Wait, what?

Getting your bearings with ServerSpec

Hardy Pottinger

Digital Library Software Developer, UCLA Library

@hardy.pottinger

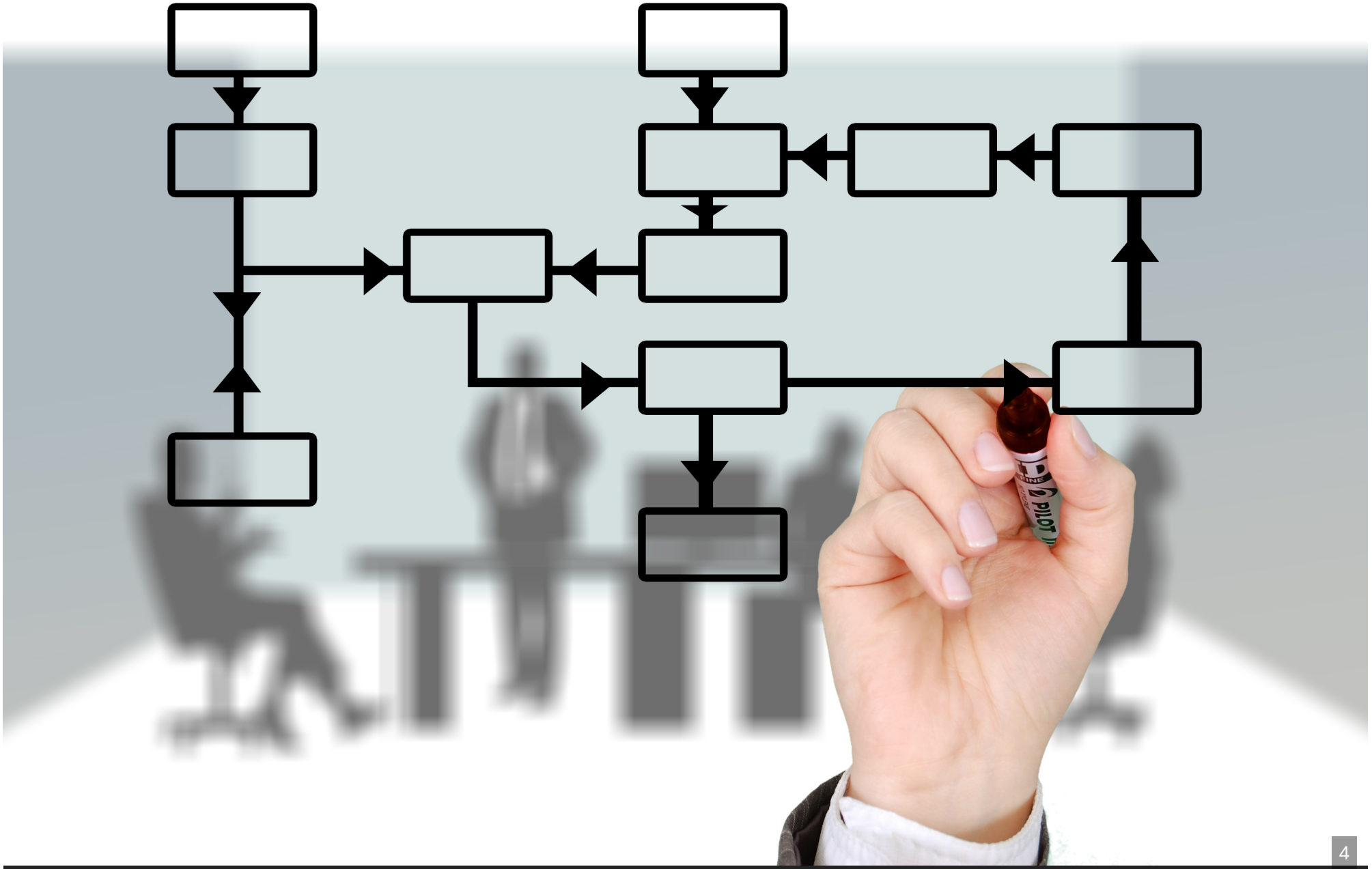
hpottinger@library.ucla.edu

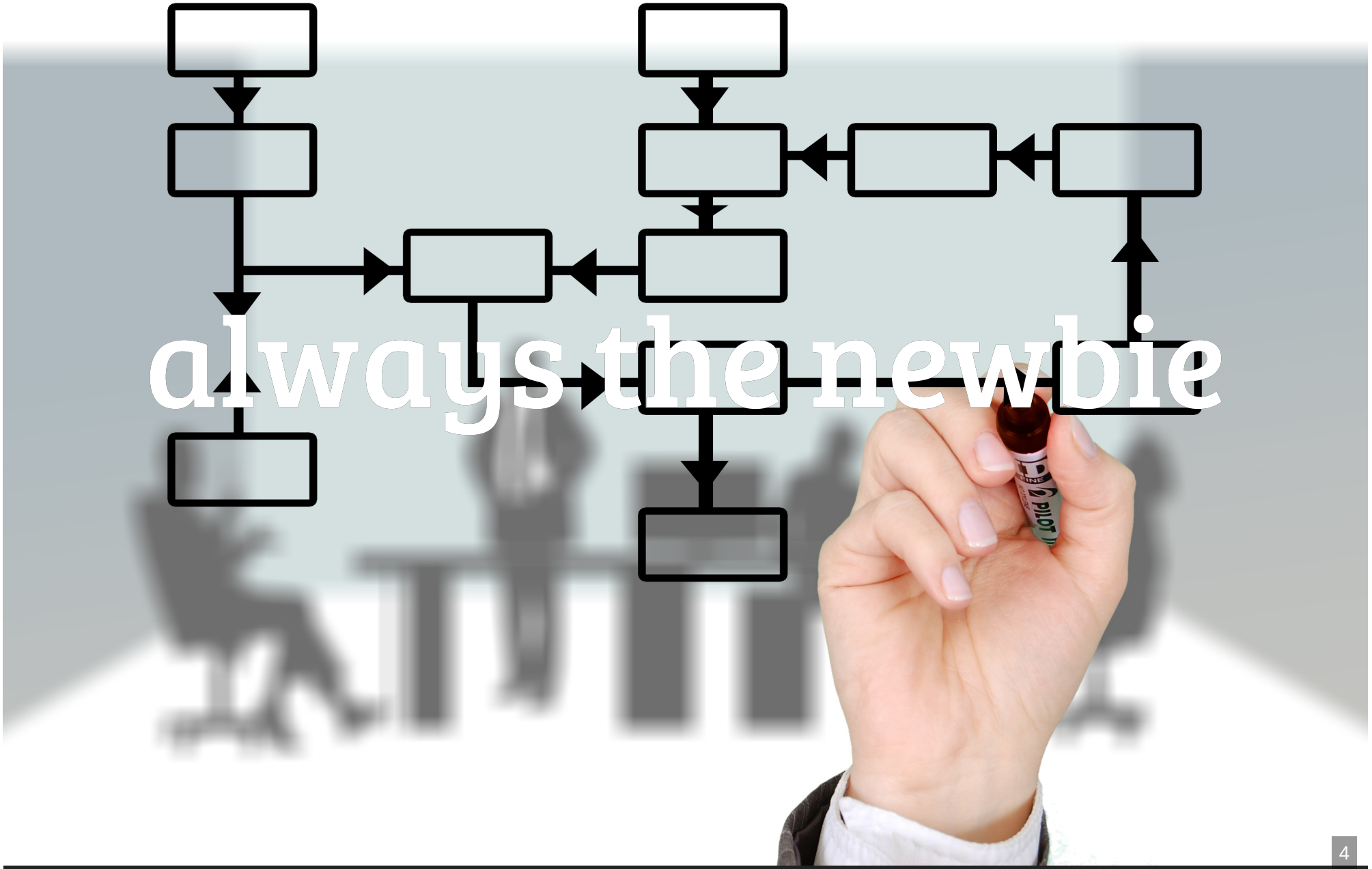
Agenda

- 30 minutes: introduction to ServerSpec
- 3 hours: 3 scenarios (with a break or two)
- 30 minutes: questions and wrap-up

We are constantly learning about our environment

- Developers shop jobs
- Have you seen the mailing list?
- We change jobs
- Our jobs change all on their own
- `We are always the newbie`




always the newbie

Why write tests?

With ServerSpec or any similar tool?

- Tests are documentation
- Your team may not survive
- Tools come and go
- No matter what happens to the tools or your team, tests will persist as documentation of your intentions and proof that the service is configured as you expected

ServerSpec

- Extension of RSpec
- Is a Ruby gem

Software Reuse

spec spec spec...

- RSpec
- ServerSpec
- DockerSpec
- SpecInfra ¯\(ツ)/¯

Installing ServerSpec

- It's a Ruby gem, you'll need Ruby 2.0.x+ installed
  `sudo gem install serverspec`
- Rake, too: `sudo gem install rake`
- SSH access to the servers you are testing
- Sudo not required, but it makes life easier

```
$ serverspec-init
Select OS type:
  1) UN*X
  2) Windows
Select number: 1
Select a backend type:
  1) SSH
  2) Exec (local)
Select number: 1
Vagrant instance y/n: n
Input target host name: waitwat
 + spec/
 + spec/waitwat/
 + spec/waitwat/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

```
$ serverspec-init
Select OS type:
  1) UN*X
  2) Windows
Select number: 1
Select a backend type:
  1) SSH
  2) Exec (local)
Select number: 1
Vagrant instance y/n: n
Input target host name: waitwat
 + spec/
 + spec/waitwat/
 + spec/waitwat/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

serverspec-init

```
$ serverspec-init
Select OS type:
  1) UN*X
  2) Windows
Select number: 1
Select a backend type:
  1) SSH
  2) Exec (local)
Select number: 1
Vagrant instance y/n: n
Input target host name: waitwat
 + spec/
 + spec/waitwat/
 + spec/waitwat/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

serverspec-init , answers

```
$ serverspec-init
Select OS type:
  1) UN*X
  2) Windows
Select number: 1
Select a backend type:
  1) SSH
  2) Exec (local)
Select number: 1
Vagrant instance y/n: n
Input target host name: waitwat
 + spec/
 + spec/waitwat/
 + spec/waitwat/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

serverspec-init , answers , boom

```
$ serverspec-init
Select OS type:
  1) UN*X
  2) Windows
Select number: 1
Select a backend type:
  1) SSH
  2) Exec (local)
Select number: 1
Vagrant instance y/n: n
Input target host name: waitwat
 + spec/
 + spec/waitwat/
 + spec/waitwat/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

serverspec-init , answers , boom , sample_spec.rb

```
require 'spec_helper'
describe package('httpd') do
  it { should be_installed }
end

describe service('httpd') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end

describe file(/etc/httpd/conf/httpd.conf) do
  it {should be_file}
  it {should contain "ServerName my-server-name"}
end
```

```
require 'spec_helper'
describe package('httpd') do
  it { should be_installed }
end

describe service('httpd') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end

describe file(/etc/httpd/conf/httpd.conf) do
  it {should be_file}
  it {should contain "ServerName my-server-name"}
end
```

spec_helper

```
require 'spec_helper'
describe package('httpd') do
  it { should be_installed }
end

describe service('httpd') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end

describe file(/etc/httpd/conf/httpd.conf) do
  it {should be_file}
  it {should contain "ServerName my-server-name"}
end
```

spec_helper , package resource

```
require 'spec_helper'
describe package('httpd') do
  it { should be_installed }
end

describe service('httpd') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end

describe file(/etc/httpd/conf/httpd.conf) do
  it {should be_file}
  it {should contain "ServerName my-server-name"}
end
```

spec_helper , package resource , service resource

```
require 'spec_helper'
describe package('httpd') do
  it { should be_installed }
end

describe service('httpd') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end

describe file(/etc/httpd/conf/httpd.conf) do
  it {should be_file}
  it {should contain "ServerName my-server-name"}
end
```

spec_helper , package resource , service resource , port resource

```
require 'spec_helper'
describe package('httpd') do
  it { should be_installed }
end

describe service('httpd') do
  it { should be_enabled }
  it { should be_running }
end

describe port(80) do
  it { should be_listening }
end

describe file(/etc/httpd/conf/httpd.conf) do
  it {should be_file}
  it {should contain "ServerName my-server-name"}
end
```

spec_helper , package resource , service resource , port resource , file resource

Resources

https://serverspec.org/resource_types.html

- packages
- services
- ports
- files
- commands
- users and groups
- cron

Run the tests

- RSpec tests usually go in a folder called `spec`
- calling ServerSpec is exactly the same as any other RSpec test

```
rspec spec
```

- this will run all the tests in the spec folder
- or `spec/ask/for/whichever/spec.rb` you want
- you can also use a Rakefile to automate more complex tests

Demo: 1 ServerSpec-Samvera

- code: tinyurl.com/uclalibrary-serverspec-samvera
- demo

Scaling up to more than one server

https://tinyurl.com/uclalibrary-serverspec-samvera

Spec_helper

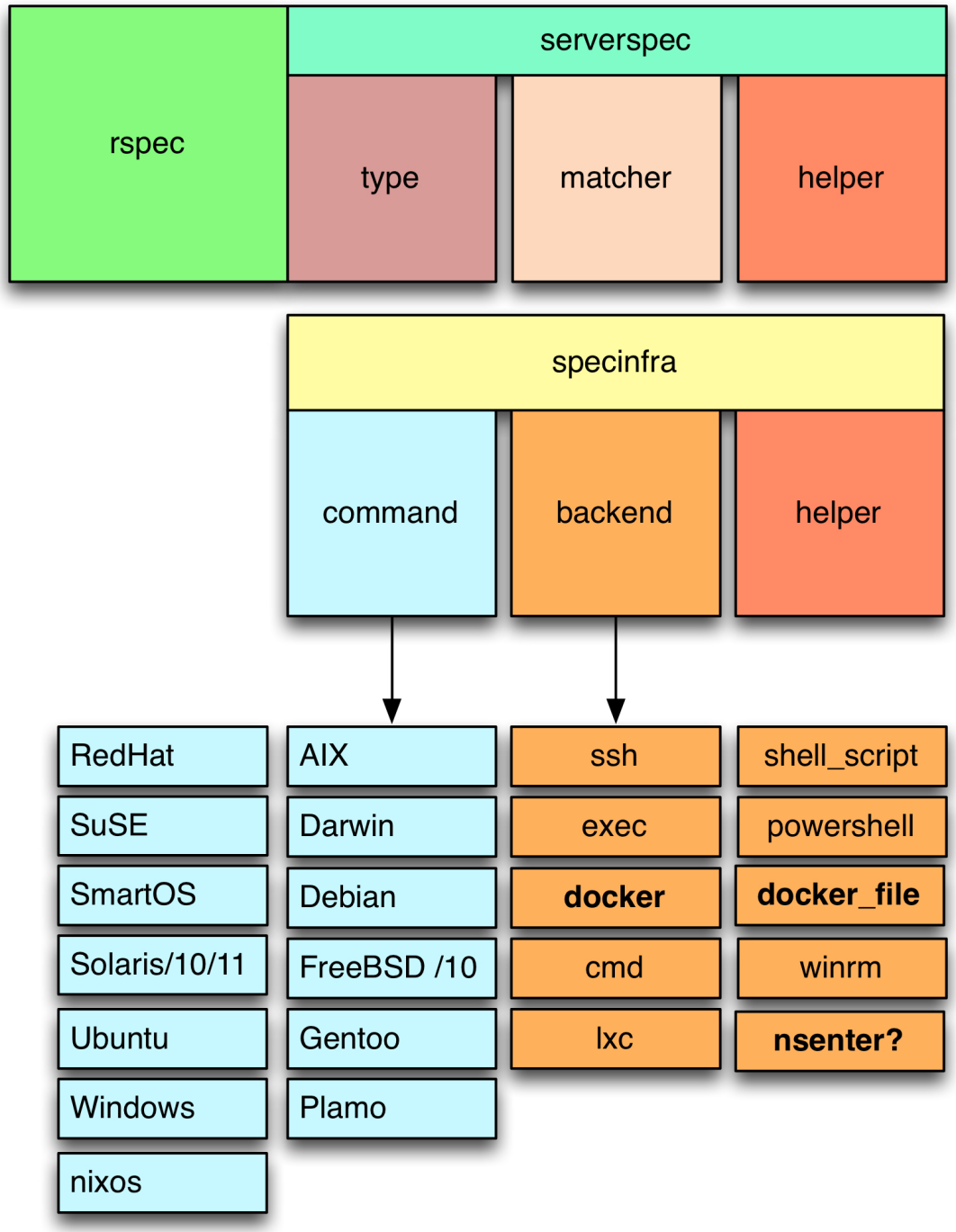
```
require 'serverspec'
require 'pathname'
require 'net/ssh'
require 'yaml'

base_spec_dir = Pathname.new(File.join(File.dirname(__FILE__)))

Dir[base_spec_dir.join('shared/**/*.rb')].sort.each{ |f| require f }

set :backend, :ssh
set :disable_sudo, true

set :path, '/usr/sbin:$PATH'

properties = YAML.load_file(base_spec_dir.join('properties.yml'))

options = Net::SSH::Config.for(host)
options[:user] = 'deploy'
host = ENV['TARGET_HOST']
```

Use a Rakefile

- automate the running of complex tests
- Rake is a software task management and build automation tool
- Similar to Make
- Written in Ruby

Sharing code

https://serverspec.org/advanced_tips.html
https://tinyurl.com/uclalibrary-serverspec-samvera


rspec
serverspec
type
matcher
helper
specinfra
command
backend
helper
RedHat
SuSE
SmartOS
Solaris/10/11
Ubuntu
Windows
nixos
AIX
Darwin
Debian
FreeBSD /10
Gentoo
Plamo
ssh
exec
docker
cmd
lxc
shell_script
powershell
docker_file
winrm
nsenter?

Gotchas

- you'll need to be sure the `ss` command is available on the test target
  - this is installed by default on RHEL
  - for Ubuntu, you'll need to install the `iproute2` package
- you'll need to be sure `/usr/sbin` is in the path, if your test target is RHEL
  - you can set the `:path` in spec_helper

Containers? Docker?

- many options are available, worth researching
- DockerSpec: https://github.com/zuazo/dockerspec

Demo: 2 Docker-Cantaloupe

- code: github.com/UCLALibrary/docker-cantaloupe
- demo

Other options

- InSpec (still Ruby, by the Chef people)
- Goss (YAML, can generate tests from current system state)
- TestInfra (Python, works well with Ansible)
- Molecule (only tests Ansible roles)
- ServerSpec-runner (organizes complex ServerSpec tests)

Why ServerSpec?

https://medium.com/@Joachim8675309/serverspec-vs-inspec-17272df2718f

- ServerSpec is targeted for the developer working on DevOps
- integrates with Vagrant and Docker
- gains a lot of fexibility from leveraging RSpec
- works well with SSH, so moving from local tests to infrastructure tests is straightforward

Three Scenarios...

getting ready (1 of 3)

- you do not have to follow along on your computer, you can just watch me
- slides are at: github.com/hardyoyo/ucdlfx19-serverspec-workshop
- if you do want to follow along, you need Ruby 2.5.1 or higher installed

Three Scenarios...

getting ready (2 of 3)

- you can check your Ruby version with `ruby -v`
- is it 2.5.1 or higher?
  - Yes, you're good
  - No, just watch me.
- to install ServerSpec, run this:

```
gem install serverspec
```

Three Scenarios...

getting ready (3 of 3)

- you also need to be able to SSH to the fakeuniversity servers

```
ssh www1.fakeuniversity.space
```

- your *username* and *password* are on the sheet of paper you picked up at the beginning

Three Scenarios...

disclaimer

The story, all names, characters, and incidents portrayed in this workshop are fictitious. No identification with actual persons (living or deceased), places, buildings, and products is intended or should be inferred.

Three Scenarios...

the premis

- you are the newbie, a new hire, at `fakeuniversity.space` library
- you are trying to make sense of it all
- you want to be useful at the same time
- this place is a mess

Scenario One

Most of our stuff is static?

- write a test to confirm Apache is running and it's the correct version
- deal with any surprises that come up

Scenario One, Static

what are we testing?

server names

- www1.fakeuniversity.space
- www2.fakeuniversity.space
- bubbles.fakeuniversity.space
  - OS: Ubuntu
  - port: 80
  - package: apache2
  - version: 2.4.7-1ubuntu4.1

Scenario One, Static

serverspec-init to the rescue! (1 of 2)

```
mkdir waitwhat
cd waitwhat
serverspec-init
Select OS type:

  1) UN*X
  2) Windows

Select number: 1

Select a backend type:

  1) SSH
  2) Exec (local)

Select number: 1
cd spec
```

Scenario One, Static

serverspec-init to the rescue! (2 of 2)

```
Vagrant instance y/n: n
Input target host name: www1.fakeuniversity.space
 + spec/
 + spec/www1.fakeuniversity.space/
 + spec/www1.fakeuniversity.space/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

Scenario One, Static

spec_helper.rb

- the default spec_helper.rb file might need tweaking
- it doesn't allow password authentication, which we need
- you can just copy the one I tested

```
cd spec
wget https://tinyurl.com/spec-helper-rb
```

Scenario One, Static

spec_helper.rb

```
# retrieve the hostname
host = ENV['TARGET_HOST']
puts "host =  '#{host}'"

# configure options for SSH
options = Net::SSH::Config.for(host)
set :host,        options[:host_name] || host
options[:auth_methods]  = ['password']

if ENV['LOGIN_USER']
    options[:user] = ENV['LOGIN_USER']
else
    options[:user] ||= Etc.getlogin
end

# and apply our SSH options
set :ssh_options, options

# Disable sudo
set :disable_sudo, true
```

Scenario One, Static

what are we testing?

details

- www1.fakeuniversity.space
- www2.fakeuniversity.space
- bubbles.fakeuniversity.space
- OS: Ubuntu
- port: 80
- package: apache2
- version: 2.4.7-1ubuntu4.1

Scenario One, Static

It's never this easy

- https://serverspec.org/advanced_tips.html

Scenario Two

We really want to move everything to Hyrax

- "We have a pilot server..."
- "Can you write a test for the pilot server?"

Scenario Two: Hyrax?

what are we testing?

details

hyraxdemo.fakeuniversity.space ip address: 34.222.253.179

- OS: Ubuntu
- services: PostgreSQL, Solr, Fcrepo, Apache
- ports: 3306, 8081, 80, 443, 8983
- packages: postgresql-10, postgresql-client-10, apache, tomcat

Scenario Two: Hyrax?

services

- a **database**: PostgreSQL
- an **index**: Solr
- an **object store**: Fcrepo
- an **application server**: Apache
- https://serverspec.org/advanced_tips.html

Scenario Two: Hyrax?

services (1 of 2)

- a **database**: PostgreSQL
  - port: 5432 | service: postgres
  - packages: postgresql-10, postgresql-client-10
  - versions: 10.7-0ubuntu0.18.04.1
- an **index**: Solr
  - port: 8983 | service: solr
  - packages: use the binary installer from Solr
  - version: 6.6.2

Scenario Two: Hyrax?

services (2 of 2)

- an **object store**: Fcrepo
  - port: 8081 | service: fedora
  - packages: tomcat and the fcrepo war file
  - version:
- an **application server**: Apache Passenger
  - port:80,443 | service: apache2
  - packages: apache2
  - versions:
- Java, Ruby, Gems, Capistrano, oh my!

Scenario Three

Gosh, everything is slow...
let's throw hardware at it

- refactor the test for the pilot server into pieces that can be reused
- write a test for each environment (dev, staging, prod) using these pieces

Scenario Three

MOAR Samvera (hardware)?

server names

- hyrax-db.fakeuniversity.space
- hyrax-solr.fakeuniversity.space
- hyrax-fcrepo.fakeuniversity.space
- hyrax-app.fakeuniversity.space

just re-use the shared behaviors we already have

More resource types:

https://serverspec.org/resource_types.html

Things to remember

- spec_helper hides a *lot* of functionality
  - if you can't figure out how to do something, the solution probably is in there
  - https://git.io/fjcAw <-helpful notes
- Spec tests are Ruby files, you have the full power of Ruby at your command

Questions

and Wrap-up

- Is this just for "getting our bearings?"
- How do we integrate this kind of testing into existing development workflows?
- Try DockerSpec if you're using Docker
  - https://github.com/zuazo/dockerspec
  - https://github.com/UCLALibrary/docker-cantaloupe/tree/master/spec

Have you seen these cool zines?

by Julia Evans

https://wizardzines.com/ or https://jvns.ca/

- better than a cheat sheet
- printable
- leave these all over your workplace

Thanks

- Inspiration: https://git.io/fjCID
- *ServerSpec Components*, adapted from "Introduction to Test-Driven Docker Development," by Peter Roßbach, Wednesday, August 12, 2015, Entwickler.de
- *It's Complicated just like life* from Wikimedia Commons
- DCE's Ansible-Samvera
- UCLA Library

Slides: github.com/hardyoyo/ucdlfx19-serverspec-workshop