Dominating Your Systems Universe with Ansible

Daniel Hanks | Sr. System Administrator – Adobe Systems Incorporated

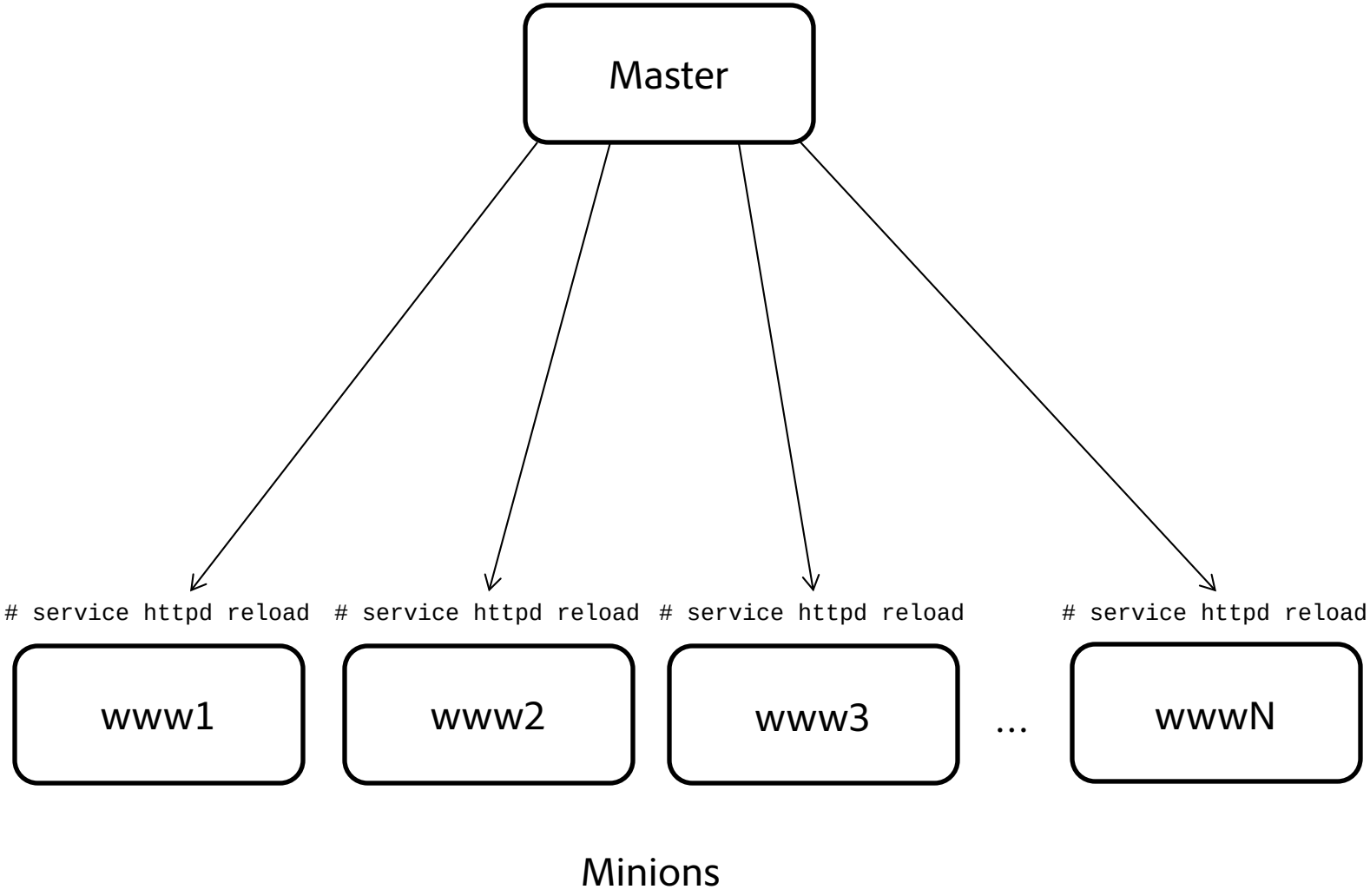
What is Ansible?

- "Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.
- Ansible's goals are foremost those of simplicity and maximum ease of use. It also has a strong focus on security and reliability, featuring a minimum of moving parts, usage of OpenSSH for transport (with an accelerated socket mode and pull modes as alternatives), and a language that is designed around auditability by humans – even those not familiar with the program."
- ansible.com

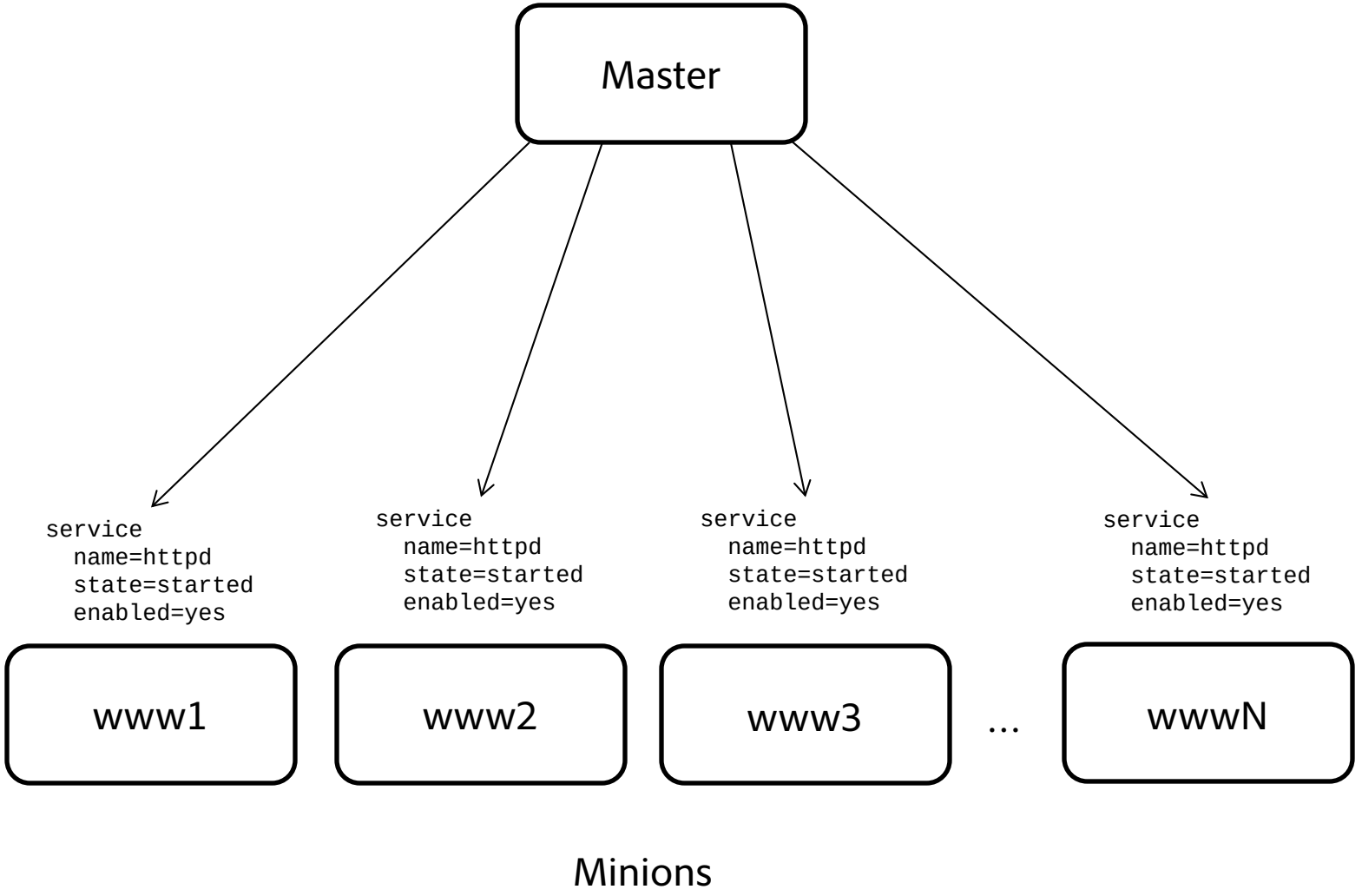
Who is Ansible?

- Michael Dehaan, founder
  - Author of Cobbler, Func
- www.ansible.com
- @ansible
- Freenode: #ansible
- Top-ten Python project on GitHub (702 contributors)

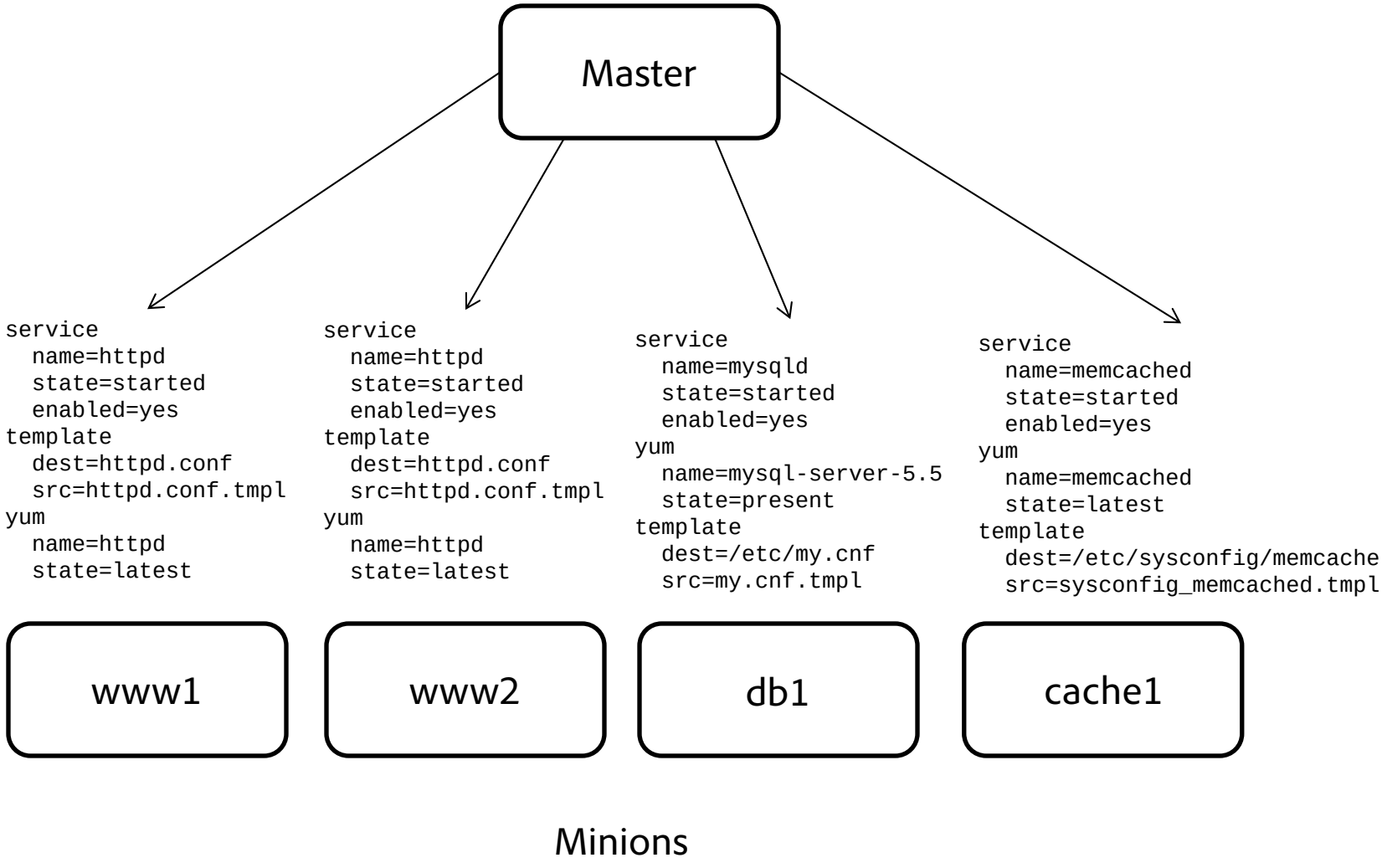
Why use Ansible? – Command and Control



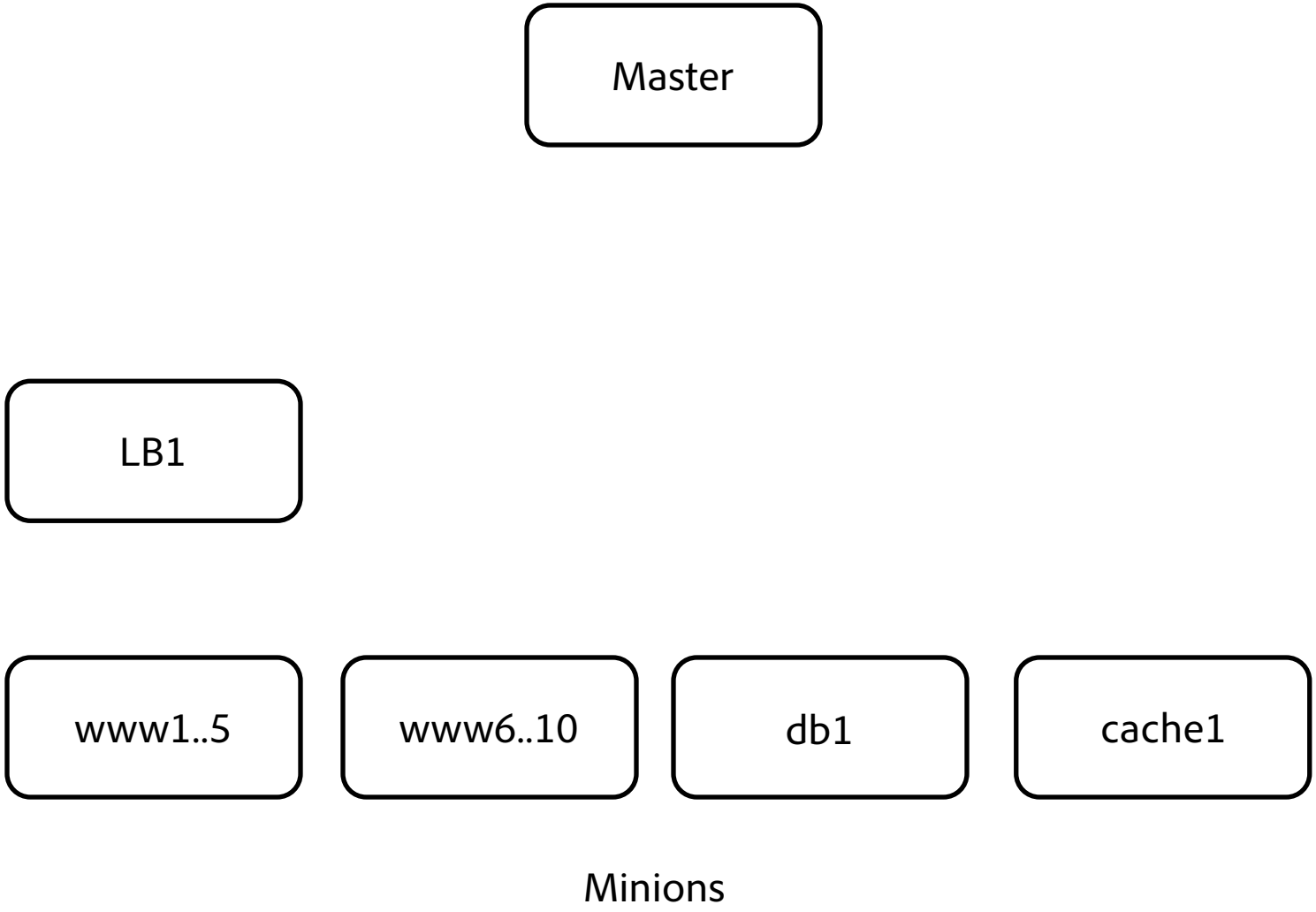
Why use Ansible? – Structured Command and Control



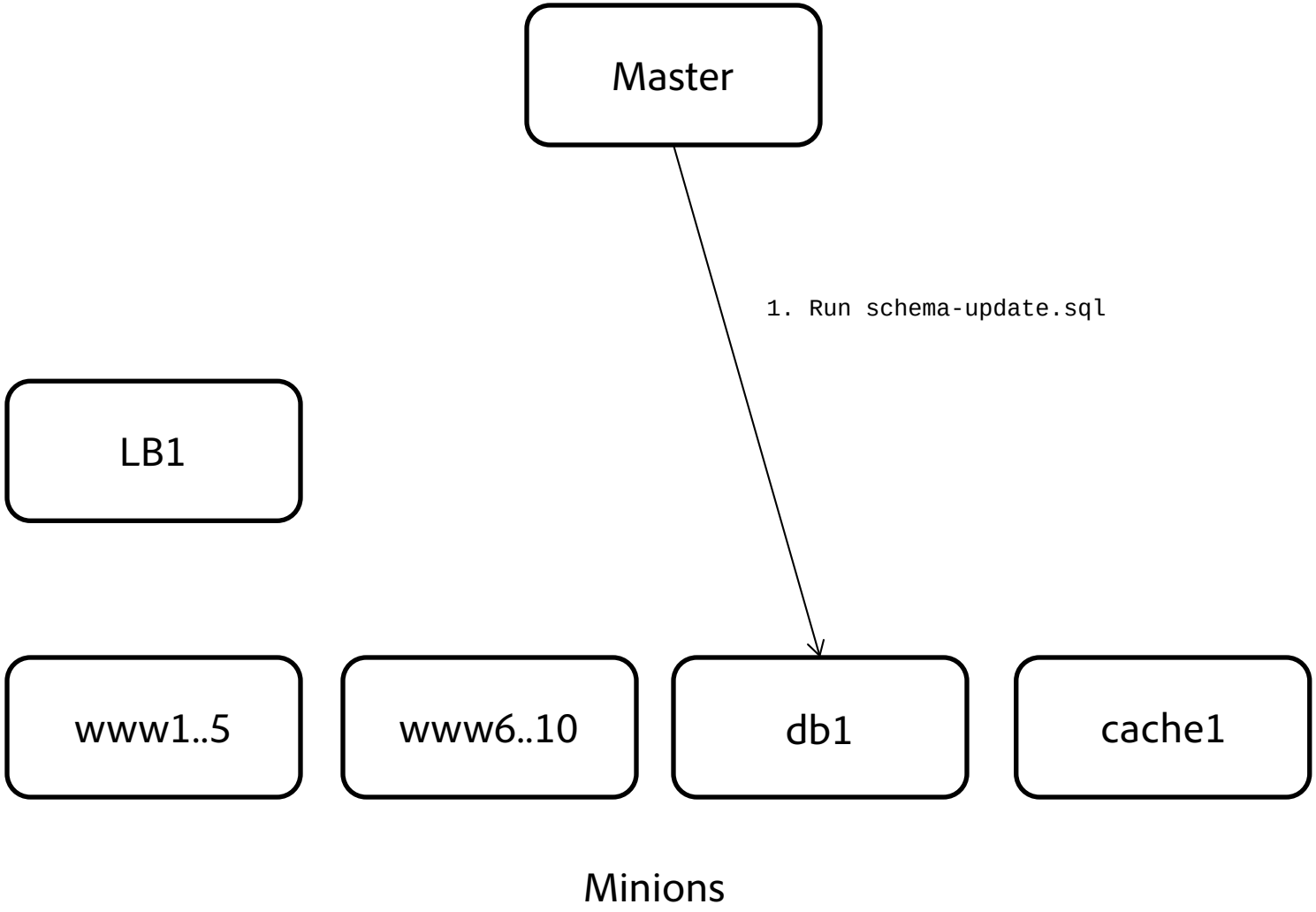
Why use Ansible? – Configuration Management



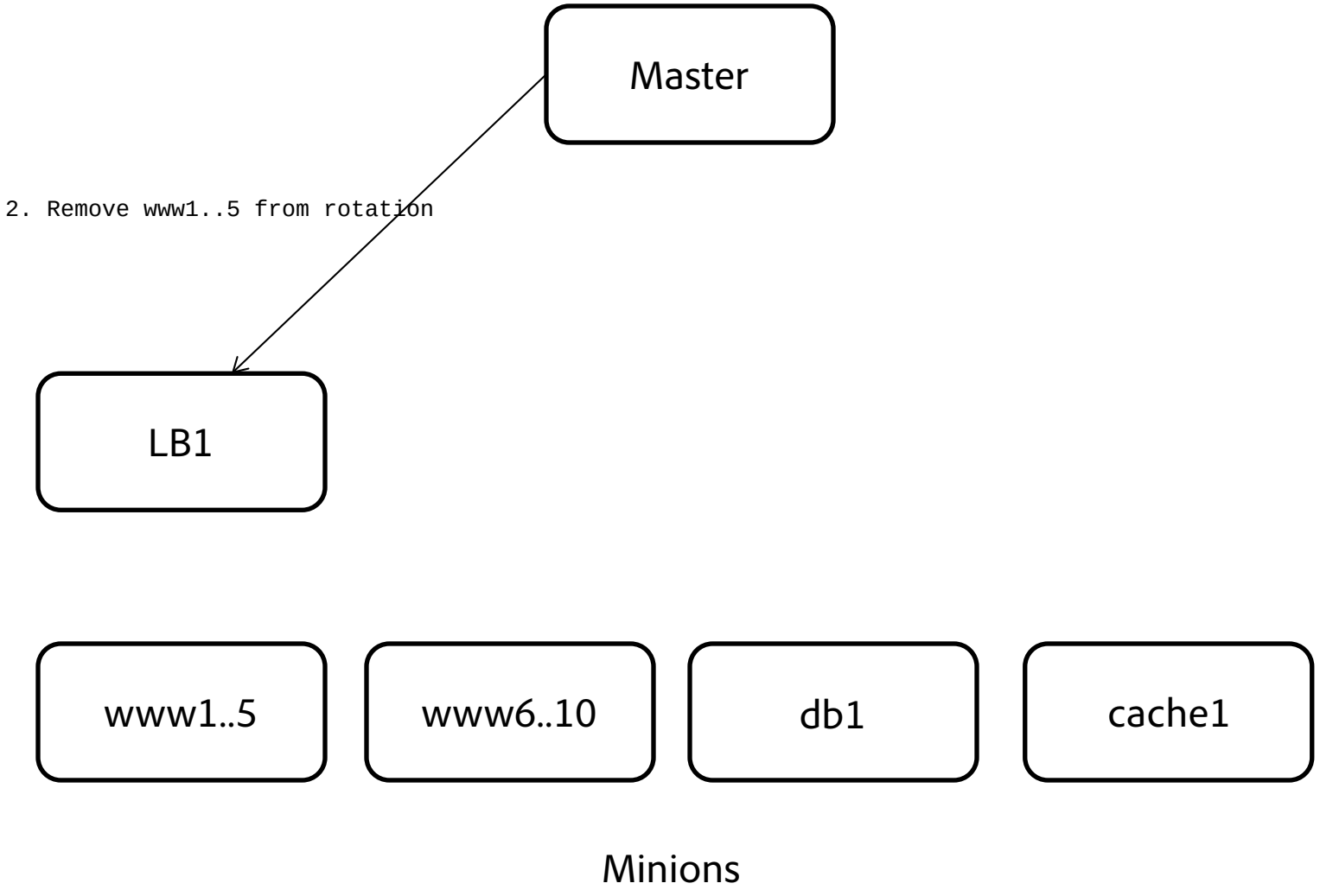
Why use Ansible? – Orchestration



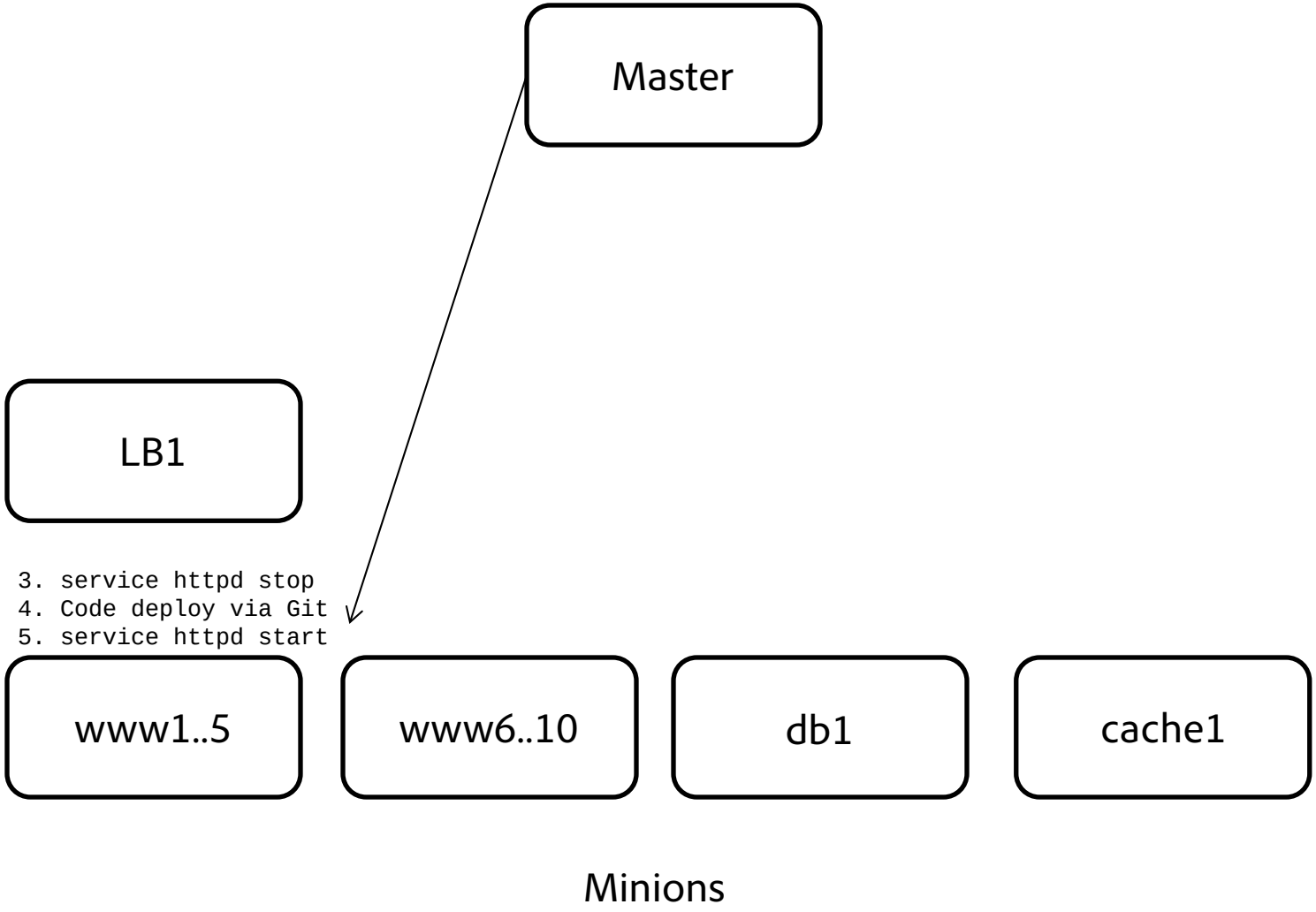
Why use Ansible? – Orchestration



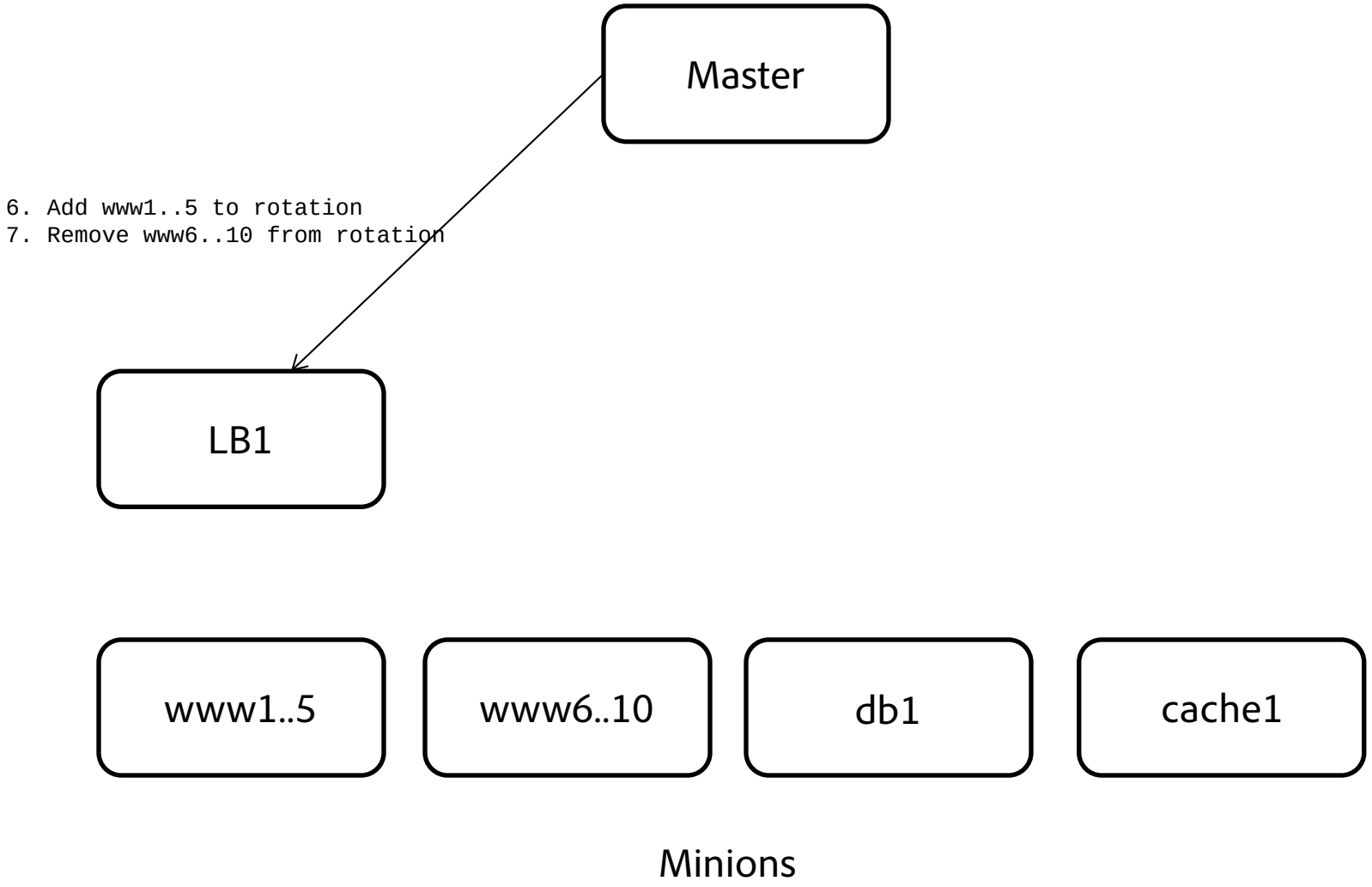
Why use Ansible? – Orchestration



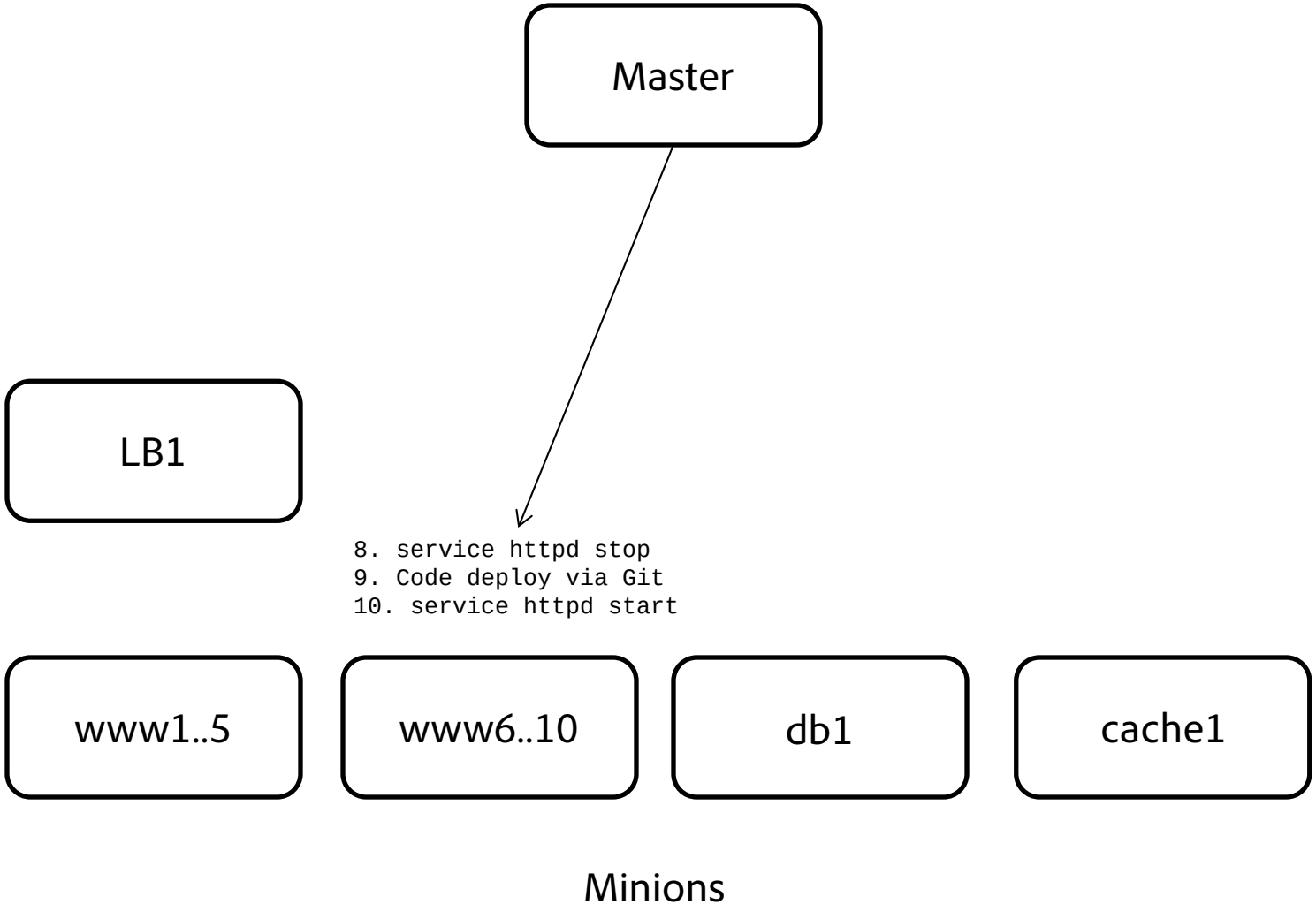
Why use Ansible? – Orchestration



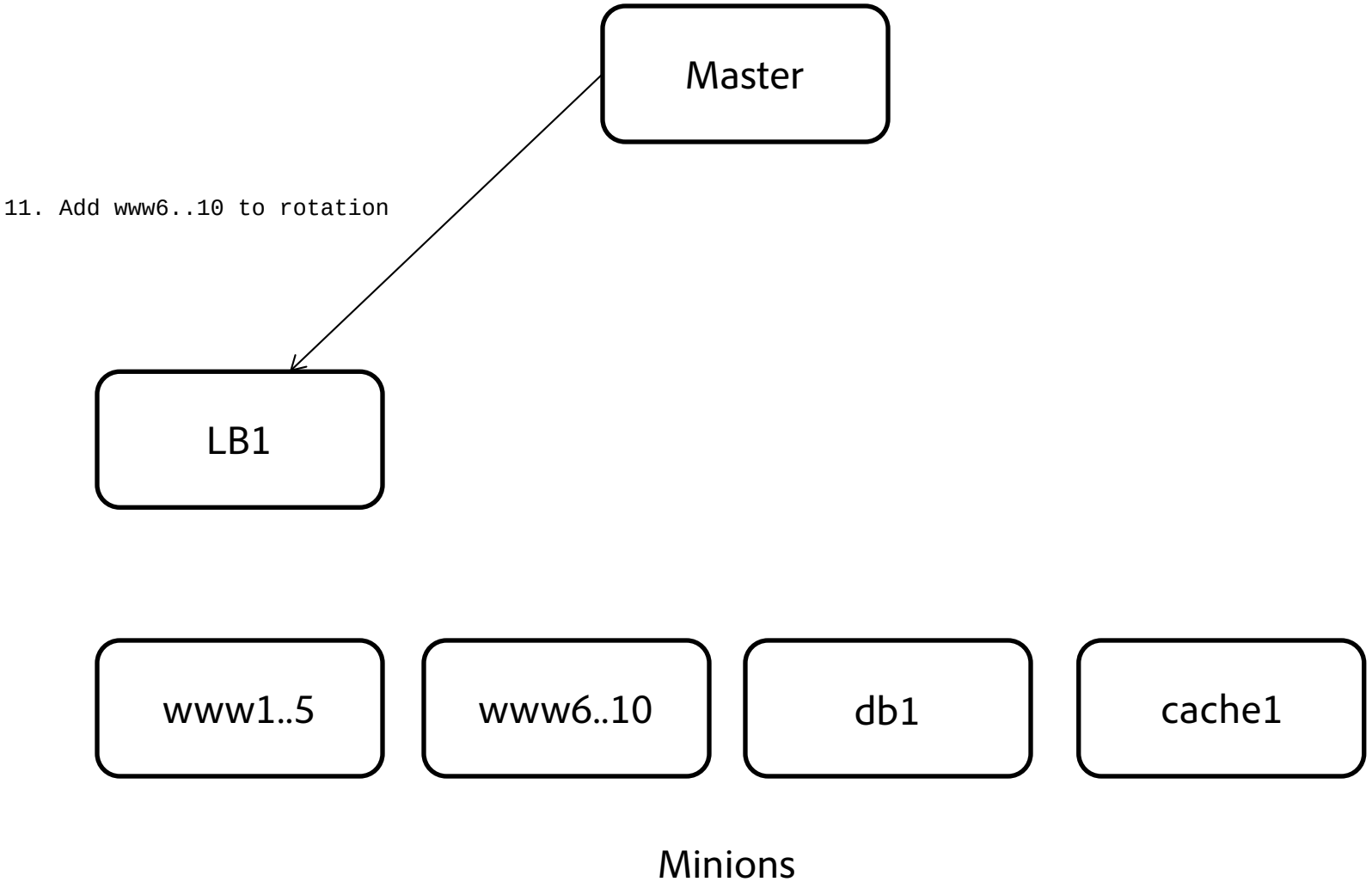
Why use Ansible? – Orchestration



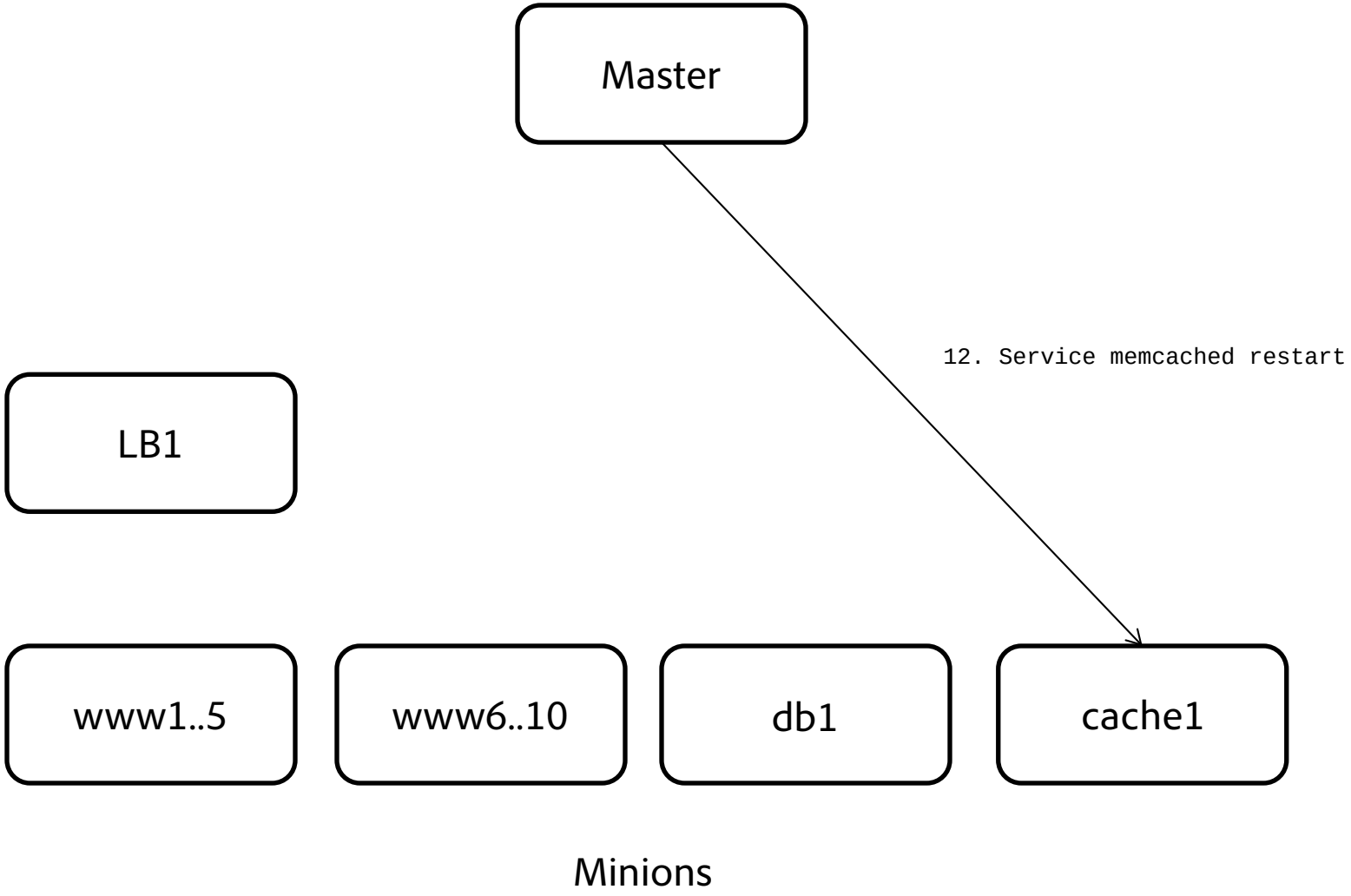
Why use Ansible? – Orchestration



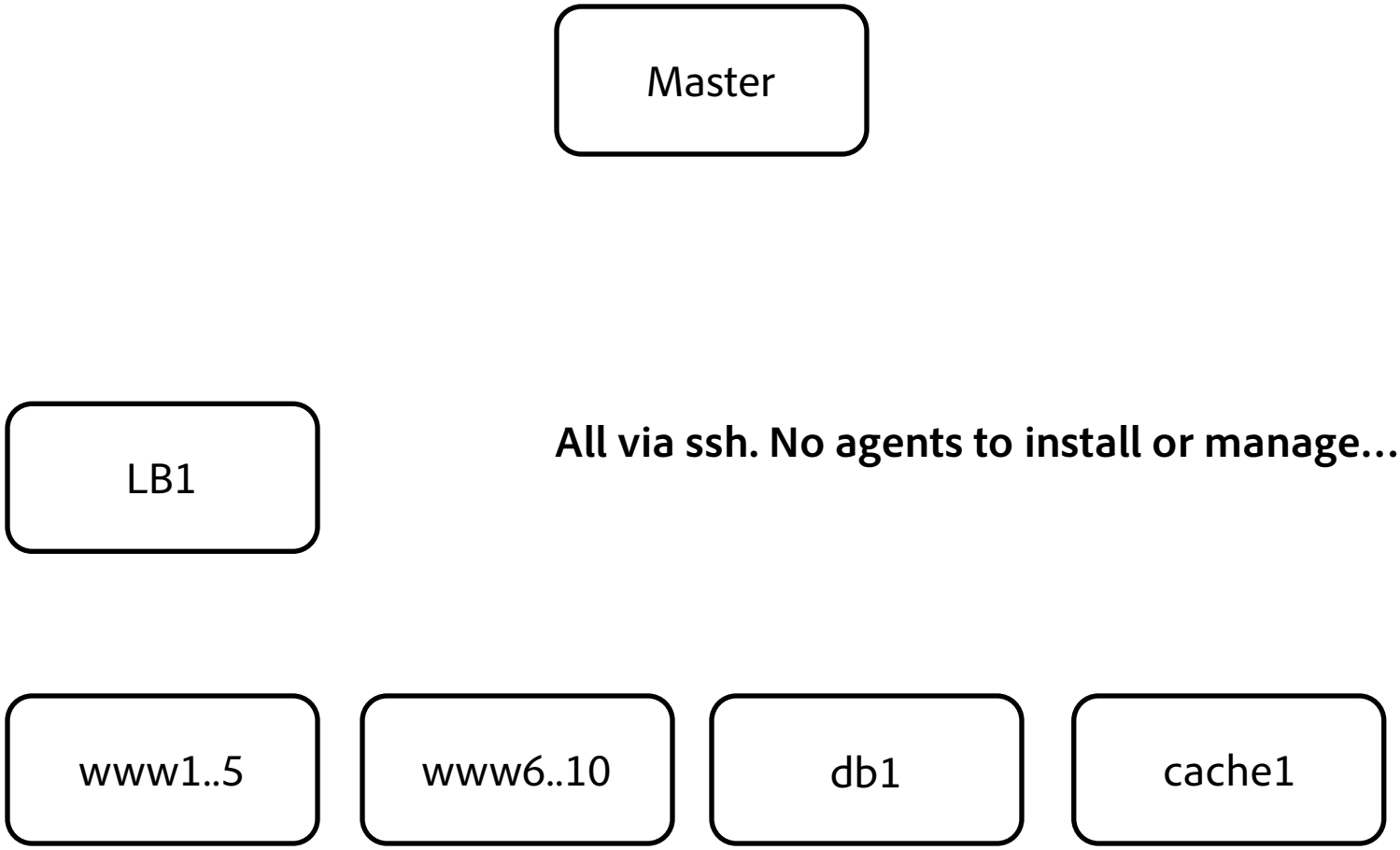
Why use Ansible? – Orchestration



Why use Ansible? – Orchestration



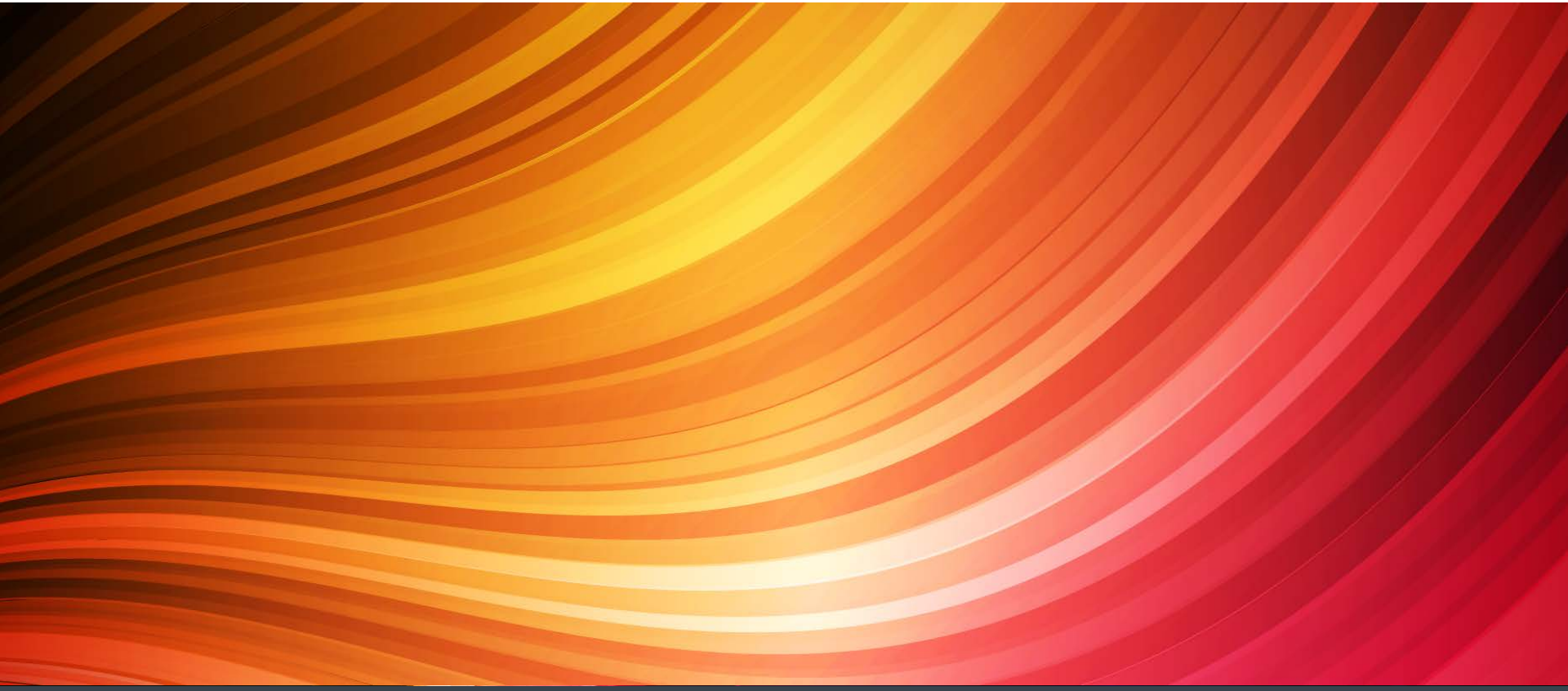
Why use Ansible? – Orchestration



Ansible Advantages

- SSH transport layer
  - Uses your existing SSH authentication infrastructure. No additional (and possibly questionable) authentication mechanisms / models to install.
  - No root access needed to run. Can use sudo, passwords, keys, different users, and any combination; whatever your ssh infrastructure looks like.
- No agents or daemons involved.
  - Fewer moving parts. Nothing to monitor.
  - Only have to install in one place. Upgrades are trivial. Easy to run out of a git checkout.
- Very readable and easy-to-understand configuration.
  - Your team will be up and running in minutes with Ansible.
  - Very low barrier to entry.
  - It can be as simple or as sophisticated as you want it to be.
  - Low semantic burden ;-P
- Very few dependencies to get up and running

Installing Ansible



Installing Ansible – Dependencies

- **On the master:**
  - python 2.6
  - paramiko
  - pyYAML
  - jinja2
  - Httplib2

- **On the minions:**
  - python 2.4
  - python-simplejson (if < python 2.5)
    - Though the ‘raw’ module doesn’t need that.

- Other modules may require more dependencies

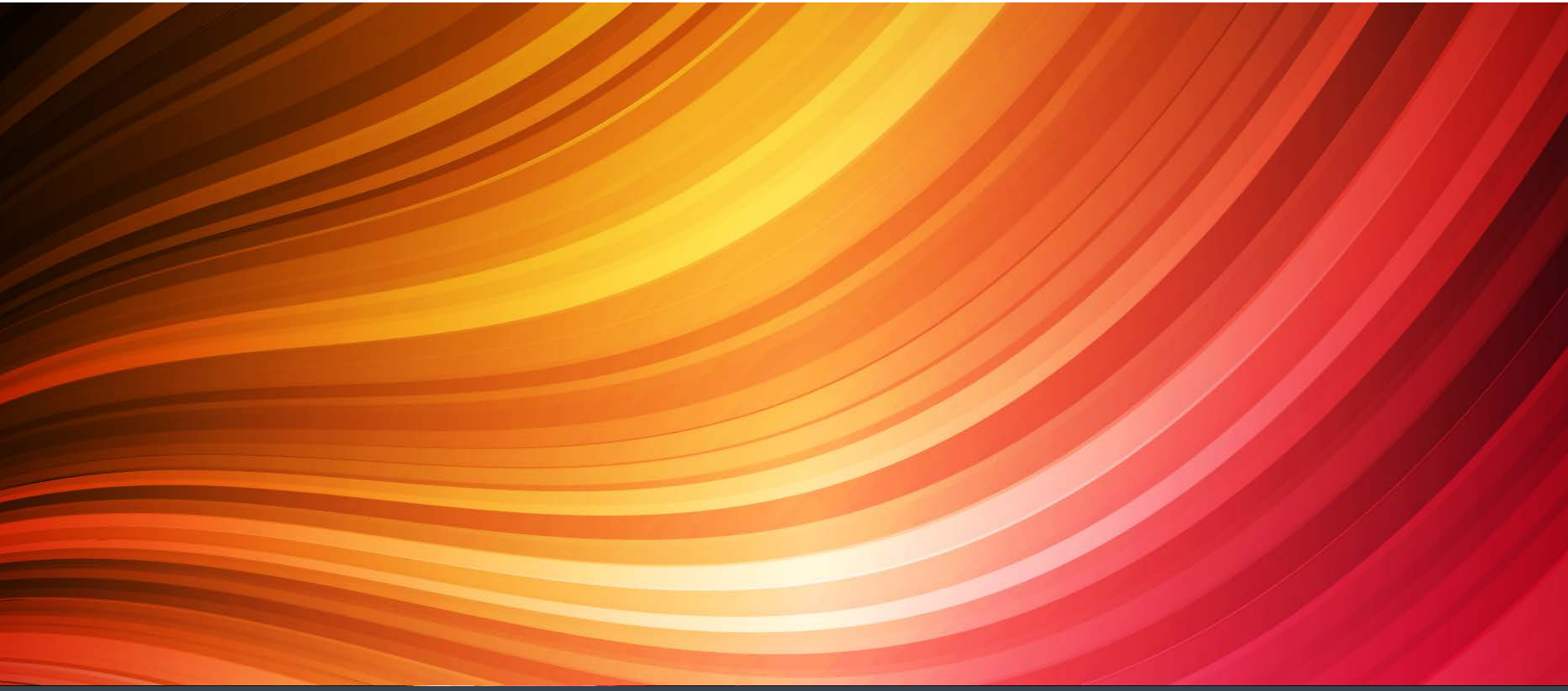
Installing Ansible

```
# Via Git
  $ git clone git://github.com/ansible/ansible.git
  $ cd ansible
  $ . hacking/env-setup

# - Yum, via EPEL
  sudo yum install ansible
# - RPM (Build your own)
  cd ansible
  make rpm
# - Apt (Ubuntu)
  sudo apt-add-repository ppa:rquillo/ansible
# - pkg (FreeBSD)
  sudo pkg install ansible
# Homebrew (MacOSX)
  brew install ansible
# Pip
  sudo pip install ansible
```

Installing Ansible

```
# And you’re done. No agents to install anywhere else.
# Everything happens (usually) over ssh
```

Ansible – Ad-hoc Invocation



Simple invocation – (Command and Control)

```
### ansible <host-pattern> [options]
### E.g.,
### ansible all –i <inv_file> -m <module> -a <arguments>

### Run uptime on all hosts
ansible all –i my_hosts –m shell –a uptime

### Run ansible w/o args for usage info (demo)

ansible all –i my_hosts –m shell –a uptime –s \
  –u other_user –U biguser –K

# -u = SSH as ‘other_user’
# -s = run commands with sudo…
# -U = sudo to ‘biguser’ before running
# -K = Before running, ask for the password sudo will ask
#      for
# And lots of other possibilities.
```

Simple invocation – (Command and Control)

```
### Run a command through a shell on all hosts
ansible all –i my_hosts –m shell –a \
  “grep MemTotal /proc/meminfo

### Run a command (not through a shell, no | or > < )
ansible all –i my_hosts –m command –a “/sbin/ldconfig”

### Run a raw command through ssh (does not go through the
### Ansible module system. Useful for running on routers
### (which don’t have Python installed on them)
ansible routers –i my_hosts –m raw –a “show int”

### Copy a local script to all boxes and run it
ansible all –i my_hosts –m script –a “/local/script.pl”
```

Inventory – Static File – Hosts and Groups

```
# In the inventory file you specify hosts, host groups,
# and variables to be associated with hosts and groups
<snip>
www.example.com

[web]
www1.corp.com
www2.corp.com
www3.corp.com
[mail]
mail1.corp.com
mail2.corp.com
mail3.corp.com
[ftp]
ftp[1..100].corp.com
</snip>

ansible web:ftp:mail –m shell –a ‘cat /etc/redhat-release’
```

Inventory – Static File – Host and Group Variables

```
### Host vars can be used to set connection attributes
www.example.com ansible_ssh_user=rob \
  ansible_python_interpreter=/usr/local/my_pyth/bin/python

### You can use host vars to make aliases
www ansible_ssh_host=www.example.com

[mail]
mail[1..10].example.com

[mail:vars]
ansible_ssh_port=5555
```

Inventory – Static File – Groups of groups

```
[us_east]
www[1..5].example.com
[us_west]
www[6..10].example.com

[us_all:children]
us_east
us_west

[emea]
www[11.15].example.com
[apac]
www[16..20].example.com

[global:children]
us_all
Emea
Apac
```

Inventory – Host and Group vars in files

```
# The preferred method of storing host and group variables
# is in separate files. If your inventory file in
# /home/me/ansible/hosts, then Ansible will also look for
# variables in
/home/me/ansible/group_vars/<group_name>
/home/me/ansible/host_vars/<host_name>

# These files are YAML formatted, and look like this:
---
some_variable: some_value
ansible_ssh_port: 1234

# More on the recommended directory layout later…
```

Selecting target hosts with patterns

```
# Once your inventory file is ready, you can select
# target hosts with patterns:
ansible <pattern> [options]

# '*' or 'all' – run against all hosts
ansible all –m yum –a 'name=httpd state=present'

# Specific hostname
ansible www.example.com –m copy –a 'src=/local/file \
  dest=/remote/file'

# Specific group
ansible us_west –m template –a 'src=/local/template.jnj \
  dest=/remote/httpd.conf'

# Use wildcards
ansible *.example.com –m unarchive –a 'src=/some/foo.tgz \
  dest=/remote/dir'
```

Selecting target hosts with patterns

```
# Use ‘:’ for OR (If the host is in us_west, or in emea)
ansible us_west:emea –m fetch –a ‘src=/etc/hosts \
  dest=/local/dir’

# ‘!’ to exclude
ansible us_all:!www1.example.com –m synchronize –a \
  ‘src=/local/var/www dest=/remote/var/www recursive=yes’

# ‘&’ for intersection – Hosts must be in db_hosts AND
# web_hosts
ansible db_hosts:&web_hosts –m seboolean –a \
  “name=httpd_can_network_relay state=yes”

# Longer combinations of the above left as an exercise for
# the reader

# ‘~’ for Regular expression matching
ansible ~(db|web_db)[5-9]* -m mysql –a \
  “name=widget_db present=yes”
```

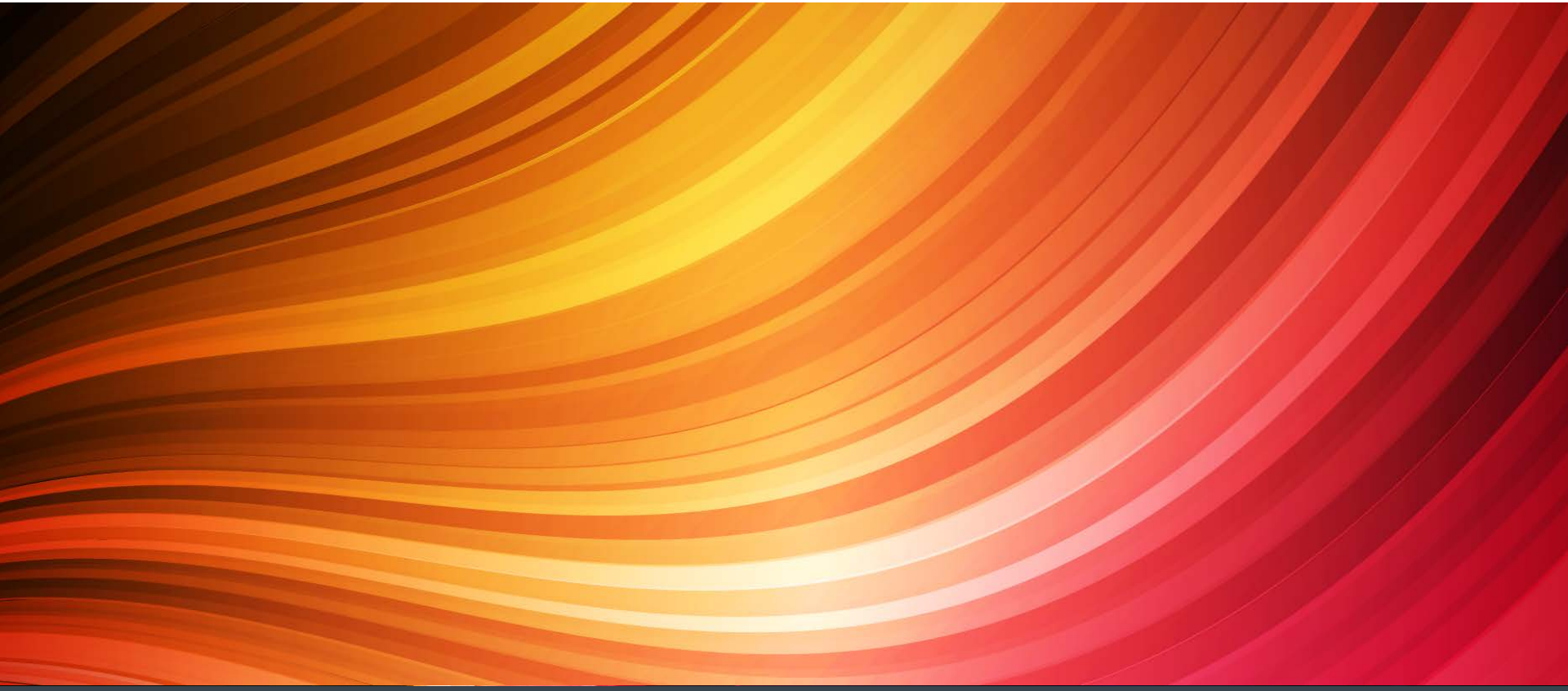
Invoking modules in ad-hoc invocations

```
# Basic idea:
# ansible <target> -m <module name> -a <module args>
# See http://docs.ansible.com/modules_by_category.html
# for the full (and growing) list of modules

# E.g.,
Ansible web –m git –a \
  ‘repo=git://githost.example.com/git/web_root.git \
   dest=/var/www
   version=release_2.0
   force=yes
   depth=1
   executable=/usr/local/my/git’

# LIVE DEMO!!!
```

Inventory - Dynamic

```
# Useful if you store host/group info in LDAP / Cobbler /
# EC2 / OpenStack / RackSpace / CMDB, etc.

# Example inventory scripts provided for Cobbler, EC2, GCE
# and several others.

# If the –i arg is an executable file, Ansible will run it
# for inventory info.

# If –i arg is a directory, Ansible will run al exes in
# the dir and combine their outputs.

# Ansible will first run the script with '--list', which
# should return a JSON dictionary of all groups / hosts / vars
# Ansible will then run the script with '--host <host>' for
# each host

# See the docs / examples for more details
```

Ansible – Playbooks – Configuration Management



Ansible Playbooks

```
# “Designed to be human-readable“

# Enable configuration management and “orchestration”

# See the "ansible-examples" directory for example playbooks

# A good idea to keep these in source control
#   Can build up a tree of reusable, modular parts and pieces
#   Doesn’t have to be one big, single file (though can be).

# Written in YAML

# Each playbook has a list of 'plays'
# Each play has a list of 'tasks'
# Plays are mapped to a group of hosts as a “role”.

# Invoked with the ansible-playbook command

# Basic example…
```

Ansible Playbooks – Example ‘play’

```
---
- hosts: dns-servers
  tasks:
  - name: Ensure BIND is installed
    yum: pkg=bind-9.5.4 state=present

  - name: Put the BIND config in place
    template:
      src=/templates/named.conf.j2
      dest=/etc/named.conf
      validate=/usr/sbin/named-checkconf

  - name: Copy BIND zones into place
    synchronize:
      archive=yes
      delete=yes
      dest=/var/named
      src=/files/var/named
      rsync_path="sudo rsync“

  - name: Make sure BIND is running, and starts at boot
    service:
      name=named
      enabled=yes
      state=started
```

Ansible Playbooks – Example ‘play’

```
---
- hosts: dns-servers                <-- Can use ‘patterns’ here as with ad-hoc
  tasks:
  - name: Ensure BIND is installed
    yum: pkg=bind-9.5.4 state=present

  - name: Put the BIND config in place
    template:
      src=/templates/named.conf.j2
      dest=/etc/named.conf
      validate=/usr/sbin/named-checkconf

  - name: Copy BIND zones into place
    synchronize:
      archive=yes
      delete=yes
      dest=/var/named
      src=/files/var/named
      rsync_path="sudo rsync“

  - name: Make sure BIND is running, and starts at boot
    service:
      name=named
      enabled=yes
      state=started
```

Ansible Playbooks – Example ‘play’

```
---
- hosts: dns-servers
  tasks:
  - name: Ensure BIND is installed      <-- Human-readable task names
    yum: pkg=bind-9.5.4 state=present

  - name: Put the BIND config in place
    template:
      src=/templates/named.conf.j2
      dest=/etc/named.conf
      validate=/usr/sbin/named-checkconf

  - name: Copy BIND zones into place
    synchronize:
      archive=yes
      delete=yes
      dest=/var/named
      src=/files/var/named
      rsync_path="sudo rsync“

  - name: Make sure BIND is running, and starts at boot
    service:
      name=named
      enabled=yes
      state=started
```

Ansible Playbooks – Example ‘play’

```
---
- hosts: dns-servers
  tasks:
  - name: Ensure BIND is installed
    yum: pkg=bind-9.5.4 state=present

  - name: Put the BIND config in place    <-- Tasks are each just a module
    template:                                 invocation. Tasks run
      src=/templates/named.conf.j2            sequentially in each play.
      dest=/etc/named.conf                    and are designed to be idempotent
      validate=/usr/sbin/named-checkconf

  - name: Copy BIND zones into place
    synchronize:
      archive=yes
      delete=yes
      dest=/var/named
      src=/files/var/named
      rsync_path="sudo rsync“

  - name: Make sure BIND is running, and starts at boot
    service:
      name=named
      enabled=yes
      state=started
```

Ansible Playbooks – Play-level and task-level options

```
---
-  hosts: dns-servers
   remote_user: named  # User to run the play as (also can define per-task)
   # Other play-level options, variables go here
   tasks:

   - name: Ensure BIND is installed
     # Task-level options go here
     yum:
     ...

-  hosts: web-servers # Start the next 'play' here…
   ...



### And we can start to see how we might orchestrate our entire infrastructure…
### We can make certain playbooks for configuration management, and others for
### system processes, deployments, maintenance tasks, etc.
### As well as continue to use it for ad-hoc commands
```

Ansible Playbooks – Facts and Variables

```
# Variables can be defined in Inventory:
[db-servers]
db1.example.com ansible_ssh_port=2222

[db-servers:vars]
git_host=git1.site.example.com

# Or in Playbooks:
-hosts: memcached-servers
 vars:
   memcached-port: 11214
   max_con: 100

# Or inside included Files / Roles
/etc/ansible/inventory
/etc/ansible/host_vars/www1.example.com
/etc/ansible/group_vars/db-servers
/etc/ansible/group_vars/web-servers
```

Ansible Playbooks – Facts and Variables

```
# Variables also come from discovered facts
ansible <target> –m setup

# You can put your own facts in /etc/ansible/facts.d
# Format files here in
#   INI format,
#   or JSON format,
#   or JSON-generating executables

# Pass in vars from the command-line (K,V, quoted JSON, YAML)
Ansible –-extra-vars “var1=value1 var2=value2”
Ansible --extra-vars “@var_file.json”
Ansible --extra-vars “@var_file.yml”

# Variables also come from facter or ohai, if installed
# They’re prefixed with facter_ and ohai_, respectively.
```

Ansible Playbooks – Facts and Variables

```
# Register variables (variables from command-output)
tasks:
  shell: /usr/bin/some_command.pl
  register: some_variable
  # Now available as {{ some_variable }} elsewhere in the play

# Jinja2 filters can be applied to variables
# As well as others provided by Ansible. E.g.,
{{ my_var | to_nice_json }}
{{ my_var | to_nice_yaml }}
{{ list | unique }}
{{ list | union(other)list }}
{{ path | basename }}
# Lot's of other useful stuff here
```

Ansible Playbooks – Ansible Vault (new in 1.5)

```
# Allows you to store sensitive bits (variables) in encrypted
# files, host vars, group vars, even task lists...

# Create a new encrypted file
# Will launch $EDITOR and allow you to edit the content
ansible-vault create my_secret_vars.yml

# Later edits to the file
ansible-vault edit my_secret_vars.yml

# Encrypt an existing file
ansible-vault edit my_other_vars.tml

# Decrypt an encrypted file
ansible-vault decrypt vars.yml

# Use a playbook which references the encrypted vars
ansible-playbook --ask-vault-pass
```

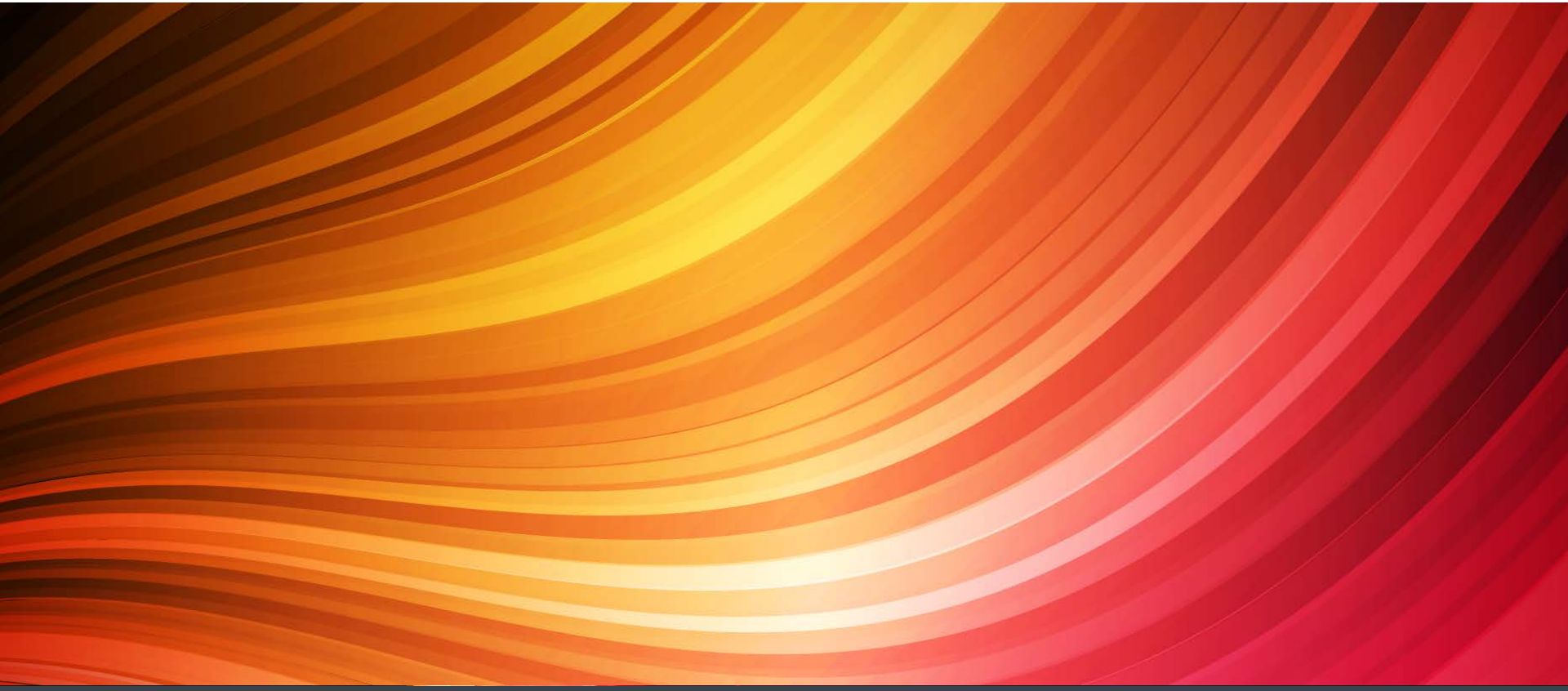
Ansible Playbooks – DEMO!

```
# Your prayers appreciated...
```

Ansible Playbooks – Homework

```
# Conditionals
# Loops
# Includes
```

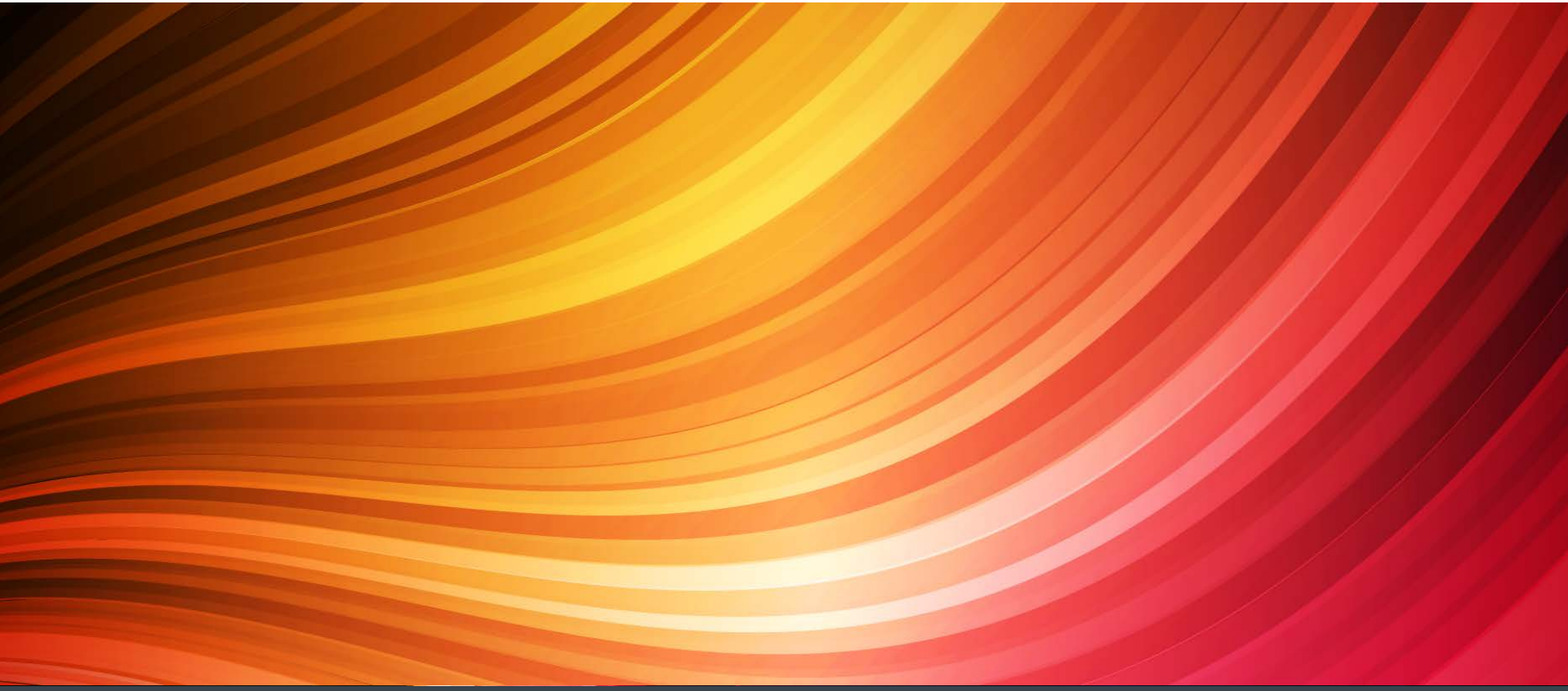
Ansible – Playbooks – Full Orchestration



Ansible Playbooks – Full-on orchestration!

```
# Update the DB schema
# For each web server (5 at a time)
  # Turn monitoring off
  # Pull it from the load-balancer
  # Do a git update of the web root
  # Run a test script
  # Put it back in the load-balancer
  # Turn monitoring on
# Restart memcached on each memcached server, one at a time.



# We'll look at a playbook, but
# DEMO left as an exercise for the reader...
```

Ansible – Other stuff



Recommended Directory layout

http://docs.ansible.com/playbooks_best_practices.html#directory-layout

Extending Ansible

- Anisble API
  - Build applications using Ansible as a library
- Dynamic Inventory Sources
- Modules
- Plugins
  - Connection types (ssh, paramiko, etc.,)
  - Vars, Filters, Lookups, etc.

Thank you!

- Daniel Hanks
- Slides at www.brainshed.com
- @danhanks
- danhanks@gmail.com
  - Feel free to reach out if you have further questions.


Adobe