Database Operations at Groupon using Ansible

Mani Subramanian
Sr. Manager
Global Database Services
Groupon
manidba@groupon.com

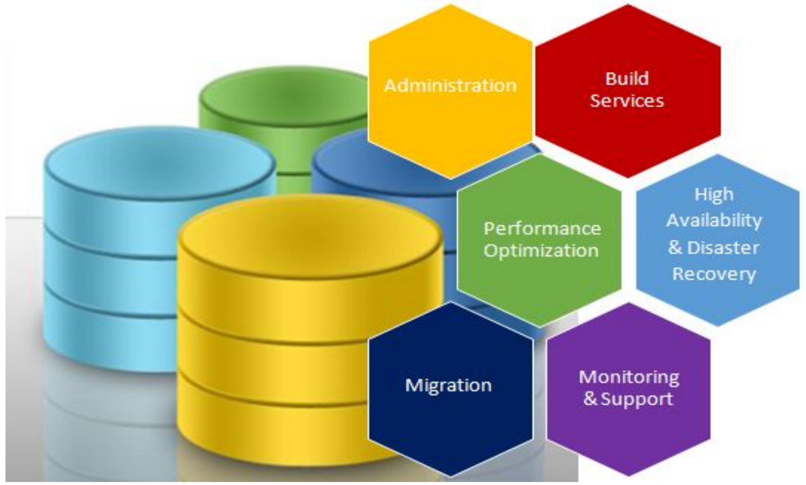
About me

- Worked as an Oracle DBA for 15+ years
- Branched out to MySQL since 2011
- Branched out to PostgreSQL since 2014
- Now - Managing Global Databases Services in Groupon

GROUPON

Global Database Services (GDS)

- Supports Production Operations
- Managing databases at scale both MySQL and PostgreSQL
- DaaS (Database as Service)
- Develop Tools and scripts for internal purpose
- Teams worldwide



GROUPON

Purpose of the Presentation

- How to make DBA life easier
    - By saving time
    - Reducing errors
    - Automating the routing Tasks
    - Eventually speedup the operations

Agenda

- What is Ansible?
- Ansible components & Architecture
- Ansible Terms & layout
- Provisioning new PostgreSQL DB instances (Demo)
- Setting up Streaming replication (Demo)
- DB failover user Ansible (Demo)
- Destroy DB instances (Demo)
- Recover database using ZFS
- Future potential developments

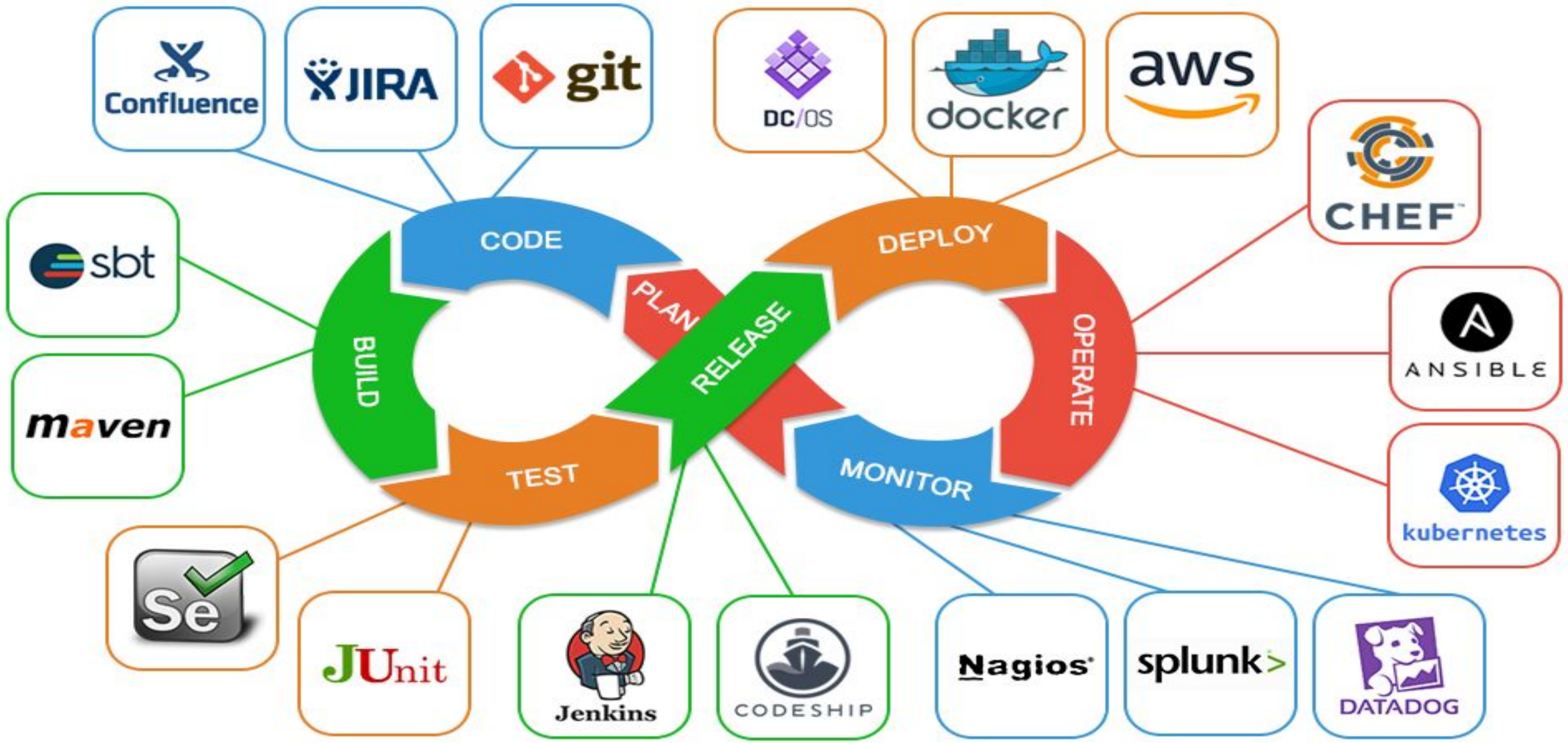
GROUPON

Environment

- FreeBSD Operating System
- ZFS FileSystem
- MySQL - Percona 5.6
- PostgreSQL - 9.4
- CARP - Common Address Redundancy Protocol
- Ansible 2.4
- Runit (init service)
- MHA for MySQL failover
- ZFS snapshots (FS level consistent backup)
- Xtrabackup (MySQL)
- Pg_basebackup(PostgreSQL)

GROUPON

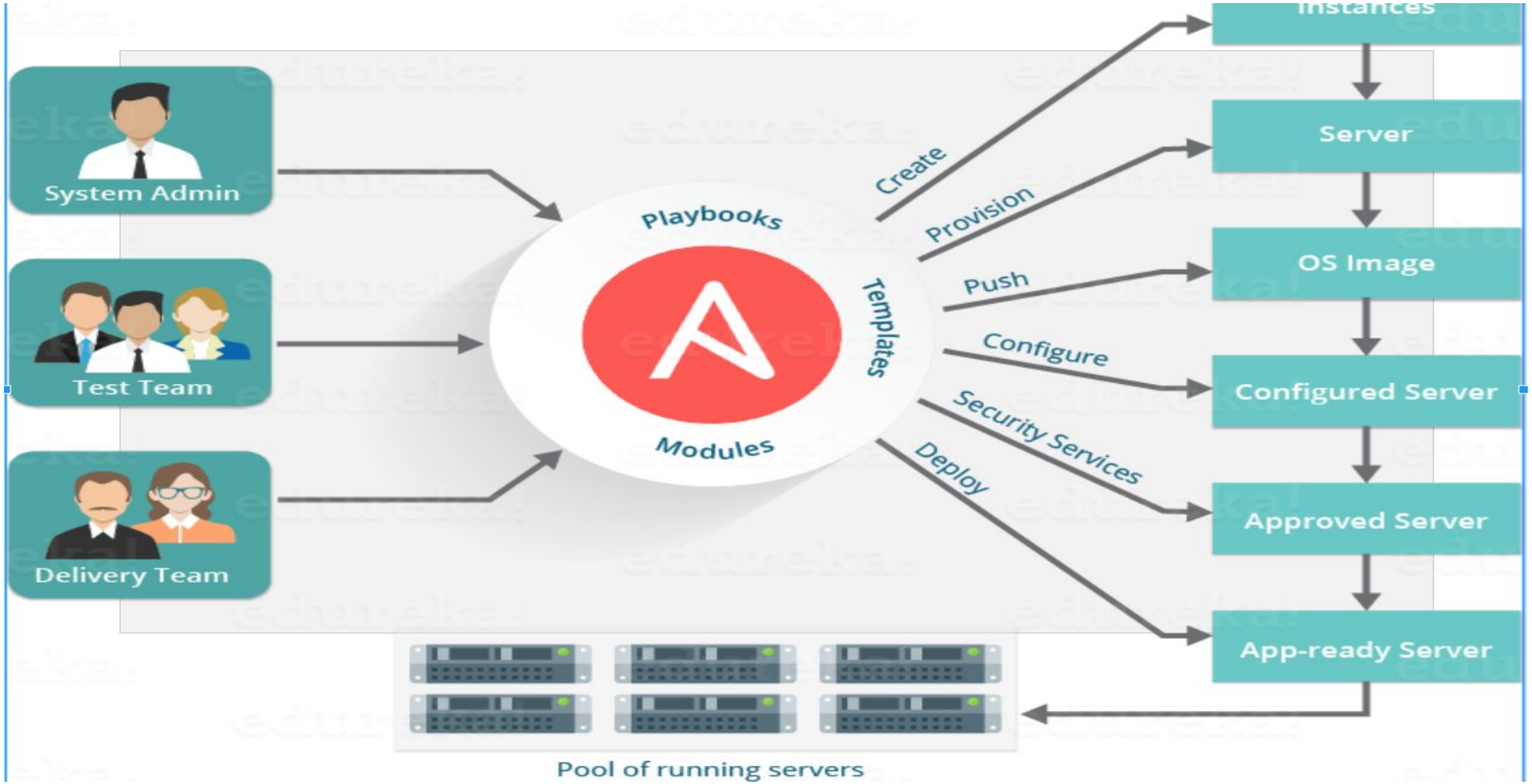
DevOps Tools



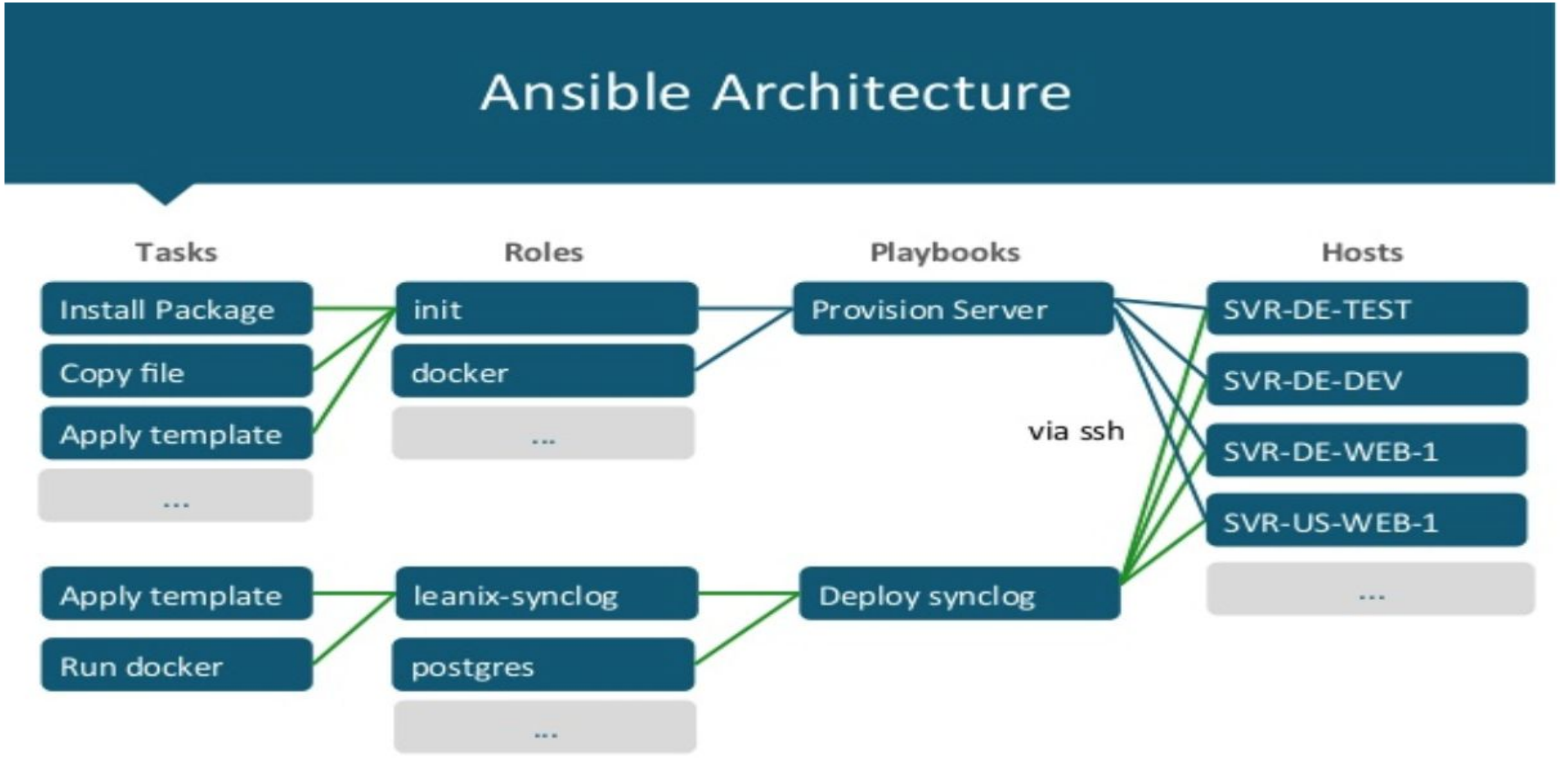
GROUPON

Ansible

- What is Ansible?
  - Radically simple IT automation engine
- Why Ansible in my opinion?
  - Automate tasks (sequence of tasks)
  - Save time and be more productive
  - Reduce mistakes or errors
- How does it work?
  - SSH Keys are your friends
  - No additional agents
  - Uses simple language (YAML in the form of Ansible playbooks)


Instances
Server
OS Image
Configured Server
Approved Server
App-ready Server
System Admin
Test Team
Delivery Team
Playbooks
Templates
Modules
Create
Provision
Push
Configure
Security Services
Deploy
Pool of running servers

GROUPON


Ansible Architecture
Tasks
Roles
Playbooks
Hosts
Install Package
Copy file
Apply template
...
Apply template
Run docker
init
docker
...
leanix-synclog
postgres
...
Provision Server
Deploy synclog
via ssh
SVR-DE-TEST
SVR-DE-DEV
SVR-DE-WEB-1
SVR-US-WEB-1
...

Simple Ansible Terms

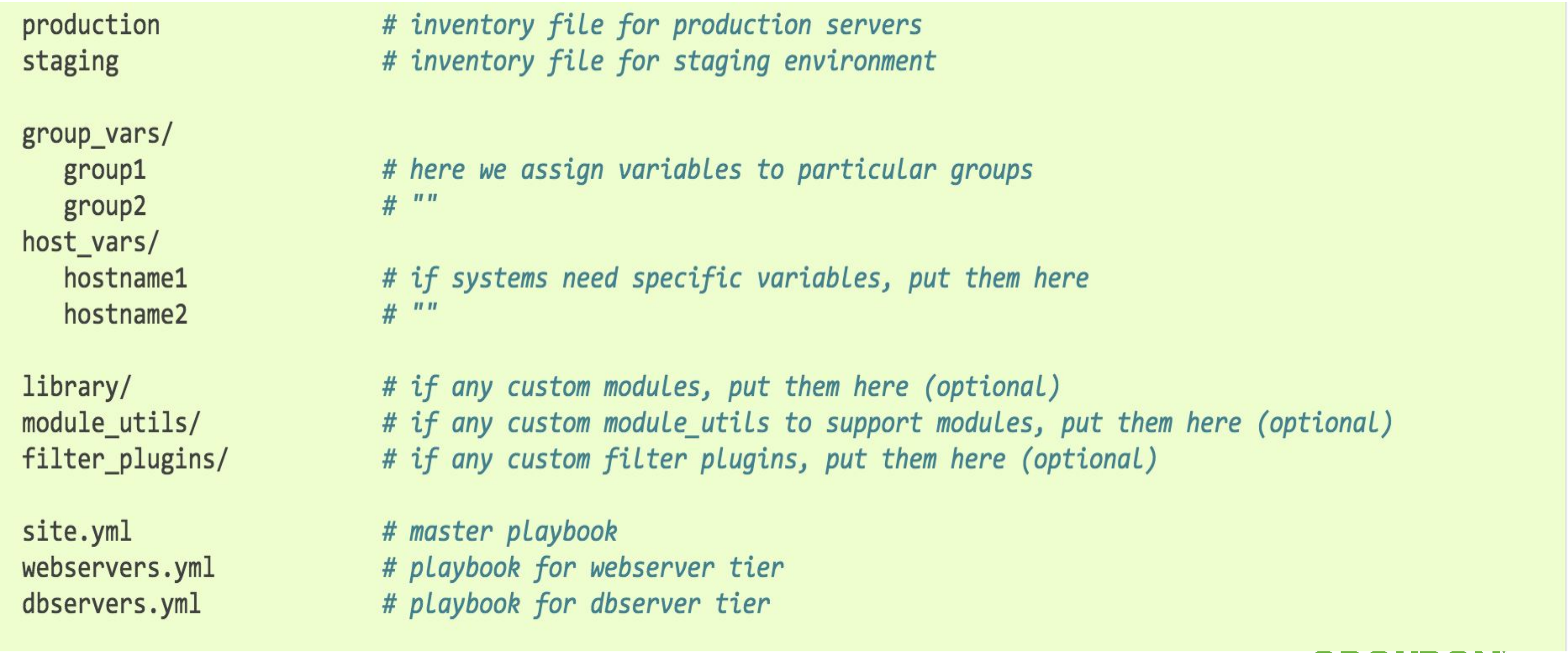
- Roles
  - Tasks
  - Handlers
  - Defaults
  - Vars
  - Files
  - Templates
  - Meta

Documentation : http://docs.ansible.com/ansible/latest/user_guide/

Roles Layout

```
ansible-foobar/
├── defaults
|   └── main.yml
├── files
├── handlers
|   └── main.yml
├── meta
|   └── main.yml
├── tasks
|   ├── check_vars.yml
|   ├── foobar.yml
|   └── main.yml
└── templates
    └── foobar.conf.j2
```

Directory Layout (1)



```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1                 # here we assign variables to particular groups
   group2                 # ""
host_vars/
   hostname1              # if systems need specific variables, put them here
   hostname2              # ""

library/                  # if any custom modules, put them here (optional)
module_utils/             # if any custom module_utils to support modules, put them here (optional)
filter_plugins/           # if any custom filter plugins, put them here (optional)

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier
```

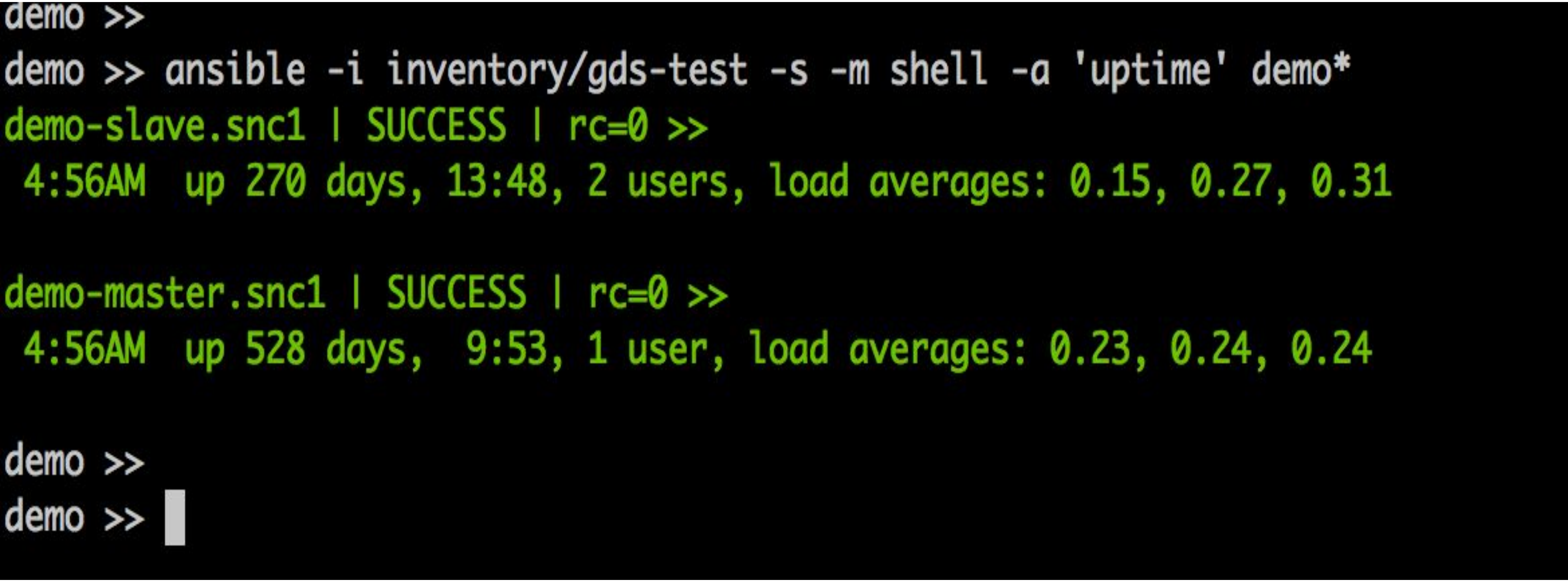
Directory Layout (2)

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

Ansible inventory file

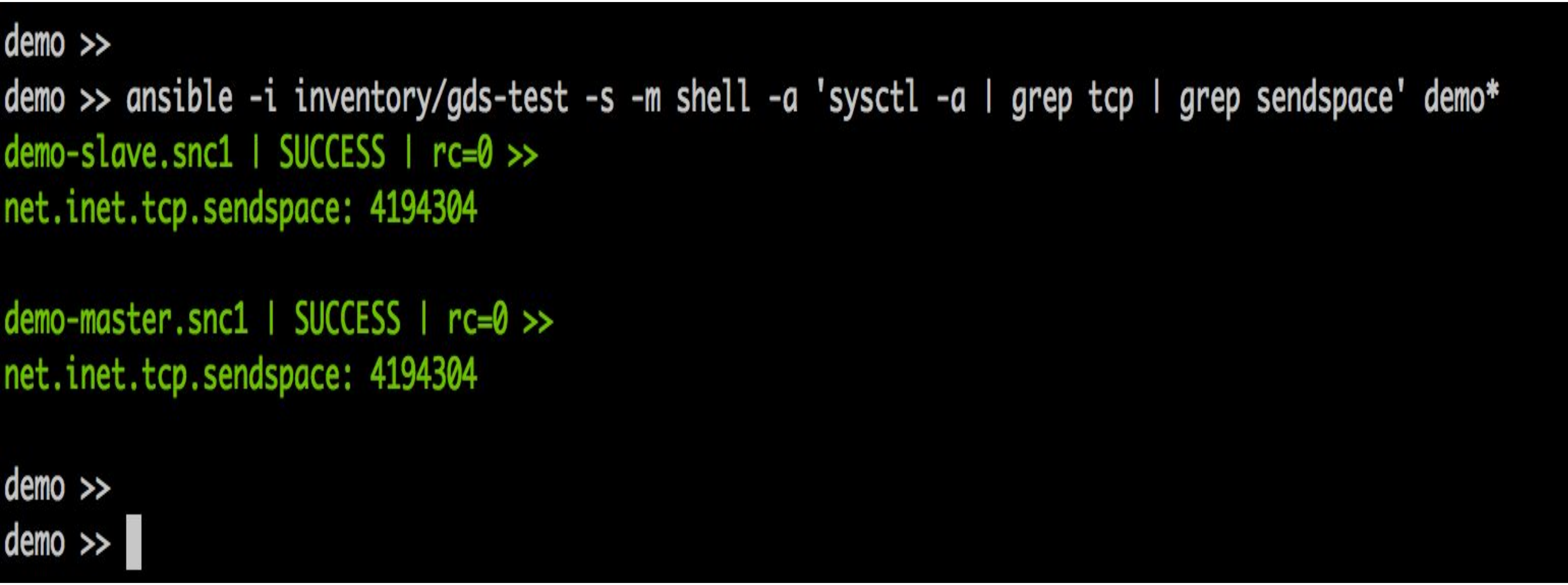
```
[demo]

demo-master.snc1

demo-slave.snc1

[sandbox]

sandbox-master.snc1

sandbox-slave.snc1

[testbox]

test-master.snc1

test-slave.snc1
```

Ansible usage - Example1



```
demo >>
demo >> ansible -i inventory/gds-test -s -m shell -a 'uptime' demo*
demo-slave.snc1 | SUCCESS | rc=0 >>
 4:56AM  up 270 days, 13:48, 2 users, load averages: 0.15, 0.27, 0.31

demo-master.snc1 | SUCCESS | rc=0 >>
 4:56AM  up 528 days,  9:53, 1 user, load averages: 0.23, 0.24, 0.24

demo >>
demo >> 
```

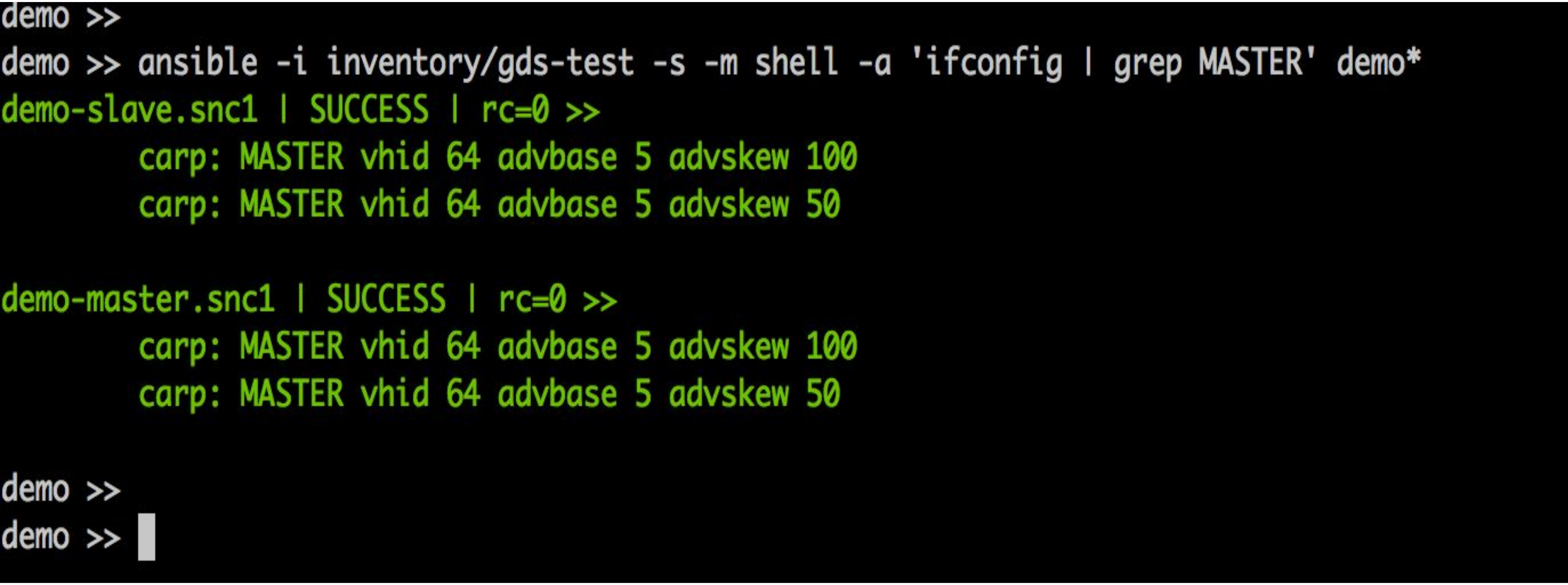
Ansible usage - Example2



```
demo >>
demo >> ansible -i inventory/gds-test -s -m shell -a 'sysctl -a | grep tcp | grep sendspace' demo*
demo-slave.snc1 | SUCCESS | rc=0 >>
net.inet.tcp.sendspace: 4194304

demo-master.snc1 | SUCCESS | rc=0 >>
net.inet.tcp.sendspace: 4194304

demo >>
demo >>
```

GROUPON

Ansible usage - Example3



```
demo >>
demo >> ansible -i inventory/gds-test -s -m shell -a 'ifconfig | grep MASTER' demo*
demo-slave.snc1 | SUCCESS | rc=0 >>
        carp: MASTER vhid 64 advbase 5 advskew 100
        carp: MASTER vhid 64 advbase 5 advskew 50

demo-master.snc1 | SUCCESS | rc=0 >>
        carp: MASTER vhid 64 advbase 5 advskew 100
        carp: MASTER vhid 64 advbase 5 advskew 50


demo >>
demo >>
```

Ansible usage - Example 4

```
ansible -s -m shell -a 'zpool list | grep tank ' demo*

ansible -m raw -a 'uptime' -i inventory/gds-test demo*

ansible -i inventory/gds-test -s -m shell -a 'zfs list -t snapshot | grep hourly'
demo*
```

DB Instances Provisioning

```
ansible-playbook -i inventory/gds-test plays/pg_instances.yml --limit
demo-master.snc1 -e "inst_type=master"
```

<<DEMO>>

```
ansible-playbook -i inventory/gds-test plays/pg_instances.yml --limit
demo-slave.snc1 -e "inst_type=slave"
```

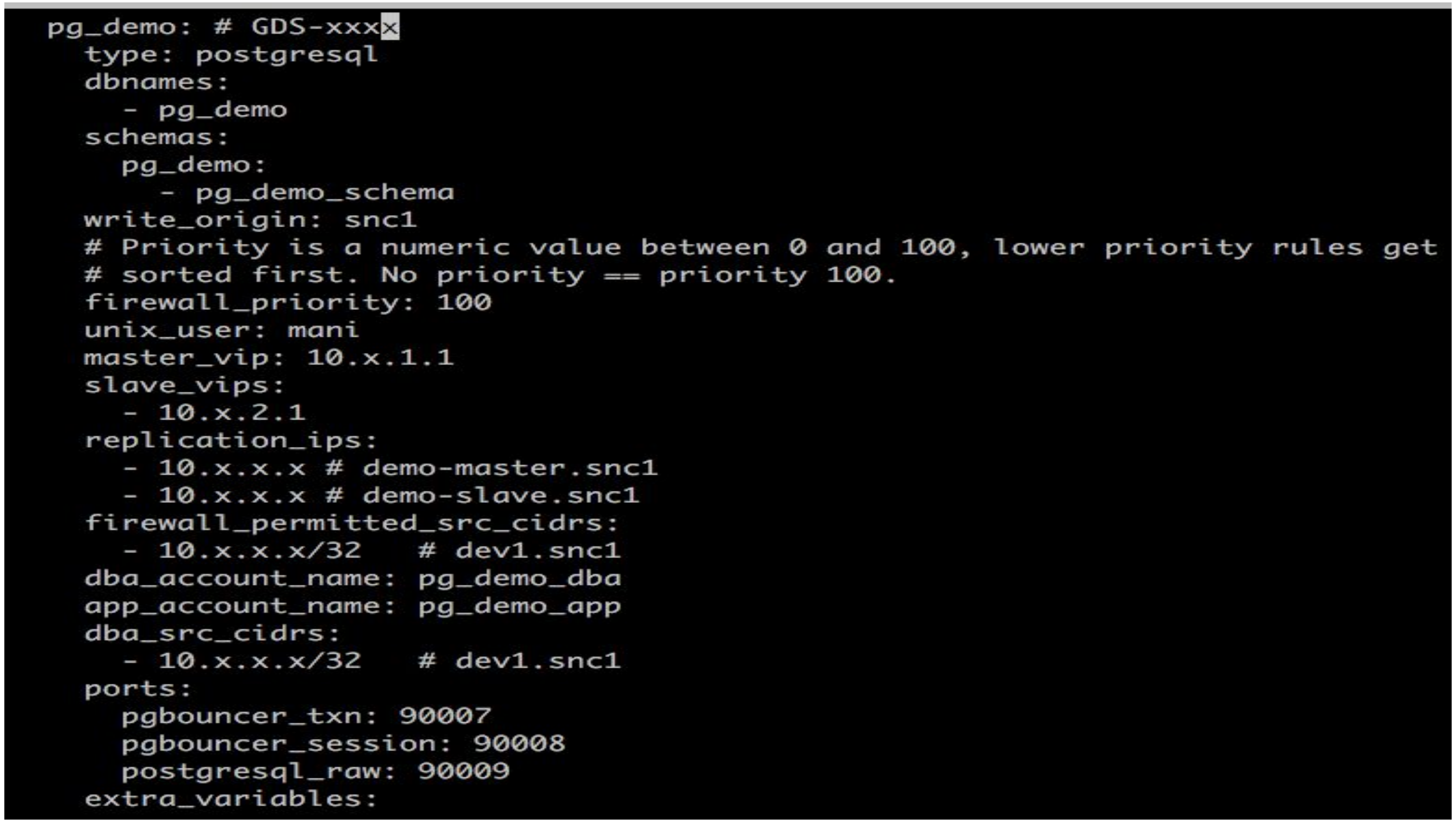
<<DEMO>>

```
ansible-playbook -i inventory/gds-test
plays/postgres/pg_install_initial_users.yml -e "master_server=demo-master.snc1
instance=pg_demo dba_password=DBAdemo911 app_password=APPdemo911"
```

Ansible-playbook options

-e	Set additional variables as key=value
-i	Inventory host path
-l	Further limit selected hosts
-v	verbose -vvv for more, -vvvv enable connection debug
- start-at-task	start the playbook at the task matching this name
--step	one-step-at-a-time, confirm each task before running

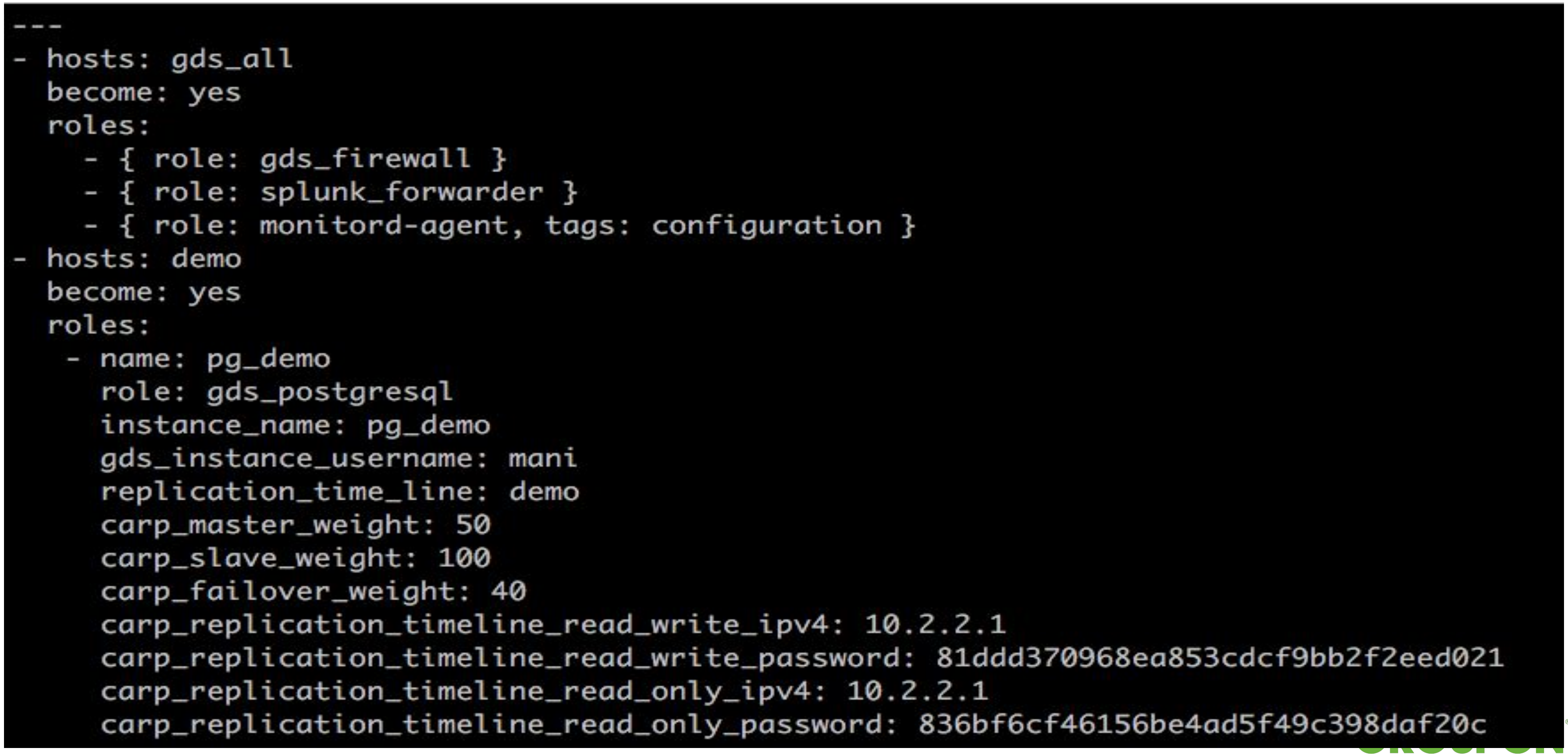
groupvar YAML for Provisioning



```
pg_demo: # GDS-xxxx
  type: postgresql
  dbnames:
    - pg_demo
  schemas:
    pg_demo:
      - pg_demo_schema
  write_origin: snc1
  # Priority is a numeric value between 0 and 100, lower priority rules get
  # sorted first. No priority == priority 100.
  firewall_priority: 100
  unix_user: mani
  master_vip: 10.x.1.1
  slave_vips:
    - 10.x.2.1
  replication_ips:
    - 10.x.x.x # demo-master.snc1
    - 10.x.x.x # demo-slave.snc1
  firewall_permitted_src_cidrs:
    - 10.x.x.x/32   # dev1.snc1
  dba_account_name: pg_demo_dba
  app_account_name: pg_demo_app
  dba_src_cidrs:
    - 10.x.x.x/32   # dev1.snc1
  ports:
    pgbouncer_txn: 90007
    pgbouncer_session: 90008
    postgresql_raw: 90009
  extra_variables:
```

GROUPON

plays/pg_instances.yml



```
---
- hosts: gds_all
  become: yes
  roles:
    - { role: gds_firewall }
    - { role: splunk_forwarder }
    - { role: monitord-agent, tags: configuration }
- hosts: demo
  become: yes
  roles:
   - name: pg_demo
     role: gds_postgresql
     instance_name: pg_demo
     gds_instance_username: mani
     replication_time_line: demo
     carp_master_weight: 50
     carp_slave_weight: 100
     carp_failover_weight: 40
     carp_replication_timeline_read_write_ipv4: 10.2.2.1
     carp_replication_timeline_read_write_password: 81ddd370968ea853cdcf9bb2f2eed021
     carp_replication_timeline_read_only_ipv4: 10.2.2.1
     carp_replication_timeline_read_only_password: 836bf6cf46156be4ad5f49c398daf20c
```

Replication Setup

```
ansible-playbook -i inventory/gds-test
plays/postgres/database-postgres-clone-slave.yml -e
"master_server=demo-master.snc1" -e "slave_server=demo-slave.snc1" -e
"instance=pg_demo"
```

<<< DEMO >>>

- Stop the slave service
- Wipes slave datadir
- Pg_basebackup to pull data
- Change ownership of data dir
- Set timeline on recovery.conf
- Start postgres service

Destroying DB instances - Part 1

```
ansible-playbook plays/postgres/pre-destroy-instance.yml -e
"node=demo-slave.snc1 instance=pg_demo"
ansible-playbook plays/postgres/pre-destroy-instance.yml -e
"node=demo-master.snc1 instance=pg_demo"
```

<<< DEMO >>>

It does

- Checks for connections
- Removes monitoring
- Stops the service

GROUPON

Destroying DB instances (part 2)

```
ansible-playbook plays/postgres/destroy-instance.yml -e
"node=demo-slave.snc1 instance=pg_demo"
ansible-playbook plays/postgres/destroy-instance.yml -e
"node=demo-master.snc1 instance=pg_demo"
```

<<< DEMO >>>

It does

- Deletes instances
- Removes snapshots
- Removes ZFS filesystem

Failover PostgreSQL database

```
ansible-playbook -i inventory/gds-staging plays/postgres/postgres-failover.yml --extra-vars
"current_master=demo-master.snc1 current_slave=demo-slave.snc1 instance=pg_demo"
```

<<< DEMO >>>

- Checkpoint on master
- Don’t allow anymore connections
- Kill all sessions
- Shutdown the master
- Checkpoint on slave
- Extract last checkpoint location
- Promote the slave
- etc....

ZFS Snapshots

It is a life saver in multiple scenarios like

- When User drops tables/wipes data mistakenly
- Rollback faster on any planned data changes
- Repeated load test on same data set is possibles
- Shipping ZFS filesystem to different box/data center for Research/Recovery purposes

<< Demo >>

Recover DB using Snapshots

```
-- Create objects

create table demo_table (c1 int, c2 varchar(10));

insert into demo_table values (generate_series(1,1000),'Demo');

select * from demo_table limit 5;
```

Recover DB using Snapshots cont.

```
-- Take snapshots

zfs snapshot -r tank/var/groupon/postgresql/data94/demo-pg_demo@pgconf19

-- changes to the db

drop table demo_table;

select * from demo_table limit 5;

-- Need to rollback & Check if you have needed snapshots

zfs list -t snapshot | grep pgconf19

zfs list -t snapshot | grep pgconf19 | awk '{print $1}' | xargs -n1 echo zfs rollback -r
```

Recover DB using Snapshots cont.

-- If all snapshots are available then stop the service

```
sv stat /var/groupon/service/postgresql-demo-pg_demo/

sv stop /var/groupon/service/postgresql-demo-pg_demo/

sv stat /var/groupon/service/gds_sandbox_demo-mysql/
```

-- Rollback to the right snapshots

```
zfs list -t snapshot | grep pgconf19 | awk '{print $1}' | xargs -n1 echo zfs rollback -r

sv up /var/groupon/service/postgresql-demo-pg_demo/

show databases;
```

Yaml file to schedule backup

```
sandbox-ro-vip.snc1-J1:
  host: sandbox-ro-vip.snc1
  instance_name:
  - sandbox_demo
  pool: us
  retention: 31d-1m-2y
  target: mysql
  template: daas_mysql_v2
  zfs_fs: snc1.prod.sandbox
```

GROUPON®

Scheduling backups

Run the key generation play:

```
ansible-playbook -i inventory/percona-demo plays/test-gen-ssh-keys.yml
```

Run the key installation play:

```
ansible-playbook -i inventory/percona-demo plays/test-install-keys.yml
```

Installing Backup Jobs:

```
ansible-playbook -i inventory/percona-demo plays/test-install-jobs.yml
```

Monitoring

- Ansible play creates instance and also pushes monitoring scripts to the hosts.
- Check-Mk agent executes to collect data for monitoring alerts
- More details http://mathias-kettner.com/check_mk.html

Future potential development

- Backfill and bulk DML using ansible
- Operations using CMDB
- Self service on DB instance provisioning and whitelisting app servers
- Schema change using ansible

Thank you

GROUPON