Master of Puppets

Infrastructure as Code and Configuration Management on an Open Source Budget

Mike Renfro (renfro@tntech.edu)

Tennessee Tech University

2023-04-24

What's the Problem, and Why Should I Care?

1. The number of services we manage is growing faster than our headcount.
2. Manual configuration leads to manual errors and doesn't scale.
3. Some automation artifacts (e.g., golden images or VM templates) may lack reproducibility (often due to uncrecorded manual changes).
4. Local, sandboxed development environments may be preferred to reduce iteration time and risk to production systems.

The Tradeoffs

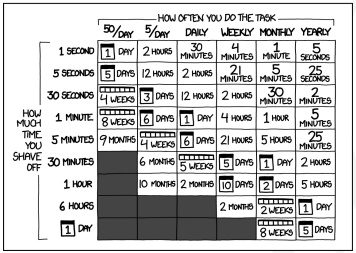

Figure 1: XKCD 1205, "Is It Worth the Time?"

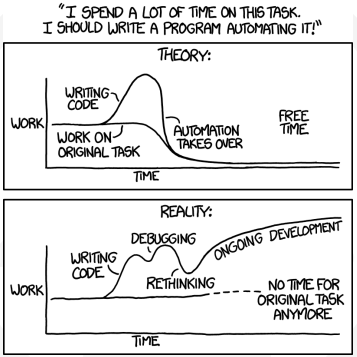

Figure 2: XKCD 1319, "Automation"

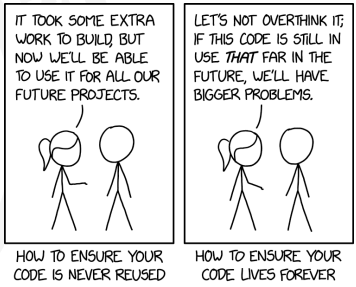

Figure 3: XKCD 2730, "Code Lifespan"

Minimum Standards for a Viable Infrastructure as Code (IaC) Solution

1. For any given service, define a single source of authority for:
   - installed packages
   - configuration files
   - running services
   - firewall rules
   - etc. with customization allowed for groups of servers.
2. Automatically apply *all* needed changes, but *only* when needed.
3. Maintain balance of consistency and separation of dev/test/prod environments.
4. Automatically maintain records of who made what change when (and ideally, why).
5. Prefer text over binaries (automation for base OS install instead of golden thick image or VM template).
6. Enable developers to test safely and minimize exposure to outside network.

Stretch Goals for a Viable IaC Solution

1. Allow multiple dev/test environments.
2. Give admins their choice of development platform (Windows, macOS, Linux).
3. Enable management of multiple server OSes (at least multiple Unix, or possibly Windows).
4. Manage endpoints as well as servers.
5. Secure and track secrets (e.g., local database passwords) in central location.
6. Be a good neighbor on already-installed systems (only manage what has to be) and expand scope from there.
7. Avoid vendor lock-in.

This Isn't the Only Possible Solution

- Some tools used here are derived from our production environment.
- Other tools are ones I've used or promoted in other projects and contexts.
- These tools provide a working reference implementation that's cross-platform (if not totally cross-architecture) with zero purchasing price and open-source licensing.
- Replace any of them with other tools matching your local preferences and standards (the concepts are unchanged).

Provisioning (1/2)

Oracle VM VirtualBox *without Extension Pack* (GNU General Public License v2) Type 2 hypervisor for x86 platforms.

- Runs on Windows, Linux, and macOS (M1/M2 is in developer preview, and hasn't been tested for this application).
- Extension Pack is not open source, and use requires separate license from Oracle.

VirtualBox Typical Use for Client Testing

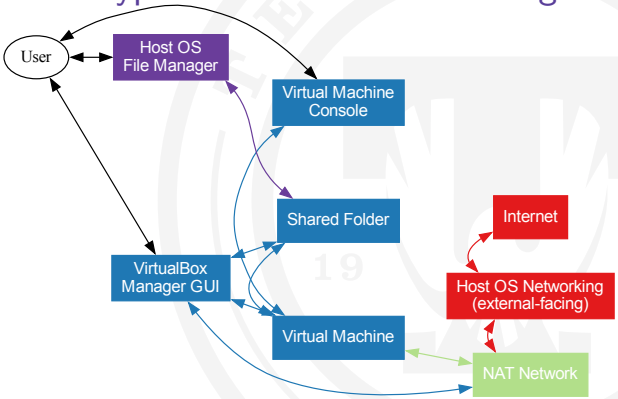

Figure 4: Typical Use of VirtualBox on Desktop

Easy, and works fine for testing new OSes or applications.

VirtualBox Less Typical Use for Server Testing

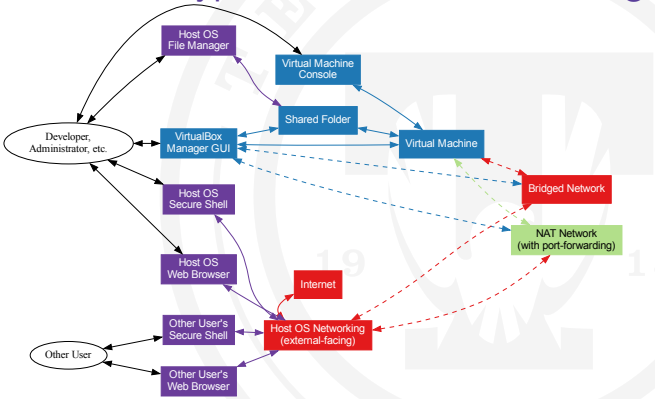

Figure 5: Less Typical Use of VirtualBox on Desktop

Easy? Maybe, but management effort scales with number of virtual machines, and the easy way to get access from your host OS tools (*e.g.*, web browser) either requires manual port forwarding or exposing the VMs to your network.

Provisioning (2/2)

HashiCorp Vagrant (MIT License) Provisioning software for virtual machines.

- Supports programmatic creation of virtual machines and networks.
- Supports in-VM provisioning via file copy, shell script, Ansible, CFEngine, Chef, Docker, Podman, Puppet, and Salt.

What's Vagrant Doing?

A Vagrant VM is just a VirtualBox VM that:

- runs in the background (headless)
- is usually derived from a base installation from https://app.vagrantup.com/
- configured with a Ruby-syntax `Vagrantfile`
- usually supports a shared folder `/vagrant` mapped from the host OS
- supports ssh from the host operating system through an automatically-forwarded port (via `vagrant ssh`)

What's the Difference with Vagrant for a Single Virtual Machine?

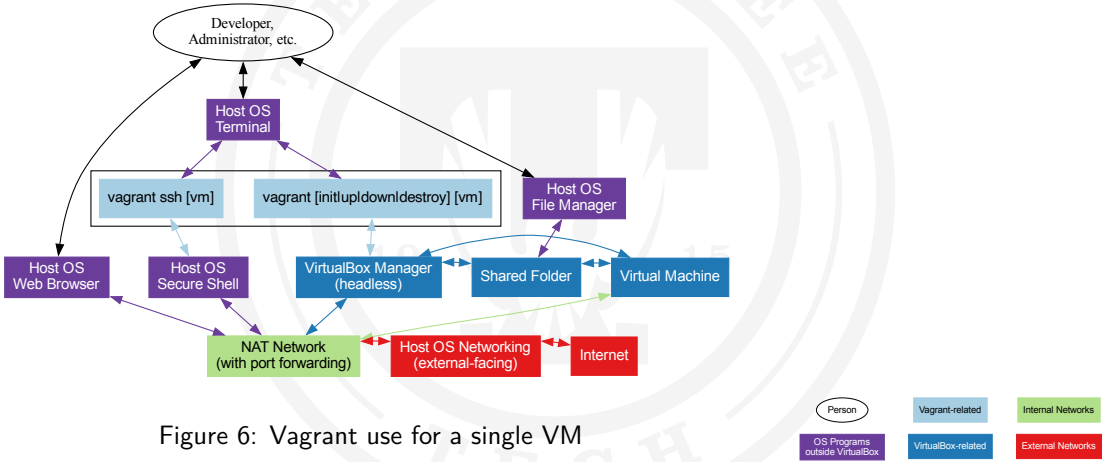

Figure 6: Vagrant use for a single VM

How Does It Scale to Multiple Virtual Machines?

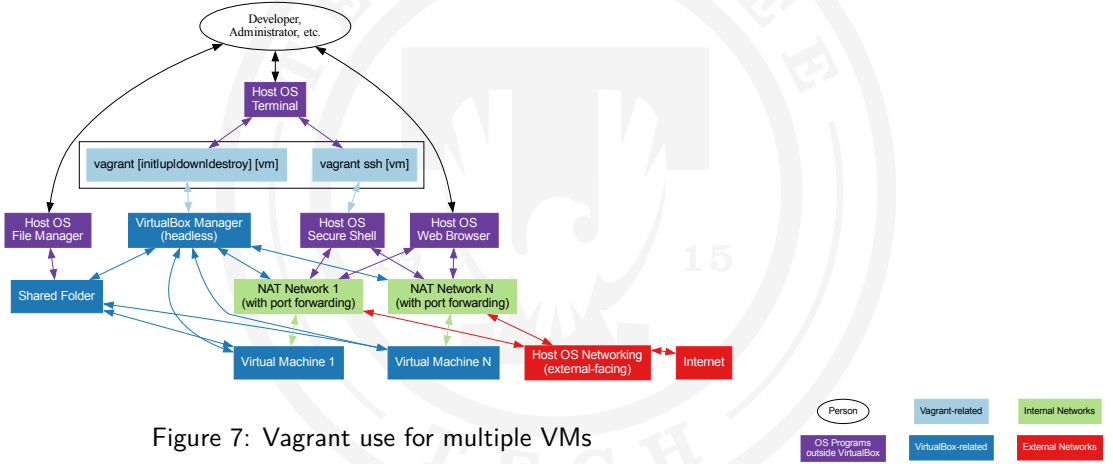

Figure 7: Vagrant use for multiple VMs

What If My Programs Can't Use Alternate Ports?

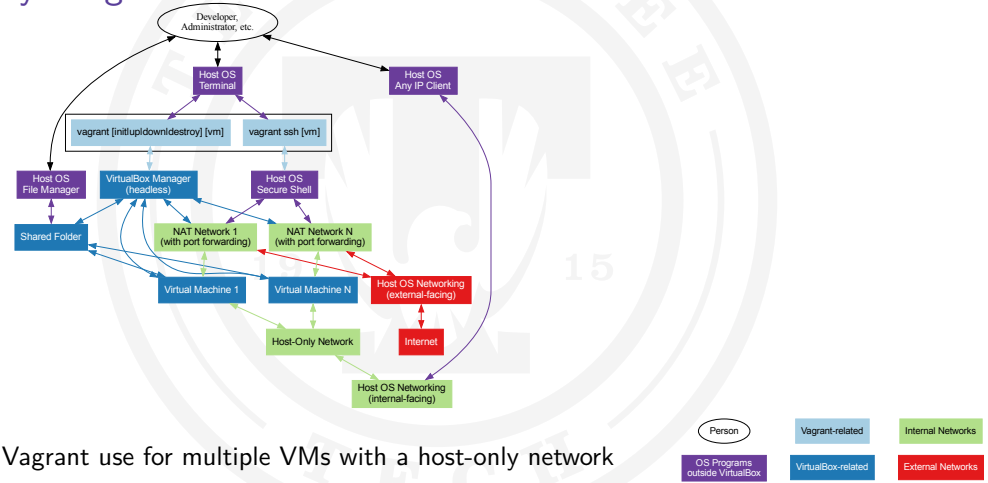

Figure 8: Vagrant use for multiple VMs with a host-only network

Toolchains

Git (GNU General Public License v2) Keeps track of and logs changes to files in folders. Allows multiple concurrent branches of development, and can push and pull code to/from remote servers (usually via `ssh`).

Puppet Agent (Apache License) Primarily needed for secret-management tools. Service can be stopped and disabled.

Puppet Development Kit (Apache License) Helper tools for developing and testing Puppet modules and classes.

Editing

Microsoft Visual Studio Code (MIT License) 800 pound gorilla of text editors, `emacs` for a new generation.

Puppet VSCode Extension (Apache License) Provides syntax highlighting, code completion, and linting of Puppet code. Integrates with Puppet Development Kit.

Configuration Management

Puppet (primary server and agent) (Apache License) a tool that helps you manage and automate the configuration of servers.

- Code is declarative, describing the desired state of your systems, not the sequence of steps needed to get there.
- Puppet primary server stores the code defining your desired state, and compiles it with facts provided by the agent into a catalog.
- Puppet agent translates the compiled catalog into host-specific commands and executes them.
- Run the agent on the Puppet primary server to have it define its own configuration.

Version Control Server

Gitea (MIT License) self-hosted Git server, features:

- single Go binary with SQLite support.
- issue tracking and wikis.
- organizational hierarchy.
- OpenID Connect single sign-on (yes, it works with Azure Active Directory).
- branch protection and review before merge.
- webhooks to trigger automation on various events.

Continuous Deployment

Adnan Hajdarevic (adnanh)'s Webhook (MIT License) lightweight configurable tool written in Go, that allows you to easily create HTTP endpoints (hooks) on your server, which you can use to execute configured commands.

r10k (Apache License) provides a general purpose toolset for deploying Puppet environments and modules. Maps a branch in a Git repository to a Puppet environment.

Combining Git branches, Gitea webhooks, adnanh's Webhook, and r10k allows easy management of multiple Puppet environments for developing and testing services.

VS Code, Puppet VS Code Extension, VirtualBox, Vagrant, Puppet Agent, Puppet Development Kit

Use default settings for all.

Git

- Windows, macOS, Linux installation: you should be able to follow Software Carpentry's template workshop instructions for installation.
- Windows, macOS, Linux setup: following with Software Carpentry's Setting Up Git page, opening a command prompt and running:
  - `git config --global user.name "Your Name"`
  - `git config --global user.email "you@yourplace.edu"`
  - and either:
    - `git config --global core.autocrlf input` for macOS and Linux, or
    - `git config --global core.autocrlf false` for Windows

  should be enough to get started.

Git in Visual Studio Code

- File / Open Folder
- Find existing folder, or create empty one.
- View / Source Control
- Initialize Repository button

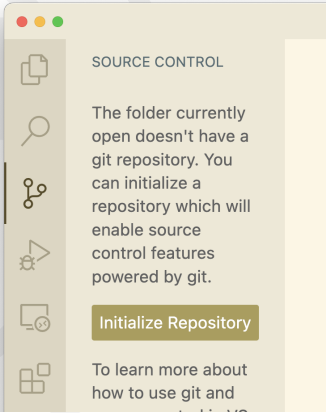

Figure 9: VS Code Initialize Repo button

Terminal in Visual Studio Code

- View / Terminal

Places you at the top-level Git repository folder.

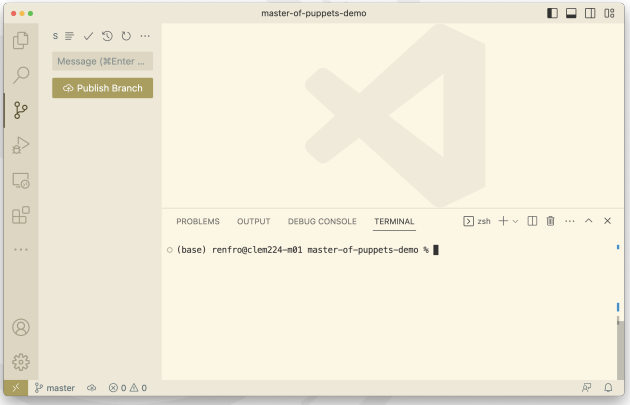

Figure 10: VS Code terminal window

How To Make the First Vagrant VM?

We want a VM:

- running a Red Hat-family OS, version 8
- supporting a shared folder `/vagrant` mapped from the host OS

Start the definition of a new Vagrant VM in the repository folder with `vagrant init bento/rockylinux-8`, and look at the `Vagrantfile` that was just created.

(The `bento` Vagrant boxes are built by the Chef Bento project.)

The First Vagrantfile (1/2)

Generated from `vagrant init bento/rockylinux-8`, filtered down to interesting commands and comments.

```
Vagrant.configure("2") do |config|
  config.vm.box = "bento/rockylinux-8"
  # config.vm.network "forwarded_port", guest: 80, host: 8080
  # config.vm.network "forwarded_port", guest: 80, host: 8080,
  #   host_ip: "127.0.0.1"
  # config.vm.network "private_network", ip: "192.168.33.10"
  # config.vm.provider "virtualbox" do |vb|
  #   # Customize the amount of memory on the VM:
  #   vb.memory = "1024"
  # end
```

The First Vagrantfile (2/2)

```
  # config.vm.provision "shell", inline: <<-SHELL
  #   apt-get update
  #   apt-get install -y apache2
  # SHELL
end
```

You'll also see a 1 badge on the Source Control button in Visual Studio Code, indicating there's one pending change in the repository folder (the creation of the `Vagrantfile`).

Installing a New Vagrant VM with `vagrant up`

From the Visual Studio Code terminal: - Running `vagrant up` builds the VM. - Running `vagrant ssh` results in you being logged into a `vagrant` account in the Linux VM, which has passwordless `sudo` rights. - If you exit back out to your host command prompt, you can do a `vagrant destroy` to shut down and delete the VM.

Once that's working, we'd like to record the new Vagrantfile into version control in Visual Studio Code, but there are complications.

Setting Up Version Control (How to Ignore Machine-Generated Files)

- View / Source Control

Notice there are other files in the repository folder now (the Source Control button probably has a badge of 3 now). These are from the `.vagrant` folder that Vagrant uses to track virtual machine information.

- Right-click one of the files, select "Add to .gitignore"
- Notice the file is absent from the Source Control button, and there's a new file `.gitignore`.
- Go back to the folder view (View / Explorer)
- Select the file `.gitignore` to open it in the editor
- Edit the only line in `.gitignore` to be just `.vagrant/` (no filename) and save the file.

Setting Up Version Control (Adding and Committing Useful Files)

- Notice the Source Control badge now reads 2 (one for `Vagrantfile`, one for `.gitignore`).
- Select both `Vagrantfile` and `.gitignore`, select "Stage Changes".
- In the "Message" text entry, enter `Define initial Vagrantfile and .gitignore` and select the "Commit" button.

In theory, Git commits should be "atomic", *i.e.*, a single, complete unit of work that can be described in a single sentence. In practice, we're often not that disciplined about it. Git commit messages should be short and imperative, completing the sentence, "when applied, this commit will . . . ".

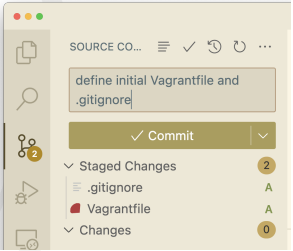

Figure 11: Readying first Git commit

What Would We Like to Change?

Things to Fix

1. VM hostname is currently `localhost`, would like that to change.
2. Need some actual configuration (packages, config files, services, etc.)
3. Need *multiple* VMs (Git server, Puppet primary server, Puppet client, etc.)
4. Need Puppet agent to poll Puppet primary server for changes (requires name resolution from DNS, host file, etc.)

Tools for Fixing Things

1. `Vagrantfile` settings
2. VM provisioners (`shell` shown by default, but `puppet` provisioner also available)
3. Provisioners can also read from files (shell scripts, Puppet manifests)
4. Files in the repository folder (show up in `/vagrant` in the VMs)
5. DRY (don't repeat yourself) principle (avoid copy/paste)

Reference Architecture

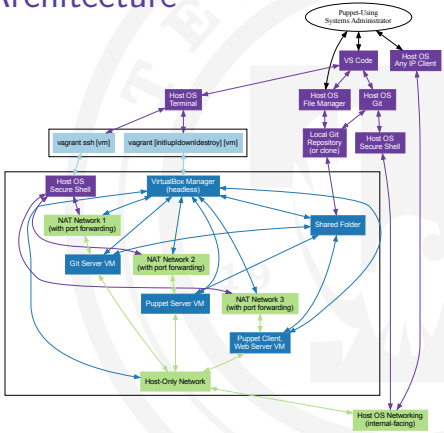

Figure 12: Reference architecture

- Administrator only directly interacts with VS Code, host file manager, and any host IP clients (*e.g.*, web browser)
- VMs communicate over shared internal network

Minimum Viable IaC Part 1: Bootstrapping a Git Server

- Vagrant allows for multiple provisioning blocks in the Vagrantfile.
- We'll use the `shell` provisioner to install the Puppet agent in each VM (later, we'll let the `puppet` provisioner to do the rest of the setup in each VM).

Initial Vagrantfile for Git server

```
Vagrant.configure("2") do |config|
  # general settings for all VMs
  config.vm.box = "bento/rockylinux-8"
  config.vm.provision "shell", path: "shell/provision.sh"
  # settings specific to the git VM
  config.vm.define "git" do |git|
    git.vm.hostname = "git"
    git.vm.network "private_network", ip: "10.234.24.2",
      netmask: "255.255.255.0"
  end
end
```

Contents of shell/provision.sh

```
#!/bin/bash
# Ensure working local DNS
nmcli con modify 'eth0' ipv4.dns-search 'theits23.renf.ro' \
  ipv4.ignore-auto-dns no ipv4.dns '10.234.24.254'
systemctl restart NetworkManager
YUM="yum -q -y"
# Install puppet
${YUM} install http://yum.puppet.com/puppet7-release-el-8.noarch.rpm
${YUM} install puppet-agent
```

Build Git Server, Verify Puppet Exists, Then Commit Changes

At the host terminal: - `vagrant up git` to build - `vagrant ssh git` to log in - `sudo -i puppet --version` to see Puppet is installed - `exit` to log out

In VS Code:

- View / Source Control
- add `Vagrantfile` and `provision.sh` to the staged changes
- commit changes with message `Define initial Git server and install puppet agent`

Useful Puppet Resource Types (Most Common in **Bold**)

- Command execution: **`exec`**, **`cron`**
- File-related: **`file`**, `filebucket`, **`mount`**, `tidy`
- Package management: **`package`**, **`yumrepo`**
- SELinux: **`selboolean`**, `selmodule`
- Services: **`service`**
- User-related: **`group`**, **`ssh_authorized_key`**, **`user`**
- Manifest structure: **`notify`**, `resources`, `schedule`, `stage`

- Resources are (usually) cross-platform, and are implemented through lower-level *providers* that are OS/platform-specific (*e.g.*, `dnf`, `yum`, `apt`, etc. for packages).
- Other resource types can be written in Ruby if needed, but it's not often you'll need to write one.

Use Existing Puppet Modules Where Feasible

https://forge.puppet.com/ has 1200+ modules for Puppet 7:

- some written and supported by PuppetLabs
- some supported by Puppet user community (aka the Vox Pupuli)
- some by individual users or companies

Bootstrapping Git Server Configuration in Puppet (1/5)

The choices to install/configure Gitea

Map install guide to low-level resources

- make users
- install packages
- download, extract archives
- create folders, set permissions
- create services
- edit config files, restart services

Use Puppet forge module

- work within documented APIs
- figure out when default settings aren't appropriate
- figure out if there's a documented API to adjust those settings
- manage module dependencies

Either way, in VS Code, make a new file `puppet/default.pp` and add:

```
node 'git.theits23.renf.ro' {
}
```

Bootstrapping Git Server Configuration in Puppet (2/5)

Going for the Puppet Forge version—inside the node entry, add:

```
$ip = $::facts['networking']['interfaces']['eth1']['ip']
$net = $::facts['networking']['interfaces']['eth1']['network']
class { 'gitea':
  ensure   => '1.18.5',
  checksum =>
    '4766ad9310bd39d50676f8199563292ae0bab3a1922b461ece0feb4611e867f2',
  custom_configuration => {
    'server'  => { 'ROOT_URL' => "http://${ip}:3000/",
      'SSH_DOMAIN' => $ip, 'DOMAIN' => $ip, },
    'webhook' => { 'ALLOWED_HOST_LIST' => "${net}/24", }
  },
}
```

Where Do We Get The Gitea Class? (3/5)

- Once the Puppet agent is installed in a VM, we have access to the `puppet module` command to install Forge modules.
- Those are shell commands, so they're `shell` provisioner lines in the Vagrantfile.
- To reduce the copy/paste, we can write a Ruby function in `ruby/install_mod.rb` to generate shell commands to install modules.

```
# "Installing a puppet module from a manifest script",
# https://stackoverflow.com/a/25009495
def install_mod(name, version, install_dir = nil)
    install_dir ||= '/etc/puppetlabs/code/modules'
    "mkdir -p #{install_dir} && " \
    "(puppet module list | grep #{name}) || " \
    "puppet module install -v #{version} #{name}"
end
```

Where Do We Get The Gitea Class? (4/5)

Add a line to the top of the `Vagrantfile`:

```
require './ruby/install_mod'
```

Add two lines under the Git network settings line to install the Gitea module and run the Puppet provisioner:

```
git.vm.provision "shell", inline: install_mod('h0tw1r3-gitea', '2.0.0')
git.vm.provision "puppet", manifests_path: "puppet"
```

Bootstrapping Git Server Configuration in Puppet (5/5)

- At host terminal, run `vagrant provision git --provision-with puppet`
- If this fails due to the Vagrantfile having changed while the VM was running, run `vagrant reload`.
- Watch Puppet download, install, and configure Gitea.
- At host terminal, re-run `vagrant provision git --provision-with puppet`
- Watch Puppet determine no further changes need to be made.
- Add and commit the changes to `Vagrantfile` and `default.pp`.

Final Configuration of Gitea Through the Web

- Head to http://10.234.24.2:3000/ in the host browser to create an administrator account in Gitea.
  - Administrator Username: gitadmin
  - Password: (anything at least 6 characters)
  - Email Address: (anything)
- On the host, generate an ssh key with `ssh-keygen -t ed25519`, then `cat ~/.ssh/id_ed25519.pub` (if you already have an ssh public key, you can cat it instead).
- Copy/paste the public key content into http://10.234.24.2:3000/user/settings/keys.

Saving a Copy of the Vagrant repository in Gitea

In Gitea web interface:

- Create new organization `theits23` to hold repositories.
- Create new, uninitialized repository `iac-project` in the `theits23` organization.

In VS Code window:

- View / Source Control
- "3 dots" button above the commit message box / Remote / Add Remote
- URL: `git@10.234.24.2:theits23/iac-project.git`, Name: `origin`
- Click the Publish Branch button

Now every time you make a commit, you'll be able to push that commit to the remote repository.

Minimum Viable IaC Part 2: Bootstrapping a Puppet Primary Server

Repeat the same procedure to make a new Vagrant VM for Puppet. Realistically, a Puppet server needs at least 4 GB RAM, and extra cores don't hurt. Add the following to the `Vagrantfile`, right below the Git server definition:

```
config.vm.define "puppet" do |puppet|
  puppet.vm.provider "virtualbox" do |vb|
    vb.cpus = "2"
    vb.memory = "4096"
  end
  puppet.vm.hostname = "puppet"
  puppet.vm.network "private_network", ip: "10.234.24.3",
    netmask: "255.255.255.0"
end
```

Bootstrapping Puppet Server Configuration in Puppet (1/N)

Before we go tearing into more configuration files with the first thing that could possibly work, let's consider what we need the Puppet primary server to do:

1. Run a `puppetserver` service where other systems can pull their settings from
2. Pull those settings from a repository in Gitea
3. Run a `webhook` service so the Git server can notify that new settings are available
4. Deploy code from a Git repository using `r10k`

So if we stick with the Puppet Forge architecture, we'll need to find modules to handle each of those.

Finding Puppet Modules for The Puppet Server (2/N)

- Don't search Puppet Forge for "puppet".
- "puppetserver" works a bit better, and includes a recent module from The Foreman lifecycle management project.

Add

```
puppet.vm.provision "shell", \
  inline: install_dep('theforeman-puppet', '16.5.0')
```

below the Puppet server's network line in the `Vagrantfile`.

Finding Puppet Modules for The Puppet Server (3/N)

Git might be easy to manage, as simple as a

```
package { 'git': ensure => present, }
```

but **automated** Git operations will require a bit more:

1. We want a private repository, so we'll need authentication.
2. Normally, that means `ssh`, which means managing identities and host public keys on the `git` client side, plus authorized public keys on the `git` server side.

Looking for ssh-related modules, we find we can add:

```
puppet.vm.provision "shell", \
  inline: install_dep('puppet-ssh_keygen', '5.0.2')
puppet.vm.provision "shell", \
  inline: install_dep('puppetlabs-sshkeys_core', '2.4.0')
```

Finding Puppet Modules for The Puppet Server (4/N)

How about `r10k`? Looks hopeful, as there's a Puppet Community maintained module:

```
puppet.vm.provision "shell", \
  inline: install_dep('puppet-r10k', '10.3.0')
```

Not so much with `webhook`, though. Only thing relevant has dependency conflicts with Puppet 7. So time to work out a way to install `webhook` from its GitHub tarball:

```
puppet.vm.provision "shell", \
  inline: install_dep('puppet-archive', '6.1.2')
```

Bootstrapping Puppet Server Configuration in Puppet (5/N)

Even with the Forge modules, still wound up with several slides of code. So go see it here.

Spoiler alert, we ended up needing to resolve a permissions issue between the Foreman's Puppet module and running `r10k` as an unprivileged `puppet` user. So we ended up needing

```
puppet.vm.provision "shell", \
  inline: install_dep('npwalker-recursive_file_permissions', '0.6.2')
```

in the `Vagrantfile` as well.

Bootstrapping Puppet Server Configuration in Puppet (6/6)

- At host terminal, run `vagrant up puppet`
- Puppet server will get installed in one run (OS, shell provisioner, puppet provisioner).
- At host terminal, re-run `vagrant provision puppet --provision-with puppet`
- Watch Puppet determine no further changes need to be made.
- Add and commit the changes to `Vagrantfile` and `default.pp`.

Getting puppet-control Repository

In VS Code:

- File / New Window
- View / Source Control
- Click the Clone Repository button
- URL: https://github.com/puppetlabs/control-repo.git
- Create a new folder somewhere outside the Vagrant repository folder, put the clone there.
- Open repository when prompted

In Gitea web interface:

- Create new, uninitialized repository `puppet-control` in the `theits23` organization.

Saving a Local (Gitea) Copy of puppet-control

In VS Code window with the puppet-control repository open:

- View / Source Control
- "3 dots" button above the commit message box / Remote / Remove Remote
- Select origin as the remote to remove
- "3 dots" button above the commit message box / Remote / Add Remote
- URL: git@10.234.24.2:theits23/puppet-control.git
- Name: origin
- Click the Publish Branch button

Webhook Between Gitea and Puppet

We want to automatically have changes to a Puppet code repository stored in Gitea automatically show up on the Puppet server.

Gitea can be configured to make a web request when certain events occur on any or all branches in a repository. In our case:

- Every change to every branch will trigger a web request to the webhook service on the Puppet server.
- The webhook service will be told what branch was changed.
- The webhook service will run a defined script with a command parameter including the branch name.
- The script will run the `r10k` program to check out that branch, pull down prewritten modules from Puppet Forge or Git repositories, and deploy an entire Puppet environment defining multiple servers.

General Data Flows

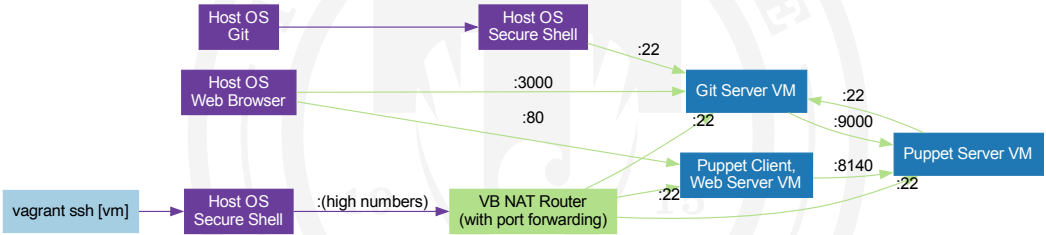

Figure 13: Data flows

/etc/webhook.yaml

```
- id: r10k
  execute-command: /usr/local/bin/r10k-deploy-ref
  command-working-directory: /tmp
  pass-arguments-to-command:
  - source: payload
    name: ref
```

/usr/local/bin/r10k-deploy-ref

```
#!/bin/bash
if [ $# -ne 1 ]; then
  echo "Usage: $0 REF"
  echo "Example: $0 refs/heads/production"
  exit 1
else
  # "How to split a string in shell and get the last field"
  # -- https://stackoverflow.com/a/9125818
  REF="$1"; BRANCH=$(echo "${REF}" | rev | cut -d/ -f1 | rev)
  r10k deploy environment "${BRANCH}" --modules --incremental
  /opt/puppetlabs/bin/puppet generate types --environment "${BRANCH}" \
    --codedir /etc/puppetlabs/code
fi
```

Gitea configuration

- Add the puppet user's public key as a deploy key for the puppet-control repository.
- Configure an outgoing webhook in the puppet-control repository, pointed at http://puppet.theits23.renf.ro:9000/hooks/r10k.

GitHub Flow for Managing Development/Testing/Bugfix Environments

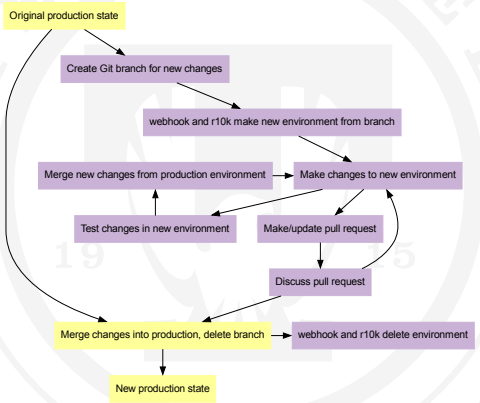

Figure 14: GitHub Flow applied to Puppet development

Roles, Profiles, and Component Modules

The roles and profiles method

- A server has one overall role
- That role can have things common with servers in other roles, including:
  - security baselines
  - who gets sudo
  - ...
- Those common things are profile classes, include all you want into the role
- Profile classes may include other profile classes, and also include component modules
- Component modules typically manage one piece of software (Apache, Samba, etc.)
- Lots of component modules for various software at Puppet Forge.

Provisioning a New Web Server in Puppet (1/10)

Modify the `Vagrantfile` to include a web server VM:

```
config.vm.define "web" do |web|
  web.vm.hostname = "web"
  web.vm.network "private_network", ip: "10.234.24.4",
    netmask: "255.255.255.0"
end
```

and the Puppet `manifests/default.pp` to include:

```
node 'web.theits23.renf.ro' {
  notify { 'web to be configured from puppet server':
    message => 'Web server to be configured from the Puppet server.',
  }
}
```

Provisioning a New Web Server in Puppet (2/10)

In VS Code window for the `iac-project` repository:

- Run `vagrant up web` and verify the web server VM is created and prints the warning message about getting configured from the Puppet server.
- Add, commit, and push the changes to `Vagrantfile` and `default.pp`.

In VS Code window for `puppet-control` repository:

- View / Source Control
- "3 dots" button above the commit message box / Branch / Create Branch From
- Use `production` branch as source, name the new branch `new_webserver`
- Click the Publish Branch button

Provisioning a New Web Server in Puppet (3/10)

In `Puppetfile`, ensure the lines

```
mod 'puppetlabs-apache', '9.1.2'
mod 'puppetlabs-concat', '7.4.0'
mod 'puppetlabs-inifile', '6.0.0'
mod 'puppetlabs-stdlib', '8.5.0'
```

exist (all these modules are from Puppet Forge). Add and commit this change with a message like `set up current module versions`.

Provisioning a New Web Server in Puppet (4/10)

In the `puppet-control` `manifests/site.pp`, replace the `node default` entry with:

```
node default {
  $role = lookup('role', Variant[String])
  case $role {
    String[1]: { include "role::${role}" }
    default: { fail('This node has no defined role.') }
  }
}
```

Save, add, and commit this change with a message like `derive node classes from Hiera` — this is a simpler version of Updating Puppet classification with hiera to use the modern lookup command.

Provisioning a New Web Server in Puppet (5/10)

Make a new file `site-modules/profile/manifests/apache.pp` in the `puppet-control` repository. Add the following lines to it:

```
# @summary Configures Apache to a site-specific standard
class profile::apache {
  class {'apache': }
  # other things can go here, like mounting
  # partitions for logs or content
}
```

Save, add, and commit this change with a message like `define apache profile`.

Provisioning a New Web Server in Puppet (6/10)

Edit the file `site-modules/role/manifests/webserver.pp` in the `puppet-control` repository. Add the following line to it below `include profile::base`:

```
include profile::apache
```

Save, add, and commit this change with a message like `update webserver role to use Apache`.

Provisioning a New Web Server in Puppet (7/10)

Use Hiera to Separate Data from Code

- Stores site-specific data in YAML, JSON, or HOCON formats
- Supports a lookup hierarchy by hostname, domain, OS, OS family, etc.
- Supports public-key encrypted data (admins encrypt values with shared public key, Puppet decrypts on the fly with private key)

Provisioning a New Web Server in Puppet (8/10)

Make a new file `data/nodes/web.theits23.renf.ro.yaml` in the `puppet-control` repository. Add the following lines to it:

```
---
role: webserver
```

Save, add, and commit this change with a message like `make 'web' a web server`. Then push all the commits to the remote Git repository with the Sync Changes button.

Provisioning a New Web Server in Puppet (9/10)

(Tip: you can use `vagrant ssh host -c "sudo -i command"` to run a privileged command on a VM and automatically log out.)

At the host terminal (and): - Edit the web server's `/etc/puppetlabs/puppet/puppet.conf` to add lines `[agent] environment=new_webserver` - Generate a certificate signing request (CSR) for the Puppet agent on the new web server with `puppet agent -t`. - Sign the CSR on the Puppet primary server with `puppetserver ca sign --certname web.theits23.renf.ro` - Apply changes to the web server through the Puppet agent with `puppet agent -t`

Provisioning a New Web Server in Puppet (10/10)

- Verify you've got a working web server by pointing the host web browser to http://10.234.24.4/

Once we're happy with the changes to the web server, we can merge them into production in Gitea:

- Make a new pull request from the `new_webserver` into `production`
- Merge the pull request and delete the `new_webserver` branch
- Edit the web VM's `/etc/puppetlabs/puppet/puppet.conf` and remove the `environment=new_webserver` line.

Minimum Standards for a Viable Infrastructure as Code (IaC) Solution

Puppet covers

1. For any given service, define a single source of authority for packages, configuration files, running services, firewall rules, etc. with customization allowed for groups of servers.
2. Automatically apply *all* needed changes, but *only* when needed.
3. Maintain balance of consistency and separation of dev/test/prod environments.

Git covers

4. Automatically maintain records of who made what change when (and ideally, why).

Vagrant (mostly) covers

5. Prefer text over binaries (automation for base OS install instead of golden thick image or VM template).
6. Enable developers to test safely and minimize exposure to outside network.

Stretch Goals for a Viable IaC Solution

Puppet covers

1. Allow multiple dev/test environments.
2. Give admins their choice of development platform (Windows, macOS, Linux).
3. Enable management of multiple server OSes (at least multiple Unix, or possibly Windows).
4. Manage endpoints as well as servers.
5. Secure and track secrets (e.g., local database passwords) in central location.
6. Be a good neighbor on already-installed systems (only manage what has to be) and expand scope from there.
7. Avoid vendor lock-in.

Things We Didn't Get To

- Cross-platform support in Puppet
- Running Puppet agent as a service
- Encrypted data in Hiera
- Options for separating Hiera data by OS, OS family, domain, etc.
- External node classifiers to centrally control which nodes get which environment
- Deeper dive into "facts" gathered by each node that inform the compiled catalog
- Distributing files and templates from Puppet
- Adding parameters to profile classes (usually populated from Hiera)
- Puppet Development Kit for building your own component modules
- More Gitea settings (branch protection, centralized authentication, . . .)

Software Credits

All presentation content edited with Visual Studio Code, tracked with Git, and hosted in GitHub.

Terminal output captured with asciinema on Unix or PowerSession on Windows.

Slides created in Pandoc's Markdown format, converted to PDF with Pandoc, TeX Live, and LaTeX Beamer. Slide theme: Cookeville, presented with Skim.

Graphs created in DOT format with Brewer paired12 color scheme, converted with Graphviz to PDF.

Questions?