port is not used for active communication, but is required internally by the app. You must still use a unique port.	Required
atgServerType	ATG Server type. [page, lock, bcc, pes] page – normal page serving instance lock – will act as lock server bcc – will be the bcc instance pes – will be the process editor	Required Use all lower case
atgLockManPort	Port a lock server will bind to. Required if atgServerType=lock	Conditionally required
bccFileSyncPort	Port the BCC will use for file sync. Required if atgServerType=bcc	Conditionally required
bccLockPort	Port the BCC lock server will bind to. Required if atgServerType=bcc	Conditionally required

```
"WEBLOGIC_managed_servers":
    [
            {
               "managedServerName": "Prod2I2",
               "managedServerHttpPort": "7120",
               "managedServerHttpsPort": "7011",
               "atgRmiPort": "7120",
               "atgFdPort": "7210",
               "atgDrpPort": "7310",
               "atgServerType": "page",
               "managedServerHost": "atg2"
            },
            {
               "managedServerName": "BCC",
               "managedServerHttpPort": "7310",
               "managedServerHttpsPort": "7011",
               "atgRmiPort": "7110",
               "atgFdPort": "7210",
               "atgDrpPort": "7310",
               "atgServerType": "bcc",
               "bccFileSyncPort": "8816",
               "bccLockPort": "9010",
               "managedServerHost": "atgsupport"
            },
            {
               "managedServerName": "Lock1",
               "managedServerHttpPort": "7320",
               "managedServerHttpsPort": "7011",
               "atgRmiPort": "7120",
               "atgFdPort": "7210",
               "atgDrpPort": "7310",
               "atgServerType": "lock",
               "atgLockManPort": "9012",
               "managedServerHost": "atgsupport"
            }
    ]
```

--weblogicSettings

Wrapper for setting global WebLogic domains settings. Currently sets JTA timeout

JSON options are:

Field name	Field Description	Required/Optional
jtaTimeout	Value, in seconds, for JTA timeout	Required

```
"WEBLOGIC_domain_settings":
  {
    "jtaTimeout": "500"
  }
```

--weblogicDatasources

Create WebLogic domain datasources.
JSON data is an array of datasources to create.
JSON options are:

Field name	Field Description	Required/Optional
dsName	Name of datsource as shown in WebLogic admin console	Required
dsJNDIName	JNDI name of datasource	Required
dsURL	JDBC URL/Connect string for datasource	Required
dsDriver	Datasource driver to use	Required
dsUsername	Username for the schema this ds connects to	Required
dsPassword	Password for the schema this ds connects to	Required
dsTargetNames	Comma separated list of WebLogic servers that will be targets for this datasource.	Required
dsMaxCapacity	Maximum capacity for this datasources connection pool	Required

dsTargetNames Note – what you list in dsTargetNames MUST exist in the domain, or the datasource creation will throw errors. This can be used in conjunction with creating a domain during VM provisioning. weblogicServers will be created first, before attempting to target them to datasources. If you use this in conjunction with weblogicServers, what you list here must match server names you created there.

```
"WEBLOGIC_datasources":
   [
   {
     "dsName": "CoreDatasource",
     "dsJNDIName": "atgcore",
     "dsURL": "jdbc:oracle:thin:@localhost:1521:orcl",
     "dsDriver": "oracle.jdbc.xa.client.OracleXADataSource",
     "dsUsername": "atgcore",
     "dsPassword": "corePassword",
     "dsTargetNames": "atg1",
     "dsMaxCapacity": "50"
   },
   {
     "dsName": "CatalogA",
     "dsJNDIName": "atgcata",
     "dsURL": "jdbc:oracle:thin:@localhost:1521:orcl",
     "dsDriver": "oracle.jdbc.xa.client.OracleXADataSource",
     "dsUsername": "atgcata",
     "dsPassword": "cataPassword",
     "dsTargetNames": "atg1,atg2",
     "dsMaxCapacity": "30"
   }
   ]
```

--weblogicManagedServer

Retrieves domain information from the WebLogic admin instance and sets server up as a managed server. Use in conjunction with –weblogic as the product must be installed before a domain can be retrieved.

Creates start/stop script for node manager.

This command is dependent on the WebLogic domain and admin server having been successfully created. This command will try for up to 15 minutes from execution to contact the admin server and retrieve the packed domain data. Communication to the admin server is via ssh.

When the –weblogicManagedServer flag is passed in, the installation process looks for the WEBLOGIC_managed_server JSON property. If found, it also looks for WEBLOGIC_common JSON data. Both are required for this option to work.

JSON options are:

Field name	Field Description	Required/Optional
wl_startNodemgr_onBoot	Can be true or false. Determines if the node manager should start on VM boot	Required

```
"WEBLOGIC_managed_server":
  {
    "wl_startNodemgr_onBoot": "true"
  }
```

--atg

Install ATG 11.1. Adds DYNAMO_ROOT and DYNAMO_HOME to installOwner .bashrc

JSON options are:

Field name	Field Description	Required/Optional
dynamoRoot	Installation path for ATG. This will be your DYNAMO_ROOT area.	Required
installOwner	Linux user that will own installation	Required
installGroup	Linux group that will own installation	Required
rmiPort	Default ATG RMI port	Required
javaHome	Path to JAVA_HOME	Required
wl_home	Path to WebLogic/MW_HOME	Required
wl_domain	WebLogic domain name	Required
wl_adminPort	WebLogic Admin server HTTP port	Required
install_crs	Install the Commerce Reference Store – true/false	Required
install_service	Install ATG Service – true/false	Required
atg_clustername	Used in ClusterName.properties. Logical name of this cluster.	Required

```
"ATG_install":
        {
                "dynamoRoot": "/u02/atg11.1",
                "installOwner": "oracle",
                "installGroup": "oinstall",
                "rmiPort": "8860",
                "javaHome": "/usr/java/latest",
                "wl_home": "/u01/middleware",
                "wl_domain": "base_domain",
                "wl_adminPort": "7001",
                "install_crs": "true",
                "install_service": "true",
                "atg_clustername": "democluster"
        }
```

--atgpatch

Install ATG patches

JSON options are:

Field name	Field Description	Required/Optional
dynamoRoot	Patch to ATG installation	Required
installOwner	User that owns the ATG installation	Required
atg_patch_archive	Name of the patch zip file you want to install	Required
atg_patch_destination	The name of the directory the zip creates. This is not what you want to call it, but what is embedded in the product patch.	Required

```
"ATGPATCH_install":
        {
                "dynamoRoot": "/u02/atg11.1",
                "installOwner": "oracle",
                 "atg_patch_archive": "p20173321_111000_Generic.zip",
                 "atg_patch_destination": "OCPlatform11.1_p1"
        }
```

--endeca

Install all Endeca components (MDEX, PlatformServices, ToolsAndFramework, CAS)

Top level fields required for all endeca components

Field name	Field Description	Required/Optional
installOwner	Linux user that will own installation	Required
installGroup	Linux group that will own installation	Required

```
"ENDECA_install": {
   "installOwner": "oracle",
   "installGroup": "oinstall",
   }
```

mdex

Installs Endeca MDEX component. Adds ENDECA_HOME and sourcing of MDEX INI data to installOwner .bashrc

JSON options are:

Field name	Field Description	Required/Optional
endecaRoot	Base path to Endeca installation	Required

```
"ENDECA_install": {
        "installOwner": "oracle",
        "installGroup": "oinstall",
        "mdex": {
                "endecaRoot": "/u01/oracle"
        }
}
```

platformServices

Install Endeca PlatformServices component. Adds sourcing of INI data to installOwner .bashrc

Creates start/stop script for service.

JSON options are:

Field name	Field Description	Required/Optional
endecaRoot	Base path to Endeca installation	Required
mdexRoot	Path to MDEX installataion, including version	Required
eacPort	EAC Port	Required
eacShutdownPort	EAC Shutdown Port	Required
installEACAgent	Install EAC Agent – Y or N	Required
installReferenceApps	Install Reference Apps – Y or N	Required
start_onBoot	Can be true or false. Determines service should start on VM boot	Required

```
"ENDECA_install": {
        "installOwner": "oracle",
        "installGroup": "oinstall",
        "platformServices": {
                "endecaRoot": "/u01/oracle",
                "mdexRoot": "/u01/oracle/endeca/MDEX/6.5.1",
                "eacPort": "8888",
                "eacShutdownPort": "8090",
                "installEACAgent": "y",
                "installReferenceApps": "y",
                "start_onBoot": "true"
        }
}
```

toolsAndFramework

Install Endeca PlatformServices component. Creates start/stop script for service.

JSON options are:

Field name	Field Description	Required/Optional
endecaRoot	Base path to Endeca installation	Required
adminPassword	Workbench admin password	Required
start_onBoot	Can be true or false. Determines service should start on VM boot	Required

```
"ENDECA_install": {
        "installOwner": "oracle",
        "installGroup": "oinstall",
        "toolsAndFramework": {
                "endecaRoot": "/u01/oracle",
                "adminPassword": "admin",
                "start_onBoot": "true"
        }
}
```

cas

Install Endeca CAS component. Creates start/stop script for service.

JSON options are:

Field name	Field Description	Required/Optional
endecaRoot	Base path to Endeca installation	Required
casPort	CAS Port	Required
casShutdownPort	CAS Shutdown Port	Required
casHostname	Hostname CAS is running on	Required
start_onBoot	Can be true or false. Determines service should start on VM boot	Required

```
"ENDECA_install": {
        "installOwner": "oracle",
        "installGroup": "oinstall",
        "cas": {
                "endecaRoot": "/u01/oracle",
                "casPort": "8500",
                "casShutdownPort": "8506",
                "casHostname": "localhost",
                "start_onBoot": "true"
        }
}
```

Example of complete Endeca JSON:

```
"ENDECA_install": {
        "installOwner": "oracle",
        "installGroup": "oinstall",
        "mdex": {
                "endecaRoot": "/u01/oracle"
        },
        "platformServices": {
                "endecaRoot": "/u01/oracle",
                "mdexRoot": "/u01/oracle/endeca/MDEX/6.5.1",
                "eacPort": "8888",
                "eacShutdownPort": "8090",
                "installEACAgent": "y",
                "installReferenceApps": "y",
                "start_onBoot": "true"
        },
        "toolsAndFramework": {
                "endecaRoot": "/u01/oracle",
                "adminPassword": "admin",
                "start_onBoot": "true"
        },
        "cas": {
                "endecaRoot": "/u01/oracle",
                "casPort": "8500",
                "casShutdownPort": "8506",
                "casHostname": "localhost",
                "start_onBoot": "true"
        }
}
```

--dgraph

Installs only the MDEX and PlatformServices Endeca components.

Adds INI sourcing for both, and creates start/stop script for PlatformServices. JSON data is as described above for each service.

--otdInstall

Installs Oracle Traffic Director. Creates start/stop script for OTD Admin console.

JSON options are:

Field name	Field Description	Required/Optional
installDir	Base path to installation	Required
installOwner	User that owns the installation	Required
adminUser	Name of the admin user	Required
adminPassword	Admin user password	Required
instanceHome	Path to install the admin service OTD instance in	Required
oraInventoryDir	Path to Oracle Inventory directory	Required
oraInventoryGroup	Group that owns oraInventory – will also own the install	Required
otd_startAdmin_onBoot	Can be true or false. Determines service should start on VM boot	Required

```
"OTD_install":
  {
    "installDir": "/u01/otd",
    "installOwner": "oracle",
    "adminUser": "otd_admin",
    "adminPassword": "password1",
    "instanceHome": "/u01/otd/production",
    "oraInventoryDir": "/u01/oraInventory",
    "oraInventoryGroup": "oinstall",
    "otd_startAdmin_onBoot": "true"
  }
```

--otdConfig

Configures OTD virtual servers. JSON is an array of config objects.

JSON options are:

Field name	Field Description	Required/Optional
configName	Name to give this config in OTD	
installDir	Base path to installation	Required
installOwner	User that owns the installation	Required
adminUser	Name of the admin user	Required
adminPassword	Admin user password	Required
virtualServerName	Name of virtual server group for these servers	Required
virtualServerPort	Port these virtual servers will receive requests on. This is the port an app would send requests to for them to be load balanced across your servers.	Required
originServers	Servers this instance should load balance across. Format is a comma separated list of <hostname>:<port>	Required
originPoolName	Origin pool name for these servers	Required
originServerType	Type of requests this server handles [HTTP, HTTPS, TCP]	Required
loadDistribution	Load balancing algorithm to use. Values are round-robin, least-response-time and least-connection-count	Required
instanceHostname	Hostname this node will run on	Required

```
"OTD_config":
   [
   {
     "configName": "atg-servers",
     "installDir": "/u01/otd",
     "installOwner": "oracle",
     "adminUser": "otd_admin",
     "adminPassword": "password1",
     "virtualServerName": "atg-servers",
     "virtualServerPort": "9000",
     "originServers": "atg1:7010,atg1:7020,atg2:7110,atg2:7120",
     "originPoolName": "atg-servers",
     "originServerType": "http",
     "loadDistribution": "round-robin",
     "instanceHostname": "localhost"
   },
   {
     "configName": "endeca-servers",
     "installDir": "/u01/otd".....
```

--db

Installs Oracle Database. Creates sample starter database. Creates start/stop scripts for database and dbconsole. Adds ORACLE_HOME to installOwner .bashrc, and adds install bin dir to users path.

JSON options are:

Field name	Field Description	Required/Optional
oracleBase	Base path to installation	Required
installOwner	User that owns the installation	Required
installGroup	Group that owns the installation	Required
oraInventoryDir	Path to Oracle Inventory directory	Required
installHost	Hostname db is being installed on. Used for IP binding. Localhost should bind to all IP's on the VM.	Required
oracleHome	Path to ORACLE_HOME	Required
oracleSID	SID of starter DB	Required
adminPW	Admin password to starter DB	Required
dbStorageLoc	Path to database storage location	Required
db_onBoot	Can be true or false. Determines service should start on VM boot	Required

```
"ORACLE_RDBMS_install":
  {
    "oracleBase": "/u01/oracle",
    "installOwner": "oracle",
    "installGroup": "oinstall",
    "oraInventoryDir": "/u01/oraInventory",
    "installHost": "localhost",
    "oracleHome": "/u01/oracle/product/11.2.0/dbhome_1",
    "oracleSID": "orcl",
    "adminPW": "password1",
    "dbStorageLoc": "/u01/oracle/oradata",
    "db_onBoot": "true"
  }
```

--copy-ssh-keys

Copy authorized_keys file from one user to another.

At provision time, ssh keys are given to the OPC user. If, for example, you want to be able to ssh to the VM directly as the Oracle user, this command will automatically copy the authorized_keys from the OPC user to the Oracle user.

JSON options are:

Field name	Field Description	Required/Optional
fromUser	User to copy keys from	Required
toUser	User to copy keys to	Required
toUserGroup	Unix group of the toUser	Required

```
"copy_ssh_keys":
  {
    "fromUser": "opc",
    "toUser": "oracle",
    "toUserGroup": "oinstall"
  }
```

--addstorage

Format and mount storage to a single mount point. Disk is formatted as ext4, mount point is created, disk is mounted and added to /etc/fstab

This command will format and mount your index 1 block storage device. This command is useful if you are only attaching one piece of block storage to your VM

This command is only compatible with OPC provisioning.

JSON options are:

Field name	Field Description	Required/Optional
mountPoint	Mount point for disk	Required
mountOwner	Owner of mount point	Required
mountGroup	Unix group of mount point	Required

```
"mount_storage":
  {
    "mountPoint": "/u01",
    "mountOwner": "oracle",
    "mountGroup": "oinstall"
  }
```

--advancedstorage

Format and mount storage for multiple block storage devices. Disks are formatted as ext4, mount points are created, disks are mounted and added to /etc/fstab

This command is only compatible with OPC provisioning.

WARNING – Mistakes with this command will possibly make your VM unusable.

JSON data is an array of disks to create and mount

JSON options are:

Field name	Field Description	Required/Optional
device	Physical device to format and mount. Refer to the Oracle Cloud documentation on how devices are setup based on the index number of your block storage.	Required
mountPoint	Mount point for disk	Required
mountOwner	Owner of mount point	Required
mountGroup	Unix group of mount point	Required

```
"advanced_storage":
[
{
    "device": "/dev/xvdb",
    "mountPoint": "/u01",
    "mountOwner": "oracle",
    "mountGroup": "oinstall"
},
{
    "device": "/dev/xvdc",
    "mountPoint": "/u02",
    "mountOwner": "oracle",
    "mountGroup": "oinstall"
}
]
```

--openstack

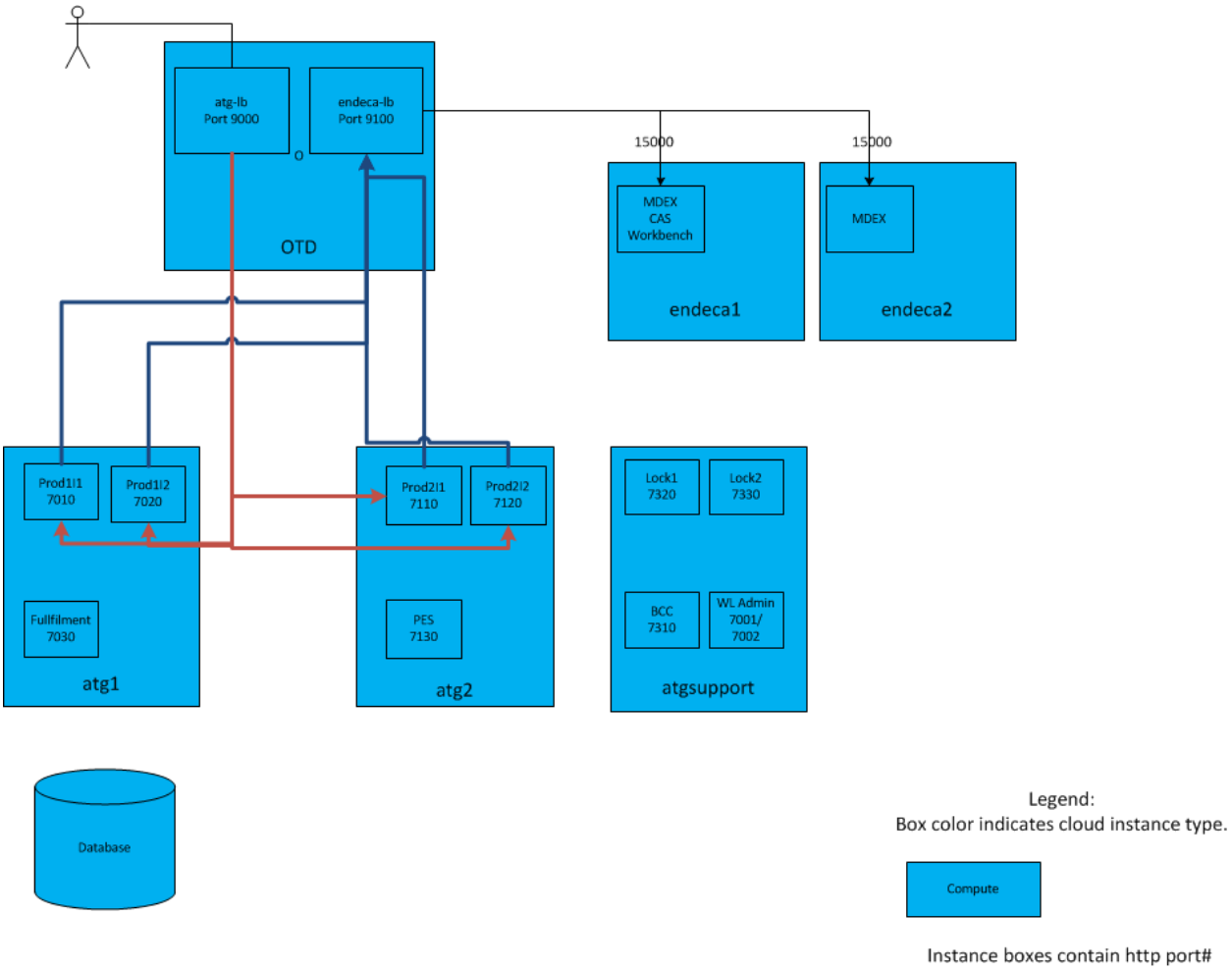
If an instance is being created against OpenStack, and is using user-data for its JSON source, this flag is required. Passing this flag lets the installer know to pull user meta-data from a different URL, and in a different way.

Complete JSON of default setup

A complete JSON file is included at defaultJson/defaultConfig.json

Diagram of default config layout

The following is a high level diagram of the setup the default VM JSON can create.



Sample commands

Invoking setup process through opc-init

The installation process is invoked by calling pywarpper.sh. This is done with the script tag under pre-bootstrap in the custom user attributes section of the provisioning process. This can be done either in the web GUI, or in orchestration scripts.

You specific the flags you want the install process to pickup as arguments to the pywrapper.sh script.

Note that the order of the flags is not important. The installation process will execute them in the correct order.

```
{
   "pre-bootstrap": {
      "script": "/opt/oracle/install/11.1/pywrapper.sh --java --otdInstall --otdConfig --copy-ssh-keys"
   }
}
```

Invoking setup process through opc-solaris-init with OpenStack

Pass in JSON as userdata to your OpenStack instance. Just the script field is required.

Example:

```
{"script": "/opt/oracle/install/11.1/pywrapper.sh --java --otdInstall --otdConfig --copy-ssh-keys"}
```

Logging

A log of the installation process is created at /opt/oracle/install/11.1/opc-installer.log

Installing WebLogic and creating a domain

```
{
   "pre-bootstrap": {
      "script": "/opt/oracle/install/11.1/pywrapper.sh --java –weblogic --weblogicDomain"
   }
}
```

WebLogic requires Java to be installed. This will first install java, then install WebLogic, then create the WebLogic domain.
Data will be read from the default JSON in the VM.

Install OTD and configure OTD

```
{
   "pre-bootstrap": {
      "script": "opt/oracle/install/11.1/pywrapper.sh  --otdInstall --otdConfig"
   }
}
```

This will first install OTD, creating an admin instance. It will then create virtual servers that load balance the servers specified in the JSON config data.

Data will be read from the default JSON in the VM.

Install Oracle DB using passed in user meta data

```
{
   "pre-bootstrap": {
      "script": "opt/oracle/install/11.1/pywrapper.sh  --db --configSource=user-data"
   },

   "commerceSetup":
   {
          "ORACLE_RDBMS_install":
           {
              "oracleBase": "/u01/oracle",
              "installOwner": "oracle",
              "installGroup": "oinstall",
              "oraInventoryDir": "/u01/oraInventory",
              "installHost": "localhost",
              "oracleHome": "/u01/oracle/product/11.2.0/dbhome_1",
              "oracleSID": "orcl",
              "adminPW": "password1",
              "dbStorageLoc": "/u01/oracle/oradata",
              "db_onBoot": "true"
           }
   }
}
```

This will install an Oracle database. This will not use the default JSON in the VM. By passing the –configSource=user-data flag, the JSON specified below the pre-bootstrap field will be used in OPC. In OpenStack, the process is the same, but there is no pre-bootstrap field.

Install Java using external JSON data

```
{
   "pre-bootstrap": {
      "script": "opt/oracle/install/11.1/pywrapper.sh  --java --configSource=http://192.168.23.123/javaConfig.json"
   }
}
```

This tells the installation process to install Java, and to retrieve the JSON configuration data from an external URL.

Generating provisioning files

A web base GUI is provided to assist in the generation of configuration data for provisioning, and generation of Ansible playbooks.

Instructions to install the tool are included in the webui directory.

Generating orchestrations and Ansible config files.

After creating storage and instances, you can generate orchestration files, JSON config data, and Ansible config scripts based on your storage and instances.

The generate configs option in the GUI will create your orchestrations and playbooks.
Files will be saves under the project folder, in the orchestrations directory.
The layout of the directories under the orchestration folder is as follows:

- ansible – ansible config files and scripts
    - Instance playbooks will be for OPC or OpenStack, based on your choice during configuration. OpenStack instances will have _os_ in the yaml filename.
- instances – OPC orchestrations for instances
- json – JSON data files for the provisioning scripts – this is the data pywrapper would use as a configSource. You must get this data to the install process through the configSource parameter.
- secapp – OPC security application orchestrations
- seclist - OPC security list orchestrations
- storage – OPC orchestrations for storage

The storage and instance orchestrations or JSON files meant to be uploaded to your identity domain. This can be used to provision your storage and instances.

You can also provision storage and instances using Ansible. It is recommended you use Ansible or Orchestrations, not both mixed together.

Running Ansible scripts

1. cd to the ansible folder under your project after you have generated the config files.
    a. Example: projects/test/orchestrations/ansible
2. Ansible playbooks are stored in the playbooks folder.
3. To execute a playbook, you must first export some environment variables. Note – without these set properly, ansible playbooks will not execute properly. You must export the correct settings to communicate with OPC and/or OpenStack, depending in your target environment choices.
4. Example for OPC:

```
export OC_ENDPOINT="https://api-z28.compute.us6.oraclecloud.com/"
export OC_DOMAIN="mydomain"
export OC_USERNAME="john.doe@domain.com"
export OC_PASSWORD=”PASSWORD123”
export OC_SSHKEY_FILE="/home/oracle/.ssh/id_rsa.pub"
export ANSIBLE_INVENTORY=$PWD/ansible_hosts
```

    a. OC_ENDPOINT is the REST endpoint for your identity domain. You can find this value in the Compute console for your domain
    b. OC_DOMAIN is your identity domain name
    c. OC_USERNAME is your username in the identity domain
    d. OC_PASSWORD is your password for the identity domain
    e. OC_SSHKEY_FILE is the path to your public ssh key. This must be the same key you have configured in the identity domain.
    f. ANSIBLE_INVENTORY should be left as is. It tells Ansible to use ansible_hosts in the current directory.
5. Example for OpenStack:

```
export OS_USERNAME=admin
export OS_PASSWORD=PASSWORD123
export OS_TENANT_NAME=demo
export OS_AUTH_URL=http://192.168.1.10:35357/v2.0
```

    a. OS_AUTH_URL is the REST endpoint for your OpenStack auth service.
    b. OS_TENANT_NAME is your OpenStack project name
    c. OS_USERNAME is your username in your OpenStack project
    d. OS_PASSWORD is your password in your OpenStack project
6. Executing playbooks is done with the ansible-playbooks command.
    a. Example:

```
ansible-playbook playbooks/atgdb_storage.yaml
```

    b. The creation YAML will be in the format [name]_[type].yaml
    c. The YAML to delete what you create is in the format [name]_[type]_cleanup.yaml

d. Executing “ansible-playbook playbooks/atgdb_storage.yaml” will create the storage defined by atgdb.
e. Executing “ansible-playbook playbooks/atgdb_storage_cleanup.yaml” will delete the storage defined by atgdb.

Note that playbooks are intentionally broken into many files. This allows you to run a single play instead of many at one time.

When running the initial setup, it is important to run plays in the correct order. For example, you cannot create an instance with storage attached until the storage object is created.

A set of shell scripts are generated along with the provisioning scripts. The scripts are located in <project>/orchestrations/ansible. These scripts will execute all generated plays in the correct order if you don’t want to execute each playbook manually. There is one script to create an environment, and another to delete an environment.