A PUPPET/FABRIC BUILD/DEPLOY SYSTEM

Adrian Nye, Dimensional Fund Advisors

WHO BUILT THIS?

- Python Software engineer, not a dev-ops guy
- Long-time Fabric user, just learned puppet
- Developed this system with Gary Wilson, another python dev who also just learned puppet

THE TASK

- Start from bare RHEL 6 VMs, with only basic services pre-installed (puppet, ntp, networking/firewall rules)
- Provide tools to build, configure, and deploy:
  - 15 existing websites in various technologies:
    python, perl, php, ruby, & combinations
  - Mysql & Mongo databases
  - Memcache servers
  - Proxy servers
  - Search servers
  - Dev/Stage/Prod copies of all this
- Automate everything
  - Never touch any server by hand

SOME CHALLENGES

- RHEL 6 is stable but very old versions of most software. For example puppet hiera just became available as RPM.
- Stage & Prod servers won't have internet access
- Deployment to Stage/Prod will be done by operations people, not apps people.
  - Need rollback
  - Must have GUI or be simple

SOME CHOICES

- RPM or Source Installs?
- Git or Tar-based Deployment?
- Chef/Puppet/Ansible/SaltStack?
  - Puppet preferred by our infrastructure group
- We're python devs, so Fabric seemed obvious, it's not going away

SO WHAT IS FABRIC?

- Executes commands either local or remote (via ssh)
- Has functions for many common tasks
- Easy to script
- Anything you can do manually by ssh to a server, you can script fabric to do.
- Goal is a repeatable, idempotent sequence of steps.

BRIEF INTRO TO FABRIC

- Useful stuff it can do:
  - Confirm before doing things if you want
  - Run stuff in parallel on multiple machines, or serially
  - run stuff as if run from a directory
  - Get & put files, append to files, comment or uncomment lines
  - Upload templates and fill in variables
  - Run sudo commands
  - Connect to one host then to another within same function

FABRIC HOSTS

- Many ways to specify
- Most common is to use the Env variable, and set env.hosts
- Can specify on command line
- Can hardcode it (build tasks always happen on build server)
- Can make lists of hosts
- Functions on fab command line are executed in order, so first function can set host, and subsequence functions can use setting

EXAMPLE FABRIC TASK

```
def tail_log(logname):
    """tail a log file.

    fab hostname tail_log:access

    logname is filename of log (without .log)
    """

    log = env.logdir + '/' + logname + '.log'

    if file_exists(log):
        run('tail -f %s' % log, pty=True)
    else:
        print "Logfile not found in %s" % env.log_dir
```

SETTING HOST CLEVERLY

```
def dev(service_name=None):
    """Sets server as appropriate for service_name for the dev
environment.
    Also sets environ, server, and service_name in env so it is inherited
    by later fab commands.   Some fab commands need an environment but
are not
    specific to a service (such as mysql commands), so service_name is
optional.
    """
    _set_host_for_environment(service_name, Environment.DEV)
```

The above function is just a clever way of setting env.host to a hostname, so that later commands in the same fab command line know what system to work on.

HOW WE USED FABRIC

- We wrote classes using fabric to do all the tasks needed in the build and deployment process for our sites (and invoke puppet, which does the rest).
- All of it is data-driven. There is a file that defines the needs of each of our services and one that defines all of our servers.

CHALLENGES WITH FABRIC

- Your responsibility to make things idempotent.
  - For example, running “mkdir dirname” is not idempotent, because it will fail the second time. In this case use a routine that tests whether the dir exists, and if not then create it.
- Output control
  - Normal output is everything (very verbose). Good for debugging, although it can hide problems in sheer volume of information.
  - You can turn down the verbosity.
  - Really want two levels simultaneously: less verbose output displayed to the terminal, and fully verbose output logged to a file. But fabric doesn’t support that yet.

DIVISION OF RESPONSIBILITIES

- Puppet and Fabric capabilities overlap
  - both can do most tasks
  - Puppet is naturally idempotent
  - Fabric is naturally step by step
- Puppet: use to enforce *STATE*
  - RPM installation
  - Creation of upstart scripts from templates
  - User accounts
  - Files, Directories, and Permissions
- Fabric: use to enforce *WORKFLOW*
  - Build software environment (python virtualenv/perl modules etc)
  - Protect from simultaneous deploys
  - Testing of support services
  - Sync of software environment from build server to deploy server
  - Checkout of git repos, switching branches
  - Media syncing
  - Run puppet
  - Graceful server restart
  - Smoke testing (is site actually working after deployment)

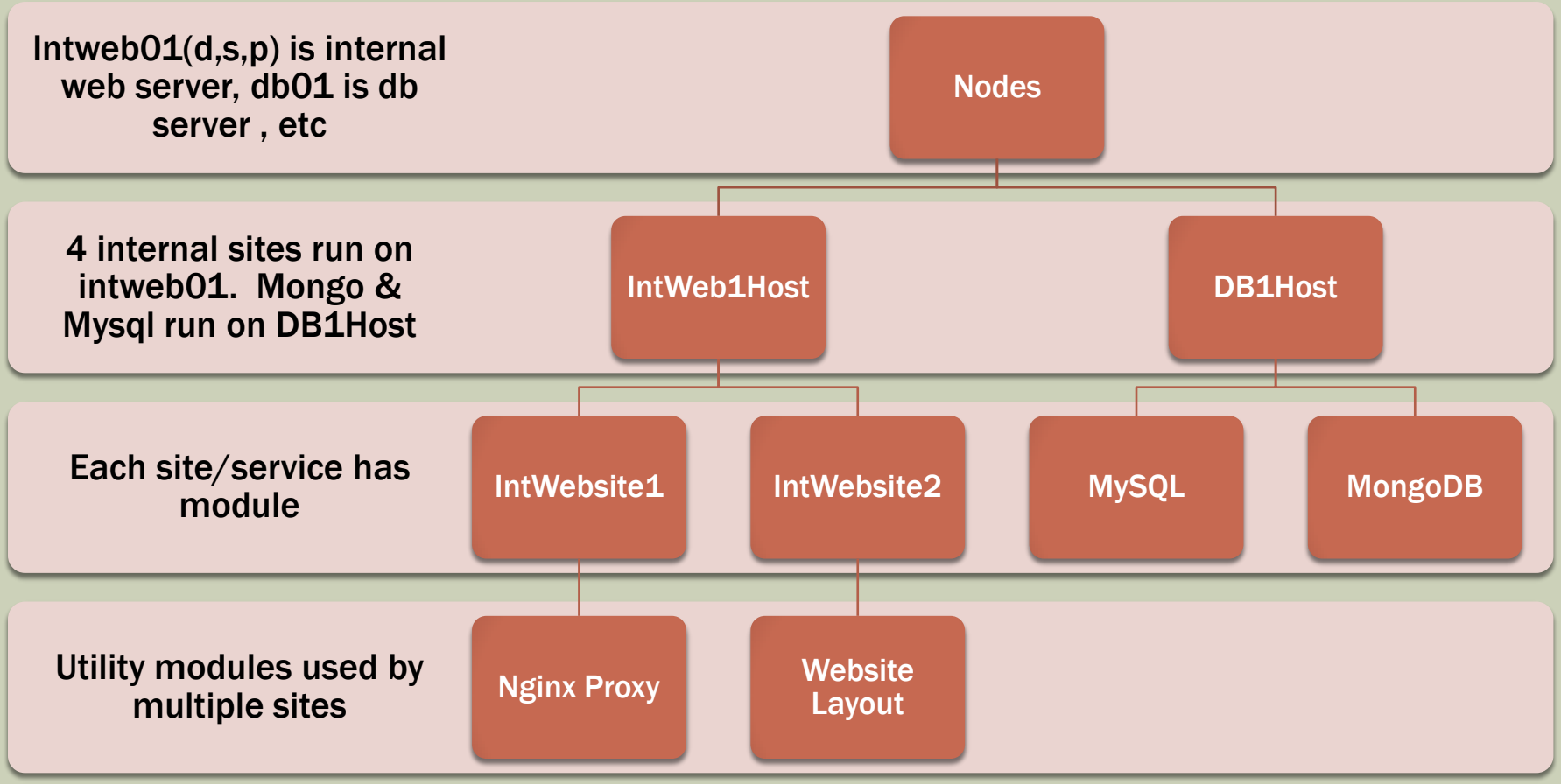
PUPPET/FABRIC GIT REPO

- All our custom puppet and fabric code is in a single git repo called “sysconfig”
- Enables everything to be run from anywhere with network access to build server.

DEV ENVIRONMENT

- Graphic of our dev servers and networking

PUPPET MODULES



PUPPET MODULES/MANIFESTS

- Nodes manifest connects hostnames with the type of host it will be, for all servers in all environments.
- Hosts manifest for each type of host (4 types of web servers, db server, cache server, proxy server, etc). This assigns sites to hosts.
- Site manifests for each type of service (each website, proxy, database). Does rpm installation, site-specific files & dirs, upstart scripts to start and stop service.
- Utility manifests for stuff needed by multiple sites, to minimize duplication. For example nginx module supports 3 different uses of nginx: fake dev load balancer proxy, ssl offloading proxy, local proxy.

PUPPET FEATURES

- Use Virtual Packages to enable every manifest to install its dependencies without regard to whether some other manifest has already installed it on the same server.
- Should use Hiera to enable Puppet/Fabric to pull from the same YAML database, but we haven't done this yet since Hiera just became available on RHEL 6.

FABRIC WORKFLOW

- 1. Build step builds software environment for site on build server
- 2. Deploy step copies the software environment from build server to the destination server, then deploys app code from scratch.

- Two-step process does several things:
  - Speeds up deployment, since build step is needed less often and takes a long time.
  - Speeds parallel deployment if you have redundant servers
  - Keeps compiling tools off destination servers. The less you install, the more secure they are.

FABRIC BUILD WORKFLOW

- Pip/Virtualenv used for Python packages, requirements file in git repo
- Cpanm used for Perl modules
- Rbenv used for ruby modules
- All packages, modules, and rpms mirrored locally
  - Improves reliability and speed
  - Simplifies version control
- Everything (except rpms) installed in /opt/comms/servicename, not system-wide. This simplifies copying the environment to the deployed server, and simplifies recreating a clean build.

FABRIC BUILD SERVICE DEFINITION

- Example service definition
  - Name (for fab commands)
  - Server type it should be installed on (not hostname)
  - Domain (without dev/stg/com)
  - Ssl or not
  - How to smoke test it
  - Init scripts it needs
  - Git repos to check out, including branch, and any media
  - Log dirs
  - Languages needed
  - Prerequisite services to check (memcache, db)
  - Related services to reload (nginx, memcache)
  - dirs containing built software environment (virtualenv, cpanm etc)

FABRIC DEPLOYMENT WORKFLOW

- 1. Mark deployment as in progress (using lock file)
- 2. Check support services
  - If db needed, is it running?
  - If memcache needed, is it running?
  - If critical support service not running, ask whether to continue.

FABRIC DEPLOYMENT WORKFLOW

- 3. clone/pull the git repo(s) needed for the site. Checkout the specified branch.

FABRIC DEPLOYMENT WORKFLOW

- 4. Move previous software environment for fast rollback
- 5. Rsync software environment for site
- 6. Rsync media for site

FABRIC DEPLOYMENT WORKFLOW

- 7. Run puppet. For convenience we support two modes:
  - Use puppet master
  - Copy developer's sysconfig repo and run puppet using those modules. This makes development a lot faster.

FABRIC DEPLOYMENT WORKFLOW

- 8. Zero Downtime Restart
- 9. Smoke Test
  - For web servers, check that site is up, login works, and run selenium tests.
  - For memcache etc, use nc to test basic operation.
  - Note that if deployment fails, there would be some downtime until previous version reinstated.

FABRIC UTILITIES WE WROTE

- Setting up SSH keys on all servers
- Log viewers
- Database backups
- Database copy from one environment to another (i.e. copy production db back to dev)
- Determine hostname from service name and environment
- Status/Start/stop/reload any remote service/site
- Media syncing from environment to environment
- Proxy server config generation
- Running puppet, using puppet master and without
- Smoke testers for different types of sites & services
- Tools to make local mirrors of internet software

FUTURE WORK

- Support for replicated (redundant) servers.
- GUI for common tasks, using Rundeck or Jenkins or TeamCity
- Network logging (Splunk)