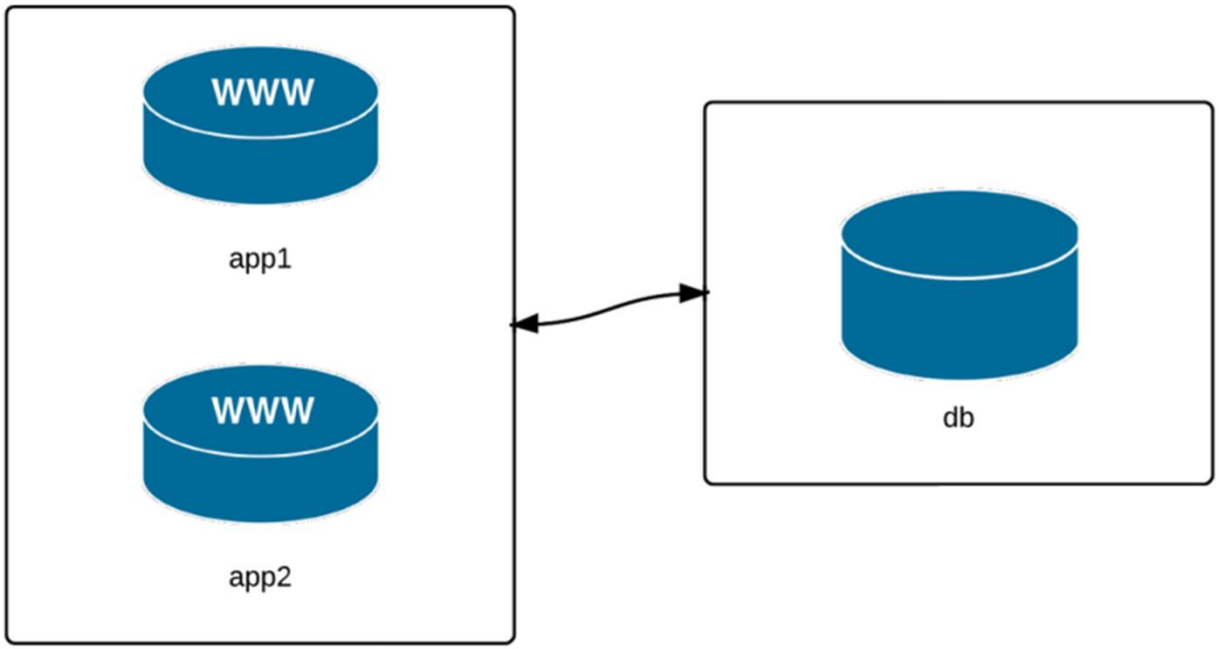
Ansible

Running ad hoc commands

- On any given day, a systems administrator has many tasks
  - Apply patches and updates via yum, apt, and other package managers
  - Check resource usage (disk space, memory, CPU, swap space, network)
  - Check log files
  - Manage system users and groups
  - Manage DNS settings, hosts files, etc
  - Copy files to and from servers
  - Deploy applications or run application maintenance
  - Reboot servers
  - Manage cron jobs

Ad hoc commands contd..

- Nearly all of these tasks can be (and usually are) at least partially automated—but some often need a human touch, especially when it comes to diagnosing issues in real time
- logging into servers individually is not a workable solution
- Ansible allows admins to run ad-hoc commands on one or hundreds of machines at the same time
- Before that, lets build local infrastructure
-



Ad hoc command contd..

- Run hostname command against all the servers
- $ ansible multi -a "hostname”
- Ansible will run this command against all three of the servers, and return the results
- By default, Ansible will run your commands in parallel, using multiple process forks, so the command will complete more quickly
- Run the same command again, but this time, add the argument -f 1 to tell Ansible to use only one fork (basically, to perform the command on each server in sequence)
- $ ansible multi -a "hostname" -f 1
- It’s fairly rare that you will ever need to do this, but it’s much more frequent that you’ll want to increase the value (like -f 10, or -f 25… depending on how much your system and network connection can handle) to speed up the process of running commands on tens or hundreds of servers
-

Ad hoc command contd..

- Let’s check if the servers have disk space available
- $ ansible multi -a "df -h“
- Lets check memory on our servers
- $ ansible multi -a "free -m“
- Let’s check date and time on each server
- $ ansible multi -a "date"

Ad hoc commands contd..

- Install NTP daemon on all servers to keep their time in sync
- Instead of running the command *yum install -y ntp* on each of the servers, we'll use *ansible's yum module* to do the same
-
- $ ansible multi -s -m yum -a "name=ntp state=present"
-
- -s option (alias for --sudo) tells Ansible to run the command with sudo.
- Now we'll make sure the NTP daemon is started and set to run on boot using *Ansible's service module*
-
- *$ ansible multi -s -m service -a "name=ntpd state=started enabled=yes"*

Ad hoc commands contd..

- Check to make sure our servers are synced closely to the official time on the NTP server
- $ ansible multi -s -a "service ntpd stop"
- $ ansible multi -s -a "ntpdate -q 0.rhel.pool.ntp.org"
- $ ansible multi -s -a "service ntpd start“
-
- For the *ntpdate* command to work, the ntpd service has to be stopped, so we stop the service, run the command to check our jitter, then start the service again
-
- Now lets set up application servers and database servers
- Since we set up two separate groups in our inventory file, *app and db*, we can target commands to just the servers in those groups

Ad hoc commands – install Django

- Our hypothetical web application uses Django, so we need to make sure Django and its dependencies are installed.
- Django is not in the official CentOS yum repository, but we can install it using Python's easy_install (which, conveniently, has an Ansible module)

(You can see we are targeting app group of servers only)

```
$ ansible app -s -m yum -a "name=MySQL-python state=present"
$ ansible app -s -m yum -a "name=python-setuptools state=present"
$ ansible app -s -m easy_install -a "name=django"
```

- Note: Recommended to Use –b (become Super User) instead of –s (Sudo switch).

```
$ ansible app -b -m yum -a "name=MySQL-python state=present”
$ ansible app -b -m yum -a "name=python-setuptools state=present”
$ ansible app -b -m easy_install -a "name=django"
```

Ad hoc commands – install Django contd

- Check to make sure Django is installed and working correctly

$ ansible app -a "python -c 'import django; print django.get_version()'“

Trouble Shooting : Python version mismatch

$sudo yum clean all
$sudo yum clean expire-cache
$sudo yum update

$sudo yum install epel-release
$sudo yum install python-django

$sudo yum install python-pip
$pip install --upgrade pip
$sudo pip install django

$ansible app -a "python -c 'import django; print django.get_version()'

Ad hoc commands – install MariaDB

- Let's install MariaDB, start it, and configure the server's firewall to allow access on MariaDB's default port, 3306
- ansible db -s -m yum -a "name=mariadb-server state=present"
- ansible db -s -m service -a "name=mariadb state=started enabled=yes"
- ansible db -s -a "iptables -F"
- ansible db -s -a "iptables -A INPUT -s 192.168.60.0 -p tcp \ -m tcp --dport 3306 -j ACCEPT"

Ad hoc commands – install MariaDB contd

- ansible db -s -m yum -a "name=MySQL-python state=present”
- ansible db -s -m mysql_user -a "name=django host=% password=12345 priv=*.*:ALL state=present“
- Note: need to run mysql_secure_installation to make a connection

Ad hoc commands – contd..

- Make changes to just one server
- ansible app -s -a "service ntpd restart" --limit "192.168.60.4“
- # Limit hosts with a simple pattern (asterisk is a wildcard). $ ansible app -s -a "service ntpd restart" --limit "*.4“
- # Limit hosts with a regular expression (prefix with a tilde). $ ansible app -s -a "service ntpd restart" --limit ~".*\.4“
- we’ve been using IP addresses instead of hostnames, but in many real-world scenarios, you’ll probably be using hostnames like nyc-dev-1.example.com