kindred

Automating Oracle database administration, practical examples with Ansible

Mikael Sandström / Ilmar Kerm

@oravirt

@ilmarkerm

Kindred in Numbers

All numbers from Q4 2017

GWR:
£238m

Ebitda:
£75m

Customers:
20m

Daily transactions:
16-18m

GWR from locally regulated markets:
42%

GWR from mobile:
70%

Employees:
1,400

Offices:
15

Nationalities:
52

Brands:
11

Our brands

A portfolio of brands within sportsbook, casino and games

Where we operate

Our Offices

Stockholm | Copenhagen | Tallinn | Antwerp | Sydney

London | Paris | Milan | Gibraltar | Malta | Darwin

Our Licenses

Australia | Belgium | Denmark | Estonia | France | Germany

Gibraltar | Ireland | Italy | Malta | Romania | United Kingdom

WHOAMI: Ilmar Kerm

- Senior database administrator in Kindred Group
- Working in IT since 2000
- Working with Oracle database since 2003
- Working at Kindred since 2007/2015
- From LAMP developer to Oracle DBA
- Oracle ACE

Blog: https://ilmarkerm.eu
Email: ilmar.kerm@kindredgroup.com
Twitter: @ilmarkerm

WHOAMI: Mikael Sandström

- Senior database administrator at Kindred Group
- Worked with Oracle for a long time
- Working at Kindred since early 2016
- Automation junkie

Blog: https://oravirt.wordpress.com
Email: mikael.sandstrom@kindredgroup.com
Twitter: @oravirt

Ansible for Oracle use cases

- Easy
  - Setting up database environment infrastructure, OS
  - Installation of RDBMS software
  - Automation of routine OS tasks
- Medium
  - Push of the button datacenter deployment
  - Initial creation of databases, pushing out initial configuration
  - Automation of routine DBA tasks: patching, creating tablespaces, …
  - One time tasks needing high level of orchestration – upgrades, …
  - Implementing missing features from Oracle
  - Enforcing common security settings
- Advanced
  - All database configuration and changes deployed as version controlled code
  - Easy to review and audit, repeatable

Agenda

- We expect that you are familiar with Ansible
- Things we cover
  - Setting up Ansible environment for Oracle DB use: inventory, modules, cxOracle
  - Handling Ansible inventory for database usage
  - Creating/managing users and roles, syncing with Active Directory
  - Services, tablespaces, resource manager, DBMS_SCHEDULER, AWR settings, INIT.ORA parameters, ASM diskgroups
  - User privileges, including grants to entire target schema
  - Gathering facts about databases
  - Managing passwords in playbooks
  - Practical playbook deployment: execution, scheduling, configuration
- Things we do not cover, because every other Ansible presentation does these
  - What is Ansible (check the first bullet point)
  - Managing OS
  - Installing Oracle RDBMS/Grid software
  - Creating databases/clusters (we will create PDBs though)

Introducing Oracle modules for Ansible

- Modules for administering Oracle database infrastructure
- Hosted in Github
  - `https://github.com/oravirt/ansible-oracle-modules`
- Currently not part of Ansible standard modules
  - Work in progress
- Written in Python
- Require cx_Oracle
  - Needed where the modules are executed
  - cx_Oracle needs an Oracle client
  - If run locally on ‘control-machine’ Instant Client is fine, otherwise just install on the database host
  - pip or yum/rpm

Introducing Oracle modules for Ansible

- Currently manages:
  - Users and roles, including synchronization from AD/LDAP
  - Grants: system and object privileges, including wildcards
  - Tablespaces
  - Parameters
  - Services
  - ASM diskgroups and volumes
  - Pluggable databases
  - DBMS_SCHEDULER jobs, windows and schedules
  - Resource manager consumer groups, including mappings by user profile
  - AWR snapshot settings
- Helper modules:
  - Gathering database facts
  - Executing SQL

Setting up – playbook structure

```
|____ansible.cfg
|____site.yml
|____build-host.yml
|____db-config.yml
|____group_vars
| |____doag
| | |____doag.yml
| | |____passwords.yml
|____library
| |____oracle_asmdg
| |____oracle_awr
| |____oracle_facts
| |____oracle_ldapuser
| |____oracle_parameter
| |____...
|____roles
| |____host
| | |____defaults
| | | |____main.yml
| | |____tasks
| | | |____main.yml
| | |____templates
| | | |____oracle-seclimits.conf.j2
| |____oracle-asm-diskgroups
| | |____defaults
| | | |____main.yml
| | |____tasks
| | | |____main.yml
| |____oracle-database
| | |____defaults
| | | |____main.yml
| | |____tasks
| | | |____main.yml
| | |____templates
| | | |____dbca.rsp.j2
| | | |____netca.rsp.j2
```

```
git clone https://github.com/oravirt/ansible-oracle-modules library
```

Ansible inventory

- Inventory describes for Ansible where the servers are
- Inventory only describes hosts
- Hosts can be divided into groups
- NB! Hosts may not exactly map into databases

Inventory and databases [Potential problem]

- One config file per db
  `group_vars/`**`db1`**

```
- Inventory
[db1]
host1.example.com

ansible-playbook dbconfig.yml -e db=db1
```

- Add another db config (db2), on the same host
  `group_vars/db1`
  `group_vars/`**`db2`**

```
- Inventory
[db1]
host1.example.com
[db2]
host1.example.com

ansible-playbook dbconfig.yml -e db=db2
```

- You might get unexpected results (variables might not come from the configuration you expected)

Inventory and databases [Solutions]

- Separate inventory files, one config per inventory

```
$ cat inventory/db1
[db1]
host1.example.com
$ cat inventory/db2
[db2]
host1.example.com
```

  and target the config with '`ansible-playbook … -i inventory/db1`' etc

- Use fake hosts

```
[db1]
host-db1    ansible_host=host1.example.com
[db2]
host-db2    ansible_host=host1.example.com
```

- Use one DNS CNAME per database, same concept as fake hosts
- Use one database per host

Inventory (random thoughts)

- Try to keep the inventories as standard as possible (for as long as possible)
- Maybe keep inventory in its own repository (to avoid duplication, if many different repos for management).
  - Possible to softlink, or just point playbook to inventory
- If environment is large, external service registry could be needed
  - Dynamic inventory script, like for AWS

Setting up scenarios – common variables

```
oracle_base: /u01/app/oracle
oracle_home: "{{ oracle_base }}/product/{{ oracle_db.version }}/db"
oracle_env:
       ORACLE_HOME: "{{ oracle_home }}"
       LD_LIBRARY_PATH: "{{ oracle_home }}/lib"
user: system
oracle_db:
       name: orcl
       passwd: "{{ oracle_passwd }}" #<- Password taken from external file
       version: 12.2.0.1
       cdb: true
       db_dest: +DATA
       fra_dest: +DATA
       pdb:
         - appdb1
         - appdb2
```

Scenario: Setting init.ora parameters 1/2 variables

```
init_parameters:
    - name: db_recovery_file_dest_size
      value: 60G
    - name: db_recovery_file_dest
      value: "{{ oracle_db.fra_dest }}"
    - name: db_create_file_dest
      value: "{{ oracle_db.db_dest }}"
    - name: filesystemio_options
      value: setall
      scope: spfile
    - name: optimizer_adaptive_features
      state: absent
    - name: audit_trail
      value: XML
      scope: spfile
    - name: sec_case_sensitive_logon
      value: TRUE
    - name: sec_max_failed_login_attempts
      value: 3
    - name: sql92_security
      value: TRUE
      scope: spfile
```

Scenario: Setting init.ora parameters 2/2 task

```
- name: Set parameters
  local_action:
        module: oracle_parameter
        hostname: "{{ inventory_hostname }}"
        service_name: "{{ oracle_db.name }}"
        user: "{{ oracle_db.user }}"
        password: "{{ oracle_db.passwd }}"
        name: "{{ item.name }}"
        value: "{{ item.value }}"
        state: "{{ item.state | default('present') }}"
        scope: "{{ item.scope | default('both') }}"
  environment: "{{ oracle_env }}"
  with_items: "{{ init_parameters }}"
  tags: param
```

Scenario: Creating PDB-s

```
- name: Create pdb
  local_action:
      module: oracle_pdb
      hostname: "{{ inventory_hostname }}"
      username: sys
      password: "{{ oracle_db.passwd }}"
      service_name: "{{ oracle_db.name }}"
      mode: sysdba
      name: "{{ item }}"
      datafile_dest: "{{ oracle_db.db_dest }}"
      state: present
  environment: "{{ oracle_env }}“
  with_items: “{{ oracle_db.pdb }}”
  tags: pdb
```

Scenario: Creating database users 1/3 variables

```
dba_user:
      - schema: dbauser
        schema_password_hash: "{{dbauser_pwhash}}"
        default_tablespace: dbauser_data
        default_temp_tablespace: temp
        grants:
              - dba

roles:
      - name: approle1
        role_grants:
              - create session
              - create table
              - select any dictionary
```

Scenario: Creating database users 2/3 task

```
- name: Create DBA user
  local_action:
          module: oracle_user
          hostname: "{{ inventory_hostname }}"
          service_name: "{{ oracle_db.name }}"
          user: "{{ oracle_db.user }}"
          password: "{{ oracle_db.passwd }}"
          schema: "{{ item.0.schema }}"
          schema_password_hash: "{{ item.0.schema_password_hash }}"
          state: "{{ item.0.state | default('present') }}"
          default_temp_tablespace: "{{ item.0.default_temp_tablespace }}"
          default_tablespace: "{{ item.0.default_tablespace }}"
          grants: "{{ item.1 }}"
          update_password: on_create
  environment: "{{ oracle_env }}"
  with_subelements:
     - "{{ dba_user }}"
     - grants
  tags: users
```

Scenario: Creating database users 3/3 task

```
- name: Create application role
  local_action:
    module: oracle_role
    hostname: "{{ inventory_hostname }}"
    service_name: "{{ oracle_db.name }}
    user: "{{ oracle_db.user }}"
    password: "{{ oracle_db.passwd }}"
    role: "{{ item.name }}"
    state: present
  environment: "{{ oracle_env }}"
  with_items: "{{ roles }}"
  tags: role
```

```
- name: Add grants to role
  local_action:
    module: oracle_grants
    hostname: "{{ inventory_hostname }}"
    service_name: "{{ oracle_db.name }}"
    user: "{{ oracle_db.user }}"
    password: "{{ oracle_db.passwd }}"
    role: "{{ item.name }}"
    grants: "{{ item.role_grants }}"
    state: present
  environment: "{{ oracle_env }}"
  with_items: "{{ roles }}"
  tags: role,grant
```

Scenario: Syncing users and roles from AD

```
- name: oracle_ldapuser
  local_action:
      module: oracle_ldapuser
      ### Target Oracle database connection information is cut from this slide
      ### LDAP parameters
      ldap_connect: "ldap://mycompany.domain:389"
      ldap_binddn: Administrator@mycompany.domain
      ldap_bindpassword: Ansible2017
      ldap_user_basedn: OU=Users,OU=Demo,DC=ansible,DC=test
      # The following filter means that objectClass is person, member of one specific group and account is not disabled
      ldap_user_filter: (&(objectClass=person)(memberOf=CN=prod_db,OU=Groups,DC=ansible,DC=test)(!(userAccountControl:1.2.840.113556.1.4.803:=2)))
      ldap_username_attribute: sAMAccountName
      ### New user parameters
      user_profile: LDAP_USER
      user_grants:
          - create session
          - create table
      group_role_map:
          - {dn: "CN=HR,OU=Groups,OU=Demo,DC=ansible,DC=test", group: "department_hr"}
          - {dn: "CN=IT,OU=Groups,OU=Demo,DC=ansible,DC=test", group: "department_it"}
      deleted_user_mode: lock
  environment: "{{ oracle_env }}"
```

All synced users must have a common dedicated profile!

Users missing from LDAP query will be locked or deleted.

Scenario: Maintaining wildcard privileges

```
- name: Manage what objects DEPARTMENT_HR can access
  local_action:
      module: oracle_privs:
      ### Target Oracle database connection information is cut from this slide
      roles:
          - DEPARTMENT_HR
      privs:
          - SELECT
          - FLASHBACK
      objs:
          - HR.%
          - HRREST.COUNTRIES
      objtypes:
          - TABLE
          - VIEW
      state: present
  environment: "{{ oracle_env }}"
```

```
SELECT owner, object_name
FROM all_objects
WHERE (owner||'.'||object_name LIKE
'HR.%' OR owner||'.'||object_name =
'HRREST.COUNTRIES')
AND object_type IN ('TABLE','VIEW')
```

Existing extra privileges not matching the privs + objects list will be revoked!

Scenario: Managing resource manager

```
- name: Manage consumer group for LDAP synced users
  local_action:
      module: oracle_rsrc_consgroup
      ### Target Oracle database connection information is cut from this slide
      name: ad_user_consgroup
      state: present
      # NB! Oracle resource manager actually does not support
      # user profile based mapping, so we expand it to user list
      map_oracle_user_profile: LDAP_USER
      grant_user_profile: LDAP_USER
  environment: "{{ oracle_env }}"
```

Managing passwords

- You should not store plain text passwords in playbooks!
- Ansible-vault
  - Ansible native solution to encrypt YML files that can be included in playbooks
  - Ansible-vault files can be committed to git
  - Password delivered through user input, file or script
- Oracle wallet
  - Ansible-oracle-modules use cx_Oracle python module that requires OCI client, therefore it also supports Oracle wallet authentication
  - Requires setting up tnsnames.ora and sqlnet.ora
  - Each TNS name needs password entry in wallet
    - Limitation for large deployments?
- Calling remote password service from playbook
  - REST API calls to e.g Passwordstate to fetch individual DB passwords
  - API key can be secured with ansible-vault

Scenario: Securing passwords with Ansible Vault

```
$ ansible-vault create passwords.yml

sys_password: Oracle123
dbauser_password: Oracle456

$ cat passwords.yml

$ANSIBLE_VAULT;1.1;AES256
37306631393231363334396666643466613062373336623836383333653963376463366234353130
39623836343561346630613131623861363862346161383 80a6166306265356131313534313565 36
66623831316637306432363763333231323139363535666137666333353463633763633333383963
353538613763316566 0a376435613736666663396233626531346530343462333865306137386434
353034666465643131313865653863643337626461303164393437643563666430346235636330 61
323738323761363038343566643735346436313034396564346137616337326534336465363462 31
343062396162376366326561333537343736
```

Scenario: Create job in all PDB 1/3 variables

```
pdb_json_query: pdb[?OPEN_MODE=='READ WRITE'].NAME

exclude_pdb:
    - PDB$SEED

common_job:
    name: DBAUSER.SEND_SLOW_SQL_REPORT
    job_type: plsql_block
    job_action: DBA_REPORTS_PKG.SEND_SLOW_SQL_REPORT
    repeat_interval: FREQ=DAILY;BYHOUR=8
```

Scenario: Create job in all PDB 2/3 tasks

```
# Connect to CDB and gather facts
- name: Gather target CDB facts
  local_action:
      module: oracle_facts
      hostname: "{{ inventory_hostname }}"
      service_name: "{{ oracle_db.name }}"
      user: "{{ oracle_db.user }}"
      password: "{{ oracle_db.passwd }}"
  register: dbfacts
  environment: "{{ oracle_env }}"
```

Scenario: Create job in all PDB 3/3 tasks

```
# Connect to all PDB-s and create the job
- name: Create a small test job in all PDBs
  local_action:
      module: oracle_job
      hostname: "{{ inventory_hostname }}"
      service_name: "{{ item }}"
      user: "{{ oracle_db.user }}"
      password: "{{ oracle_db.passwd }}"
      # Job info
      state: present
      name: "{{ common_job.name }}"
      job_type: "{{ common_job.job_type }}"
      job_action: "{{ common_job.job_action }}"
      repeat_interval: "{{ common_job.repeat_interval }}"
  environment: "{{ oracle_env }}"
  when: dbfacts.ansible_facts.cdb
  with_items: "{{ dbfacts.ansible_facts | json_query(pdb_json_query) |
difference(exclude_pdb) }}"
```

Task only runs when target database is CDB. Task is repeated for each PDB.

Scheduling/running playbooks

- We use Jenkins
- Basically 3 ways of running jobs
  - Manually
  - Trigger based
    - File is committed to Git
    - Git HEAD branch is merged to stable PROD branch
  - Scheduled

Running playbooks (a few other options)

- Ansible Tower (commercial offering)
- Ansible AWX (open source Tower)
- Rundeck
- Semaphore
- *Tensor*
- *Polemarch*

Oracle infrastructure as code

- No changes should be done manually on database infrastructure
- Changes done in Ansible code only
- Ansible code version controlled in git
- Commit/merge to git automatically executes Ansible playbooks that deploy the change

DEMO: Oracle infrastructure as code

- Demo environment:
  - Oracle 12cR2, with one CDB
  - Git
  - Bitbucket
  - Ansible executed from Jenkins
- Action flow:
  - Change in playbook committed to git (Bitbucket)
    - Add new ASM diskgroup
    - Change init.ora parameters in CDB
    - Create and initialise new application PDB
      - Add role with system grants
      - Add tablespaces
      - Add application schema
      - Add service
  - Bitbucket commit hook fires a job in Jenkins

Ansible is great, but...

- ... it is not for everything
- There are better solutions for:
  - Managing schema objects
  - Deploying database code
- Better look at specific tools for these purposes:
  - Liquibase
  - FlyWay
- But... Ansible can execute these tools as part of the release process orchestration

Resources

- Ansible homepage: https://www.ansible.com/
- Oracle database modules for Ansible:
  https://github.com/oravirt/ansible-oracle-modules
- Deploying RAC with Ansible:
  https://github.com/oravirt/ansible-oracle

Download scripts

- Here you can download the scripts used in this presentation
  - https://github.com/oravirt/oug-ansible-oracle

Thank You