

# Introduction to DevOps using Fabric and Ansible

By Tim Henderson

[tadh@case.edu](mailto:tadh@case.edu) [hackthology.com](http://hackthology.com)

[github.com/timtadh](https://github.com/timtadh)

# A programmer's delight?

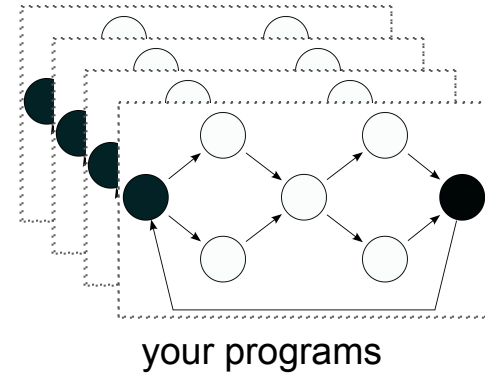
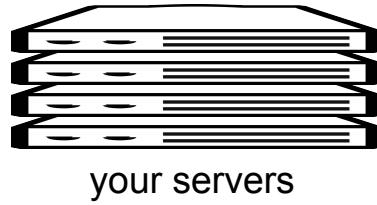
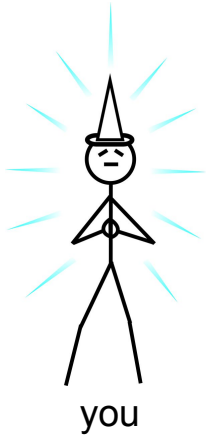
Server management should be:

1. Simple
2. Well documented
3. Repeatable
4. Testable
5. Auditable

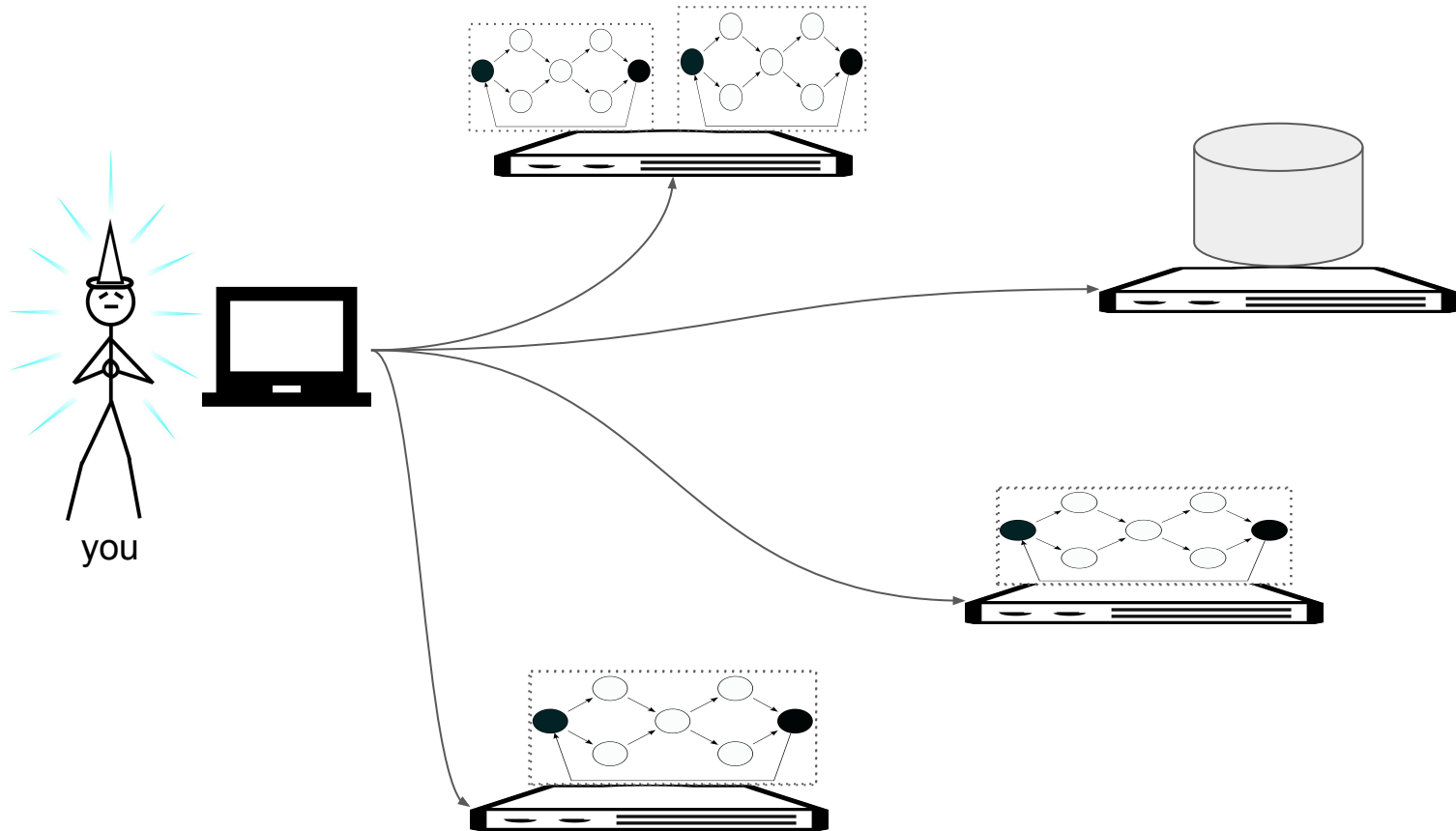
part 01

- step 0:
  - desc: never type commands on a server
  - reasons:
    - hard to keep servers in sync
    - hard to audit
    - hard to know server state
- step 1:
  - desc: use a tool
  - reasons:
    - easy to review changes
    - server state is known
- step 2:
  - desc: standardize all processes
  - reason: no special servers

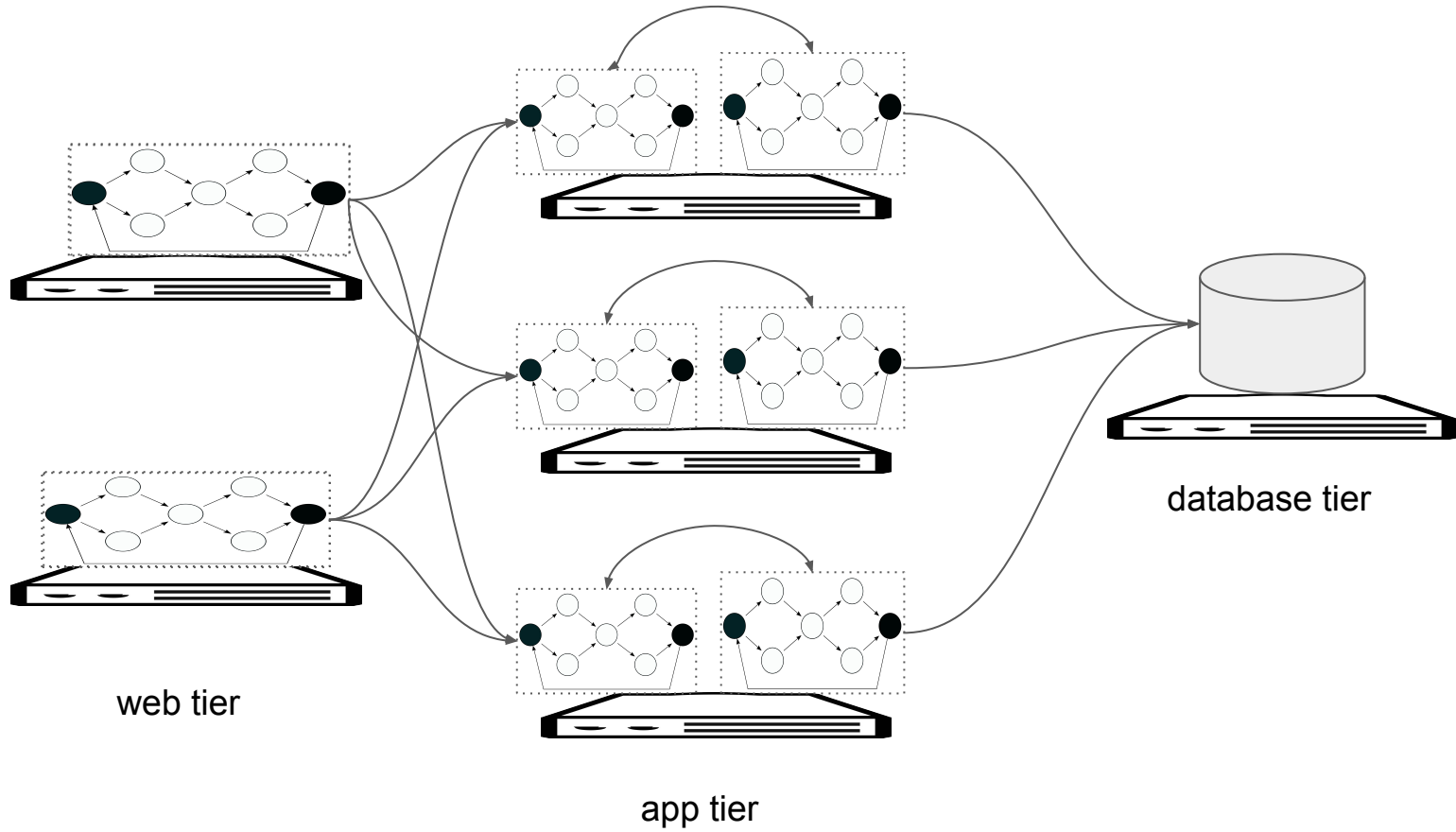
# definitions



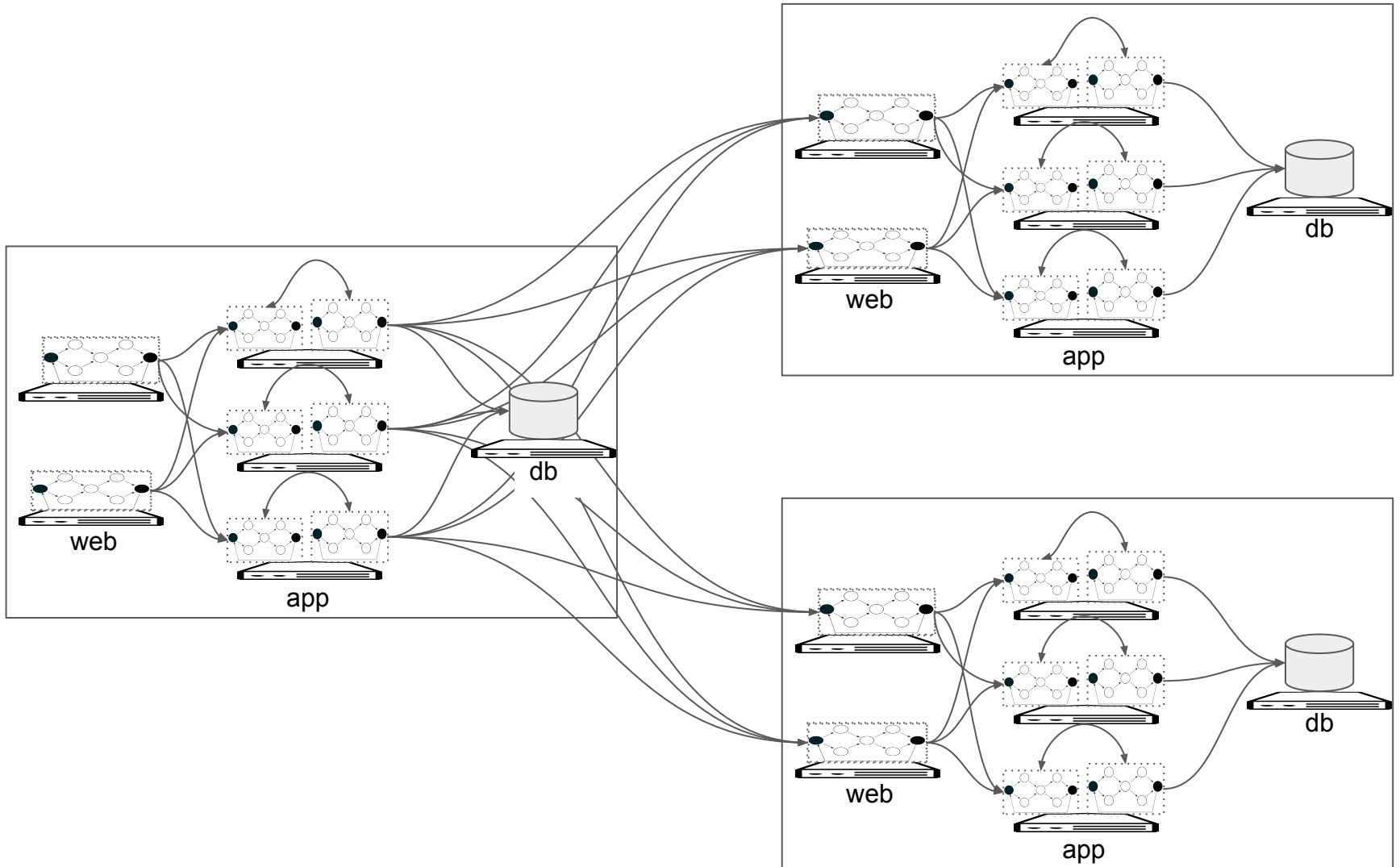
# definitions



a single service

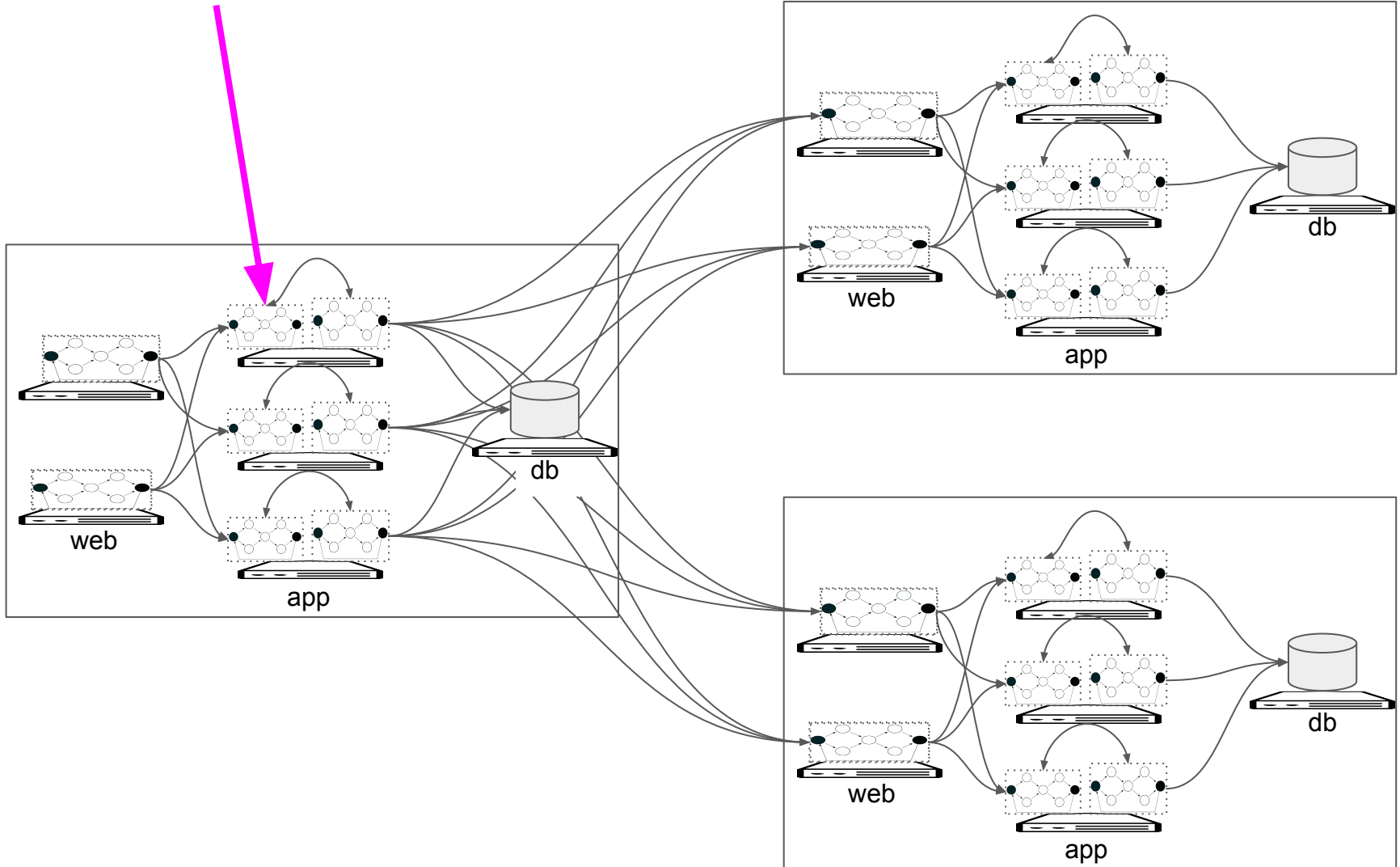


your “app”

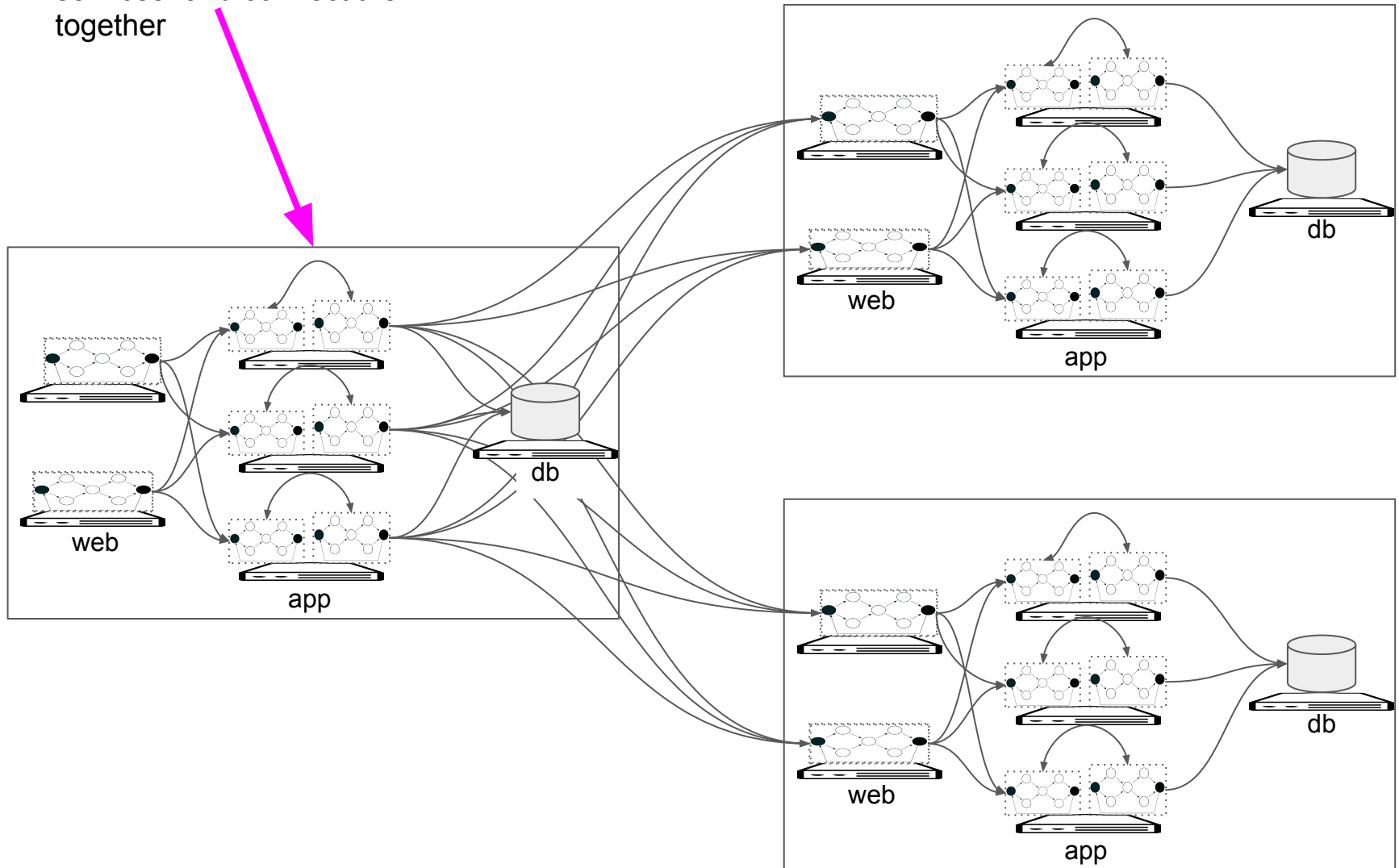




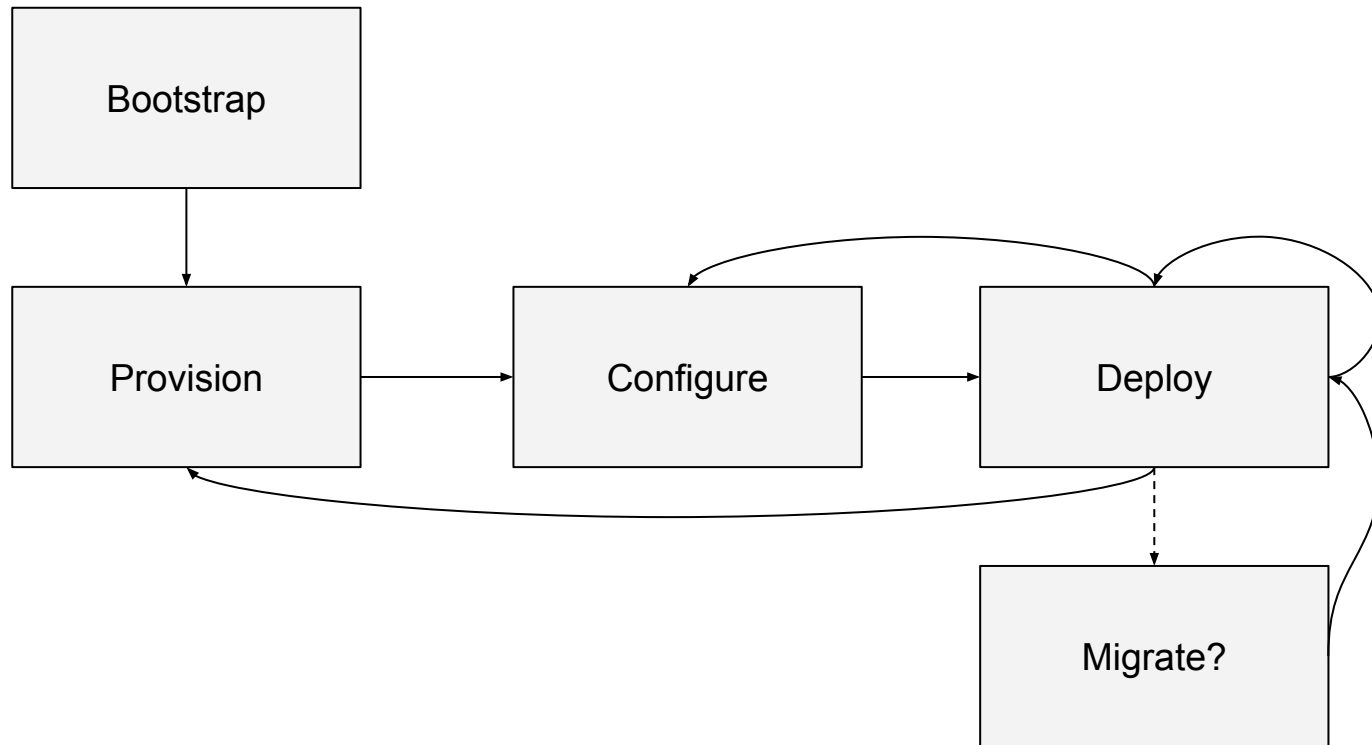
we provision individual  
“network services” onto  
individual servers.



but we also need to be able  
to control whole “logical  
services” and connect them  
together



# lifecycle of a network service



part 10

# Conquer the Gibson

Using Ansible and Fabric



# fabric

```
$ mkdir ops-example && cd ops-example
$ virtualenv --no-site-packages env
$ source ./env/bin/activate && pip install fabric
$ cat >fabfile.py <<EOF
> from fabric import api
>
> def who():
>     api.run('echo $(whoami) on $(uname -s)')
>
> EOF
$ fab -H localhost who
[localhost] Executing task 'who'
[localhost] run: echo $(whoami) on $(uname -s)
[localhost] out: henderson on Linux
[localhost] out:
Done.
Disconnecting from localhost... done.
```

# provision - config - deploy

```
from fabric import api

def provision():
    api.sudo('apt-get install python python-dev python-pip')
    api.sudo('pip install virtualenv')
    api.run('git clone git@host:group/app')
    api.run('virtualenv --no-site-packages app/env')

def config():
    api.put('startup.sh', 'startup.sh')
    api.sudo('mv startup.sh /etc/init.d/app')
    api.put('app-config.ini', 'app/config.ini')

def deploy():
    api.run('git -C app/ pull')
    api.run('source app/env/bin/activate && '
            'pip install -U -r requirements')
    api.sudo('service app restart')
```

simple, but brittle



enter ansible, a higher level tool

# ansible

```
$ mkdir ops-example && cd ops-example
$ virtualenv --no-site-packages env
$ source ./env/bin/activate && pip install ansible
$ cat >HOSTS <<EOF
> yourhost
> EOF
$ ansible all -i HOSTS -m shell \
>     -a 'echo $(whoami) on $(uname -s)'
yourhost | SUCCESS | rc=0 >>
henderson on Linux
$ cat >playbook.yml <<EOF
> - hosts: all
> tasks:
>     - shell: echo $(whoami) on $(uname -s)
> EOF
```

# ansible

```
$ mkdir ops-example && cd ops-example
$ virtualenv --no-site-packages env
$ source ./env/bin/activate && pip install ansible
$ cat >HOSTS <<EOF
> yourhost
> EOF
$ ansible all -i HOSTS -m shell \
>     -a 'echo $(whoami) on $(uname -s)'
yourhost | SUCCESS | rc=0 >>
henderson on Linux
$ cat >playbook.yml <<EOF
> - hosts: all
> tasks:
>     - shell: echo $(whoami) on $(uname -s)
> EOF
```

# ansible

```
$ cat >playbook.yml <<EOF
> - hosts: all
>   tasks:
>     - shell: echo $(whoami) on $(uname -s)
> EOF
$ ansible-playbook -v -i HOSTS playbook.yml
PLAY [all]
```

TASK [setup]

```
*****
ok: [yourhost]
```

TASK [command]

```
*****
changed: [yourhost] => {"changed": true, "stdout_lines":
["hendersont on Linux"]}
```

PLAY RECAP

```
*****
yourhost      : ok=2 changed=1 unreachable=0 failed=0
```

so far just a complicated ssh

modules, roles, and templates  
make ansible worthwhile

## a motivating module: git

```
$ cat >gitplay.yml <<EOF
> - hosts: all
>   tasks:
>     - name: update code for app repo
>       git:
>         repo: https://host.com/group/app.git
>         dest: /opt/app/
>         accept_hostkey: True
>         force: yes
>         update: yes
>         version: stable-branch
> EOF
```

```
$ ansible-playbook -i HOSTS gitplay.yml
```

...

TASK [command]

\*\*\*\*\*

```
changed: [yourhost] => {"after":
"9c46a49a73103de9a929718c223326149cb9accd", "before": null,
"changed": true}
```

...

# roles add structure to your code

```
$ tree -L 3 .
```

```
.
├── HOSTS
├── playbook.yml
├── roles
│   ├── base
│   │   ├── files/
│   │   ├── handlers/
│   │   └── tasks/
│   ├── go
│   │   ├── tasks/
│   │   └── templates/
│   ├── hadoop:base
│   │   ├── files/
│   │   ├── handlers/
│   │   ├── tasks/
│   │   ├── templates/
│   │   └── vars/
│   └── hdfs:datanode
│       └── meta/
```

```
├── tasks/
├── templates/
├── vars/
└── hdfs:namenode
    ├── handlers/
    ├── meta/
    ├── tasks/
    ├── templates/
    └── vars/
```



# roles are directories, with a specified structure

```
$ tree roles/hdfs:namenode/
```

```
roles/hdfs:namenode/
```

```
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   ├── config.yml
│   ├── main.yml
│   └── provision.yml
├── templates
│   ├── hdfs-namenode.conf
│   ├── hdfs-nfs.conf
│   └── hdfs-portmap.conf
└── vars
    └── main.yml
```

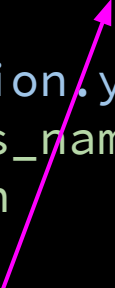
tasks/main.yml is the execution entry point

```
$ cat roles/hdfs:namenode/tasks/main.yml  
- include: "{{mode}}.yaml"
```

the mode variable is set in the calling playbook

```
$ cat roles/hdfs:namenode/tasks/main.yml
- include: "{{mode}}.yaml"
```

```
$ cat provision.yml
- hosts: hdfs_namenodes
  user: admin
  vars:
    mode: provision
  roles:
    - role: hdfs:namenode
  tags:
    - hdfs
    - hdfs:namenode
```



# which causes main.yml to load the correct tasks

```
$ cat roles/hdfs:namenode/tasks/main.yml
```

```
- include: "{{mode}}.yml"
```

```
$ cat provision.yml
```

```
- hosts: hdfs_namenodes
```

```
user: admin
```

```
vars:
```

```
  mode: provision
```

```
roles:
```

```
  - role: hdfs:namenode
```

```
tags:
```

```
  - hdfs
```

```
  - hdfs:namenode
```

```
$ tree roles/hdfs:namenode/  
roles/hdfs:namenode/
```

```
├── handlers
```

```
│   └── main.yml
```

```
├── meta
```

```
│   └── main.yml
```

```
├── tasks
```

```
│   ├── config.yml
```

```
│   ├── main.yml
```

```
│   └── provision.yml
```

```
├── templates
```

```
│   ├── hdfs-namenode.conf
```

```
│   ├── hdfs-nfs.conf
```

```
│   └── hdfs-portmap.conf
```

```
└── vars
```

```
    └── main.yml
```

# implementing lifecycle phases at the service level

```
$ cat roles/hdfs:namenode/tasks/main.yml
- include: "{{mode}}.yaml"
```

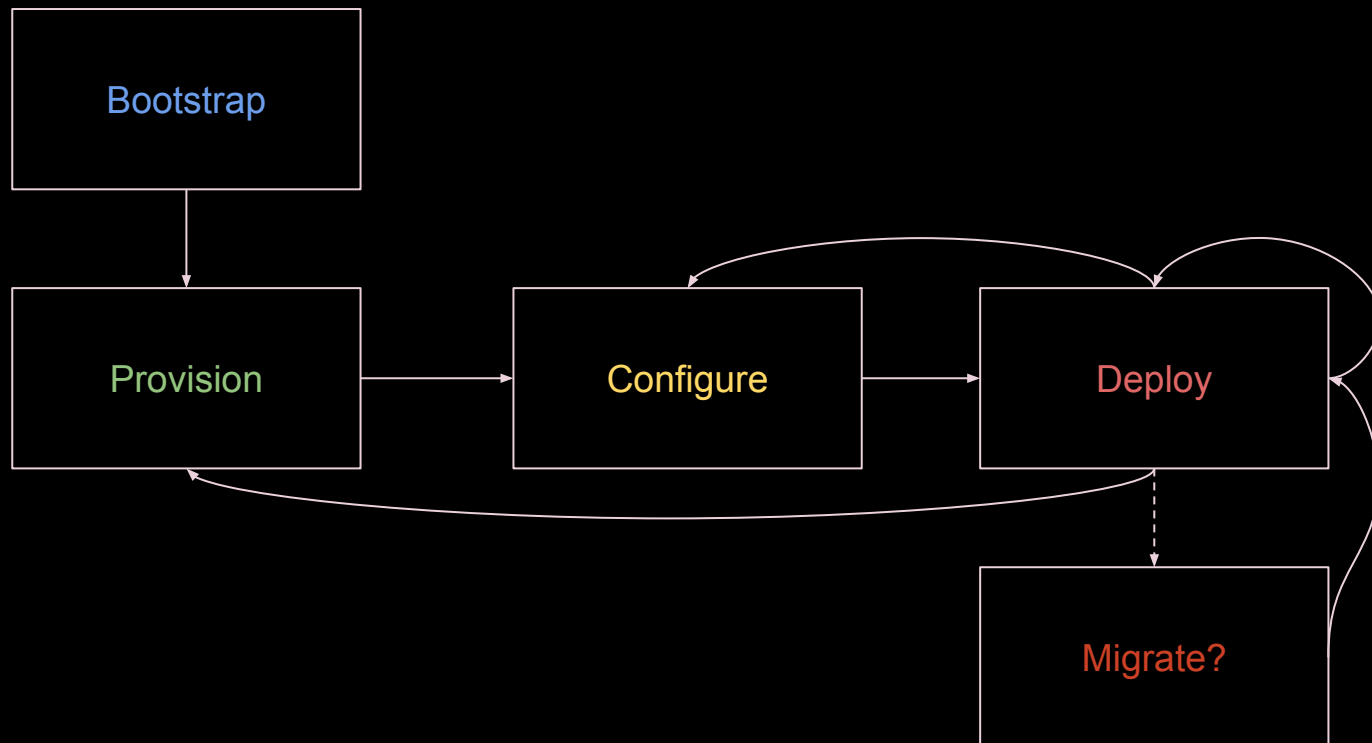
```
$ cat provision.yml
- hosts: hdfs_namenodes
  user: admin
  vars:
    mode: provision
  roles:
    - role: hdfs:namenode
      tags:
        - hdfs
        - hdfs:namenode
```

```
$ tree roles/hdfs:namenode/
roles/hdfs:namenode/
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   ├── config.yml
│   ├── main.yml
│   └── provision.yml
├── templates
│   ├── hdfs-namenode.conf
│   ├── hdfs-nfs.conf
│   └── hdfs-portmap.conf
└── vars
    └── main.yml
```

# 5 top level playbooks are used to run tasks

```
$ ls *.yaml
bootstrap.yaml
provision.yaml
config.yaml
deploy.yaml
migrate.yaml
```

```
$ ansible-playbook -i HOSTS lifecycle.yaml
PLAY [hdfs_namenodes]
*****
TASK [setup]
*****
ok: [host] ....
```



each playbook lifecycle calls the appropriate roles

```
$ cat config.yml
- hosts: hdfs_namenodes
  user: admin
  vars:
    mode: config
  roles:
    - role: hdfs:namenode
  tags:
    - hdfs
    - hdfs:namenode

- hosts: zookeeper
  user: admin
  vars:
    mode: config
  user: admin
  roles:
    - role: zookeeper
  tags:
    - zookeeper
```

# templates enable shared configuration between files

```
$ cat roles/hadoop\ :base/templates/core-site.xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>
      hdfs://{{hostvars[groups['hdfs_namenodes']][0]]
              ['ansible_fqdn']}}:{{hdfs_namenode_port}}
    </value>
  </property>
  <property>
    <name>io.file.buffer.size</name>
    <value>131072</value>
  </property>
  ...
</configuration>
```



let's stop here

# takeaways

## High level

1. Devs increasingly are increasingly performing operational roles
2. Ops folk are increasingly performing developer roles
3. Operations now looks more like programming than before

## Practical

1. Don't run commands on servers
2. Use a tool
3. Follow a standardized process for everything

## resources

- <https://docs.ansible.com/>
  - [https://docs.ansible.com/ansible/playbooks\\_best\\_practices.html](https://docs.ansible.com/ansible/playbooks_best_practices.html)
  - <https://github.com/ansible/ansible-examples>
- <http://docs.fabfile.org/>
  - <http://docs.fabfile.org/en/1.12/tutorial.html>
- <https://www.vagrantup.com/>
- <https://www.docker.com/>