Serverspec in cloud provision

Serguei Kouzmine
kouzmine_serguei@yahoo.com

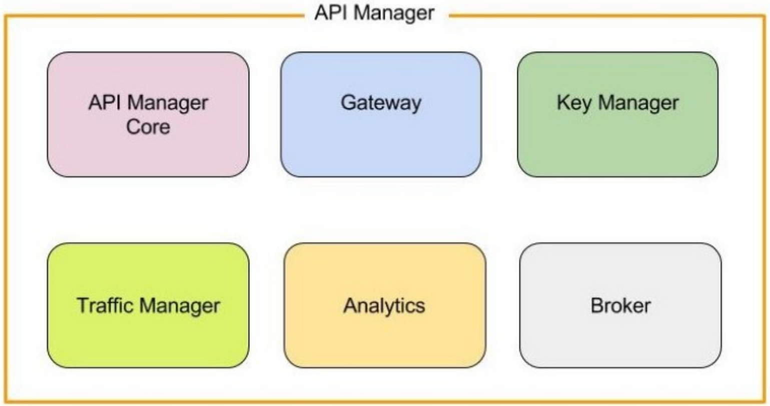
Serverspec in cloud provision

Successful operation automation usually needs, along with core tool technical skills some curiosity and knowledge about the system under test - not strictly but is recommended. For example, (no)SQL, Angular/MQ experience would be a great help in web testing, HTML / Javascript no longer sufficient .

Often more critical than provision engine (Puppet, Chef, Ansible, DSC learning curve.

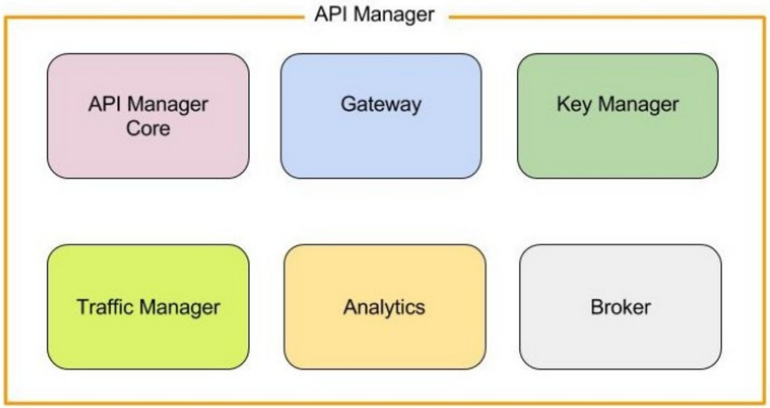
In a similar case, an QA engineer fluent in Selenium Appium or Katalon often is or eager to grow Web developer skills but unlikely ever interested in the code base of those tools.

IaaS to PaaS mutation challenge



Cloud architect envisions building a platform for some integrated enterprise modern application stack

IaaS to PaaS mutation challenge



API Manager

API Manager Core

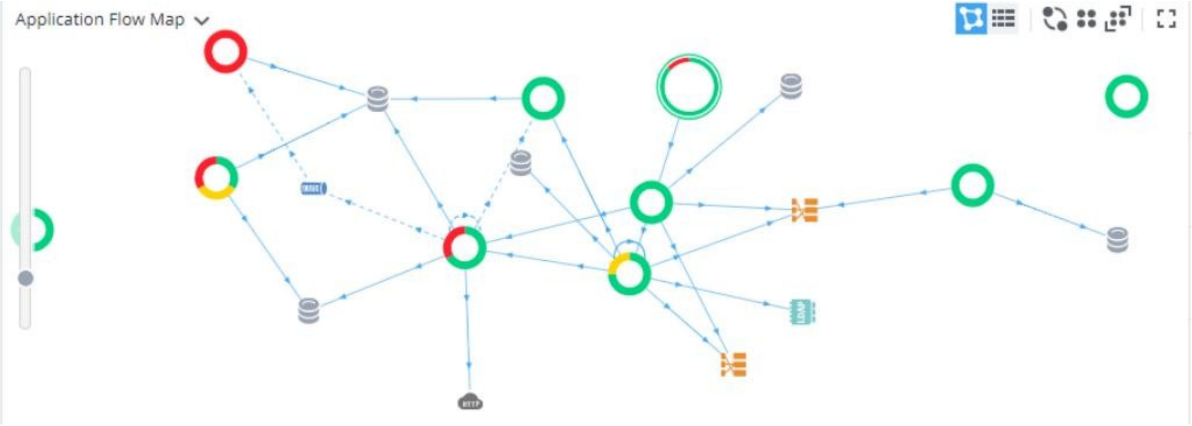
Gateway

Key Manager

Traffic Manager

Analytics

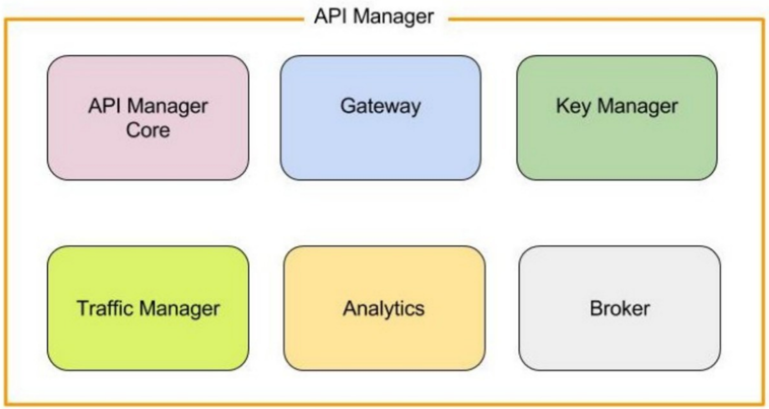
Broker

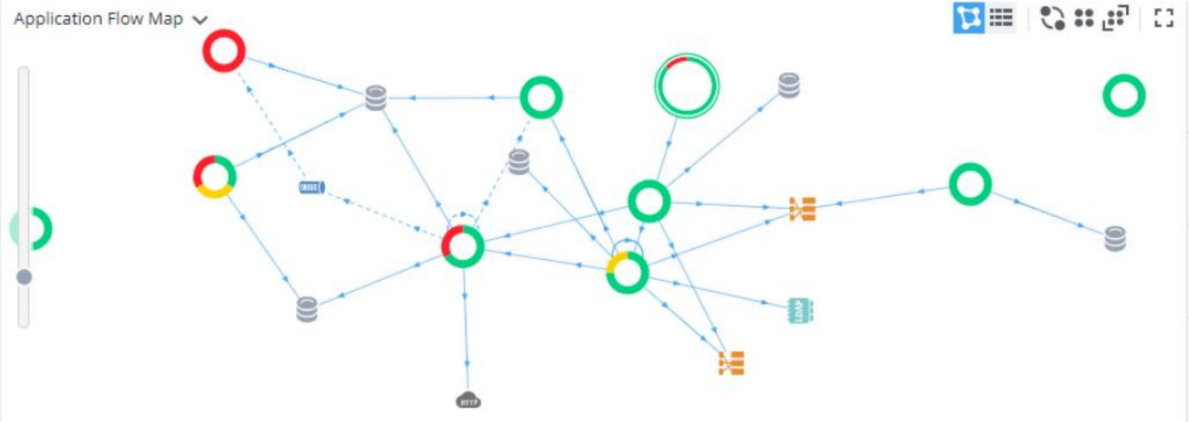


Cloud cluster will be provisioned by Puppet

https://wso2.com/

IaaS to PaaS mutation challenge



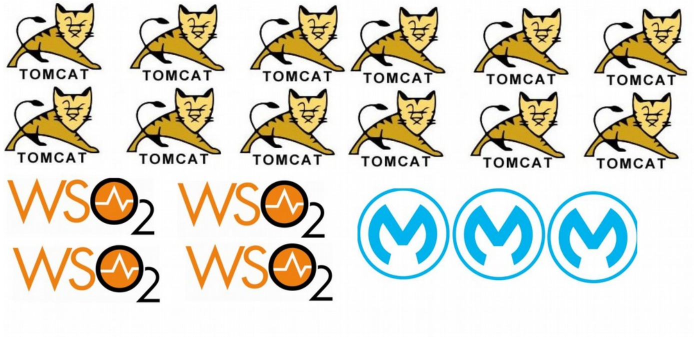


- WSO2 API Manager
- WSO2 API Manager Analytics
- WSO2 App Manager
- WSO2 Application Server
- WSO2 Business Process Server
- WSO2 Business Rules Server
- WSO2 Carbon
- WSO2 Complex Event Processor
- WSO2 Dashboard Server
- WSO2 Data Analytics Server
- WSO2 Data Services Server
- WSO2 Elastic Load Balancer
- WSO2 Enterprise Integrator
- WSO2 Enterprise Mobility Manager
- WSO2 Enterprise Service Bus
- WSO2 Enterprise Service Bus Analytics
- WSO2 Enterprise Store
- WSO2 Governance Registry
- WSO2 Identity Server
- WSO2 Identity Server Analytics
- WSO2 Identity Server Key Manager
- WSO2 IoT Server
- WSO2 Machine Learner
- WSO2 Message Broker
- WSO2 Private PaaS
- WSO2 Storage Server

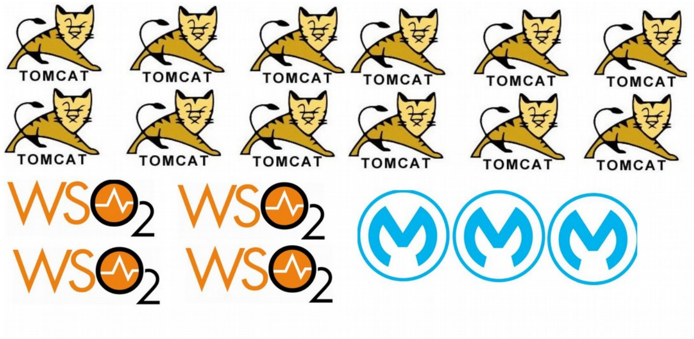
https://wso2.com/

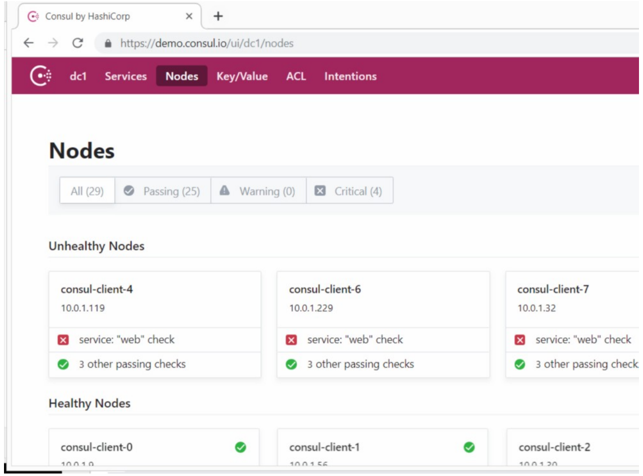
IaaS to PaaS mutation challenge

IaaS to PaaS mutation challenge



IaaS to PaaS mutation challenge





Nodes

All (29) Passing (25) Warning (0) Critical (4)

Unhealthy Nodes

consul-client-4
10.0.1.119
☒ service: "web" check
3 other passing checks

consul-client-6
10.0.1.229
☒ service: "web" check
3 other passing checks

consul-client-7
10.0.1.32
☒ service: "web" check
3 other passing check

Healthy Nodes

consul-client-0 ✓

consul-client-1 ✓

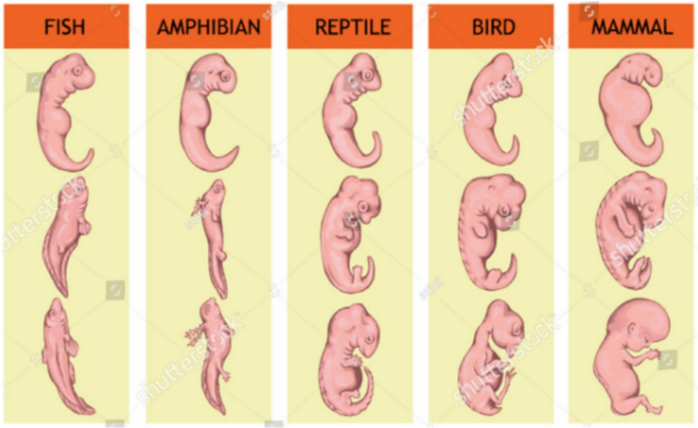
consul-client-2

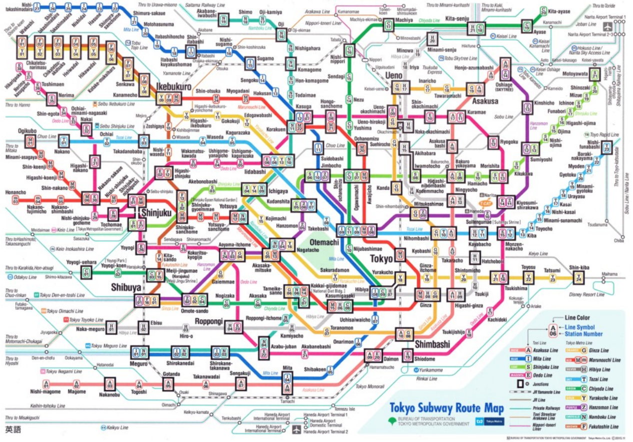
`org.springframework.cloud.client.discovery`

public interface `DiscoveryClient`

`DiscoveryClient` represents read operations commonly available to Discovery service such as Netflix Eureka or consul.io

IaaS to PaaS mutation challenge





- Control repository and r10k effectively manage environments through git braches
- Hiera separate parameters from the manifests with easy parameter override
- Puppet operates system information and module specific facts to compile and apply the catalog

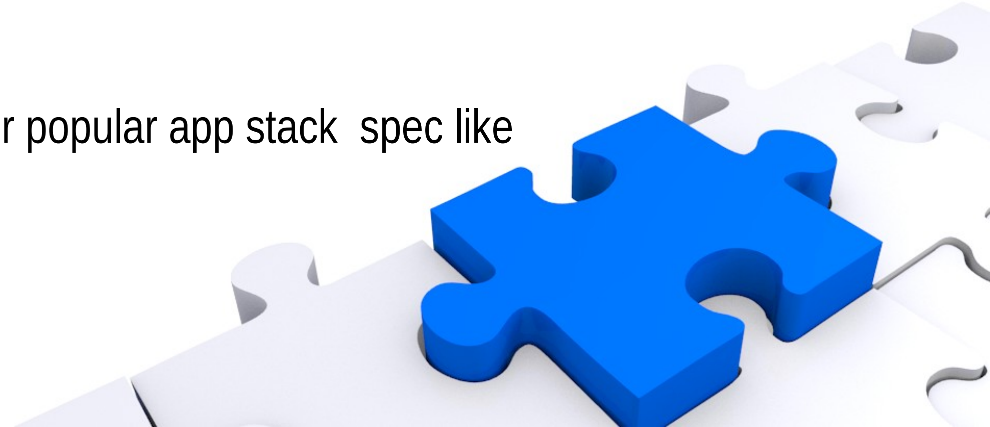
Serverspec comes to rescue

The home page https://serverspec.org/ describes core resource types:
bond | bridge | cgroup | command | cron | default_gateway | docker_container | docker_image | file | group | host | iis_app_pool | iis_website | interface | ip6tables | ipfilter | ipnat | iptables | kernel_module | linux_audit_system | linux_kernel_parameter | lxc | mail_alias | mysql_config | package | php_config | port | ppa | process | routing_table | selinux | selinux_module | service | user | x509_certificate | x509_private_key | windows_feature | windows_registry_key | yumrepo | zfs

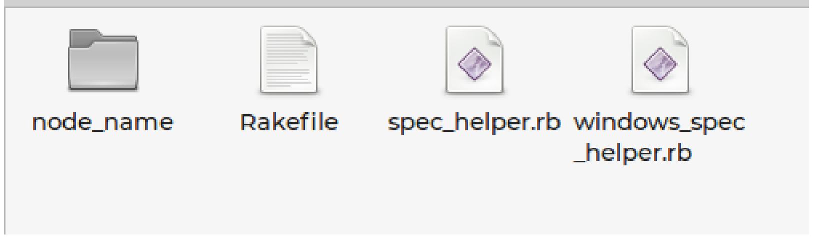
Code hosted on github in mizzy/serverspec, mizzy/specinfra ,vvchik/vagrant-serverspec, covers 20+ operating systems

A very similar inspec/inspec framework exists for Chef.

A handful of active projects present extended types created for popular app stack spec like npm, ELK etc.



Intro to Serverspec



- Directory with spec file(s) named as target node for multi node provisioning
- Helper file `spec_helper.rb` with OS-specific configuration ssh, sudo, tempfile and sets the `:backend` to either `:cmd, :exec, :ssh or :winrm.` Usually helper files for Windows and unix are stored separately
- `Rakefile` where path to every spec file passed to constructor of `RSpec::Core::RakeTask`
- An `rspec <filename>` command is also possible (with the same effect) and often used with Docker

Intro to Serverspec

The most basic spec are just a copy of the http://serverspec.org with the object name changed

```
context 'Basic' do
  service_name = ‘jenkins’
  describe service(service_name) do
    it { should be_enabled   }
    it { should be_running   }
  end
end
```

Where are practical expectations

The stock example would fit to validate the Jenkins module from Puppet Forge, but in practical scenario one's expectations are more specific

```
context 'Jenkins Security' do
  xmlfile = '/var/lib/jenkins/config.xml'
  describe file(xmlfile) do
    it { should exist }
    its(:content) { should match '<useSecurity>false</useSecurity>'}
  end
end
```

Exploring Serverspec

Having noticed that the 'file' method is backend run of a `cat` ‘run_command’ , and intending to processes XML in Ruby one would compose the test which does just that. Gets complex quickly

```
require 'rexml/document'
  describe file(xmlfile) do
    begin
      content = Specinfra.backend.run_command("cat '#{xmlfile}'").stdout
      begin
        doc = Document.new(content); // ready for some DOM processing
        result = true
      rescue ParseException => e
        result = false
      end
    rescue => e
      result = false
    end
    it { result.should be_truthy }
  end
```

https://github.com/mizzy/serverspec/blob/master/lib/serverspec/type/hadoop_config.rb

Exploring Serverspec

Running vendor command is the ultimate way to query app configuration, especially when provision is broken and focus on verifying too close to what Puppet is modifying could yield a false positive

```
context 'Mysql Datadir' do
  custom_datadir = '/opt/mysql/var/lib/mysql/'
  describe command(<<-EOF
    mysql -sBEe 'select @@datadir;'
  EOF
  ) do
    its(:exit_status) {should eq 0 }
    # implicitly confirm the mysql is running
    its(:stdout) { should match custom_datadir }
  end
end
```

Raise of custom Serverspec

One is basically interested in

- application configuration files (JSON, XML)
- systemd service details, various response headers to web requests,
- open TCP ports
- download artifact hashes
- Jenkins fine level details of Jenkins job and pipeline configs
- GAC and assemblies
- Scheduled Tasks (Windows) and cron jobs (unix)
- Puppet last run reports

As a result one has quickly growing collection. E.g. repo `https://github.com/sergueik/serverspec_custom_types` contains 100 or more snippets for each Linux and Windows.

https://github.com/mizzy/serverspec/blob/master/lib/serverspec/type/hadoop_config.rb

Serverspec implementation

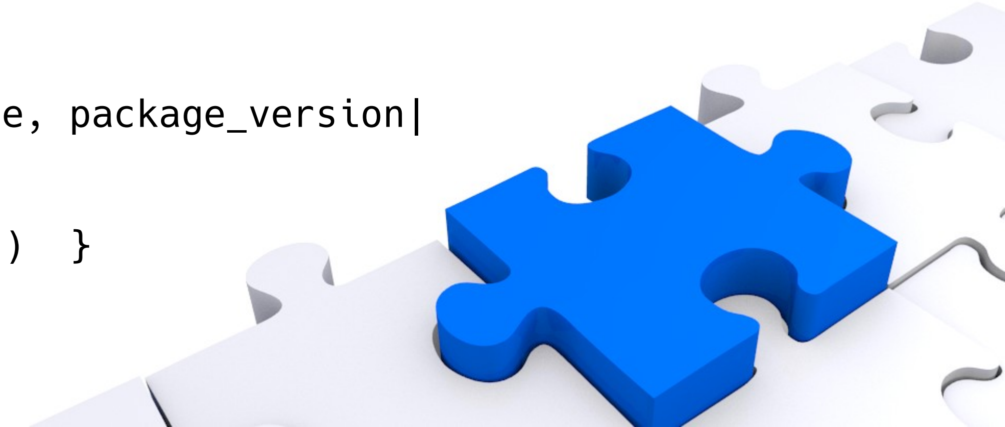
Extreme serverspec

The engine responsible for the serverspec execution resembles that of Puppet or Chef: source is sent from developer to the target node to be eventually converted into target OS specific low level commands to execute - result is sent back to the developer. Unlike provision the serverspec is executed for its direct, not side effects. Both serverspec and Puppet has significant amount of code wrapping the actual command in some custom DSL, however a plain Exec/Command class is still available.

In the extreme case in the body of a Ruby spec Command, one could find a full source code of a java class that would be compiled and run in the target node to load and examine some cryptic JDBC, or ELK configuration changes applied in the course of node provision:

There isn’t any ‘spy’ facilities for server spec or unit test developments, neither are any in pure Ruby, Java or .Net, nor there is any ‘recording’ environment. To help new developers learn and quickly adopt to server spec follow clear *Rspec/Cucumber* semantics:

```
describe service(‘tomcat’) do
  it {should be_running }
end

describe port(8443) do
  it { should be_listening.with(‘tcp’) }
end

{'linux-kstat' => '0.1.3' }.each do |package_name, package_version|
   describe package(package_name) do
     it { should be_installed.by('gem')
                   .with_version(package_version)  }
    end
  end
```



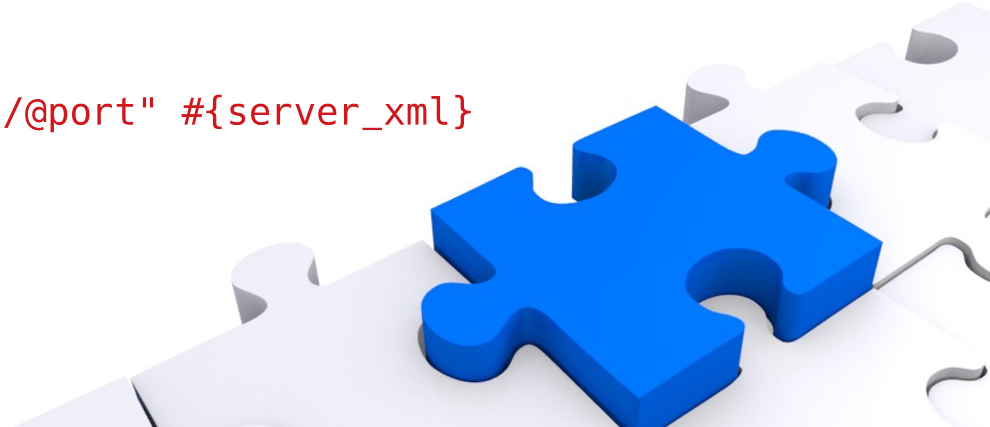
Soon with the growing number of detail the only two qualifying resources are the file and the command

```
context 'Virtual Host settings' do
  describe file('/etc/httpd/conf.d/vhost.conf') do
    [ 'ProxyRequests Off',
       # multiple settings
    ].each do |line|
      its(:content) { should match "^\s*" + Regexp.escape(line) }
    end
  end
end
```

Eventually the *command* is where the tricky part is:

```
 Context ‘Security headers’ do
   describe command('curl -k -I http://localhost') do
     its(:stdout) { should match /Server: Apache\/\d\.\d+\.\d+ (:?Unix|CentOS)/i }
   end
 end

context 'Tomcat shutdown port' do
  server_xml = "#{catalina_home}/conf/server.xml"
  describe command(<<-EOF
   xmllint --xpath "/Server[@shutdown='SHUTDOWN']/@port" #{server_xml}
  EOF
  ) do
    its(:exit_status) { should eq 0 }
    its(:stdout) { should match 'port="-1"' }
  end
end
```

https://tomcat.apache.org/tomcat-8.5-doc/appdev/web.xml.txt



Gradually the *command* itself could become tricky but reusable (Ruby or libxml2 used to focus on specific XML node):

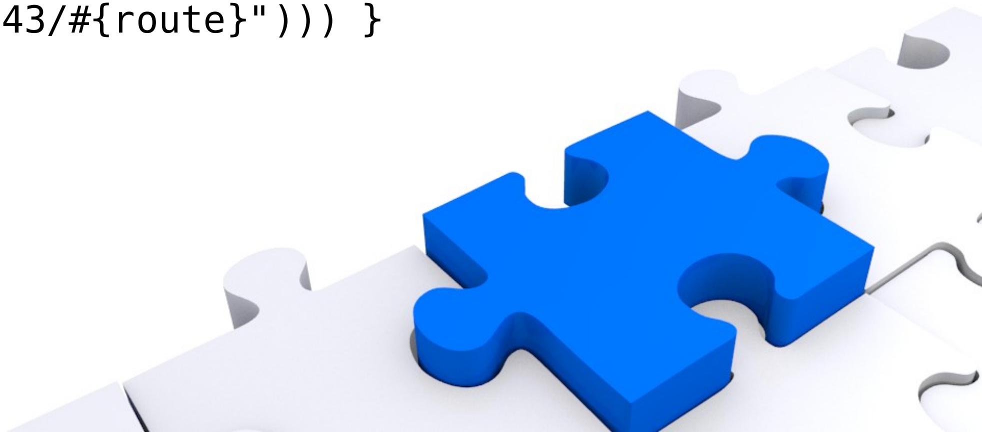
```
describe 'redirect port 8080' do
  doc = Document.new(content)
  result = REXML::XPath.first(doc, "/Server/Service/Connector[@port = \"8080\"]
/@redirectPort").value
  it { result.should match '8443' }
end

context 'Tomcat servlet configuration' do
  class_name = 'com.mycompany.mypackage.ControllerServlet'
  describe command(<<-EOF
   xmllint --xpath "//*[local-name()='servlet']/*[local-name()='servlet-class']/text()"
#{web_xml}
  EOF
  ) do
    its(:stdout) { should match Regexp.new(class_name) }
 end
```

https://tomcat.apache.org/tomcat-8.5-doc/appdev/web.xml.txt

Eventually the *command* is where the tricky part is (jq used to focus into the specific node of the JSON configuration):

```
context 'Consul service health check configuration' do
 {
   'myservice' => 'api/health'
 }.each  do |service, route|
    describe command("jq '.service.checks[].http' < '/etc/consul.d/#{service}.json'") do
        let(:path) { '/bin:/usr/bin:/usr/local/bin'}
        its(:stdout) { should match( Regexp.new(
              Regexp.escape("https://127.0.0.1:8443/#{route}"))) }
    end
  end
end
```

https://tomcat.apache.org/tomcat-8.5-doc/appdev/web.xml.txt



On Windows, Powershell and C# is used to retrieve obscure information about .Net

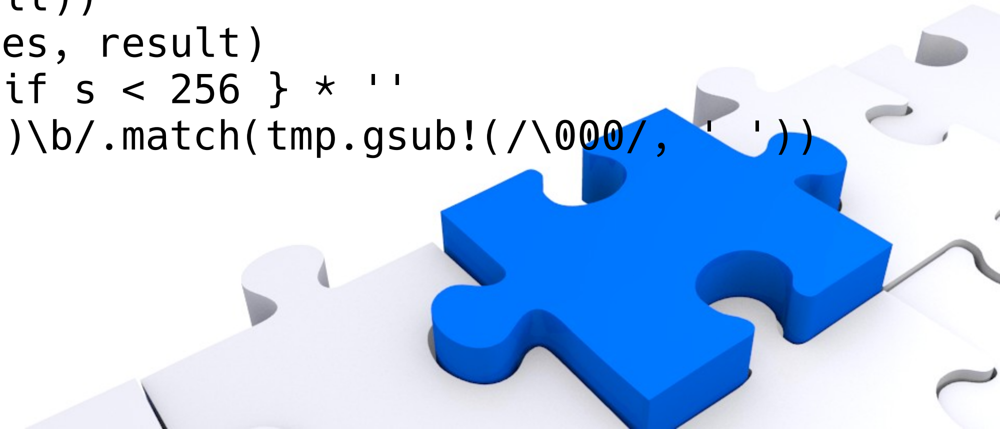
```
context 'specific assembly in GAC' do
  assembly_name = 'WindowsFormsIntegration'
  token = '31bf3856ad364e35'
  describe command(<<-EOF
  $result = @()
    [Object].Assembly.GetType('Microsoft.Win32.Fusion').GetMethod('ReadCache')
    .Invoke($null, @([Collections.ArrayList]$result, '#{assembly_name}', [UInt32]2 ))
  $result
  EOF
  )
  do
    its(:stdout) { should contain
     "#{assembly_name}, Version=3.0.0.0, Culture=neutral, PublicKeyToken=#{token}" }
  end
end
```

Any passing *serverspec expectation* may easily become a Puppet *fact*:

```
if Facter.value(:kernel) == 'windows'
  Facter.add('version') do
    begin
      script = "([xml](get-content -path 'version.xml')).Info.Product.version"
      command = "powershell.exe -executionpolicy remotesigned -command \" &
{ #{script} }\""
      result = Facter::Util::Resolution.exec(command)
    rescue => ex
      $stderr.puts ex.to_s
    end
  end
end
```

A proper *Serverspec expectation* could replace native Ruby Puppet module *fact* which can be quite cryptic:

```
Facter.add('version') do
  extend FFI::Library
  ffi_lib 'version.dll'
  attach_function :resource_size,:GetFileVersionInfoSizeA [:ptr, :ptr ], :int
  attach_function :version, :GetFileVersionInfoA, [ :ptr,:int,:int,:buf_out], :int
  version_information = '\VarFileInfo\Translation'.encode('UTF-16LE')
  result = ' ' * (resource_size(filepath, nil))
  status = version(filename, 0, size_in_bytes, result)
  tmp = result.unpack('v*').map{ |s| s.chr if s < 256 } * ''
  version_match = /FileVersion\s+\b([0-9.]+)\b/.match(tmp.gsub!(/\000/, ' '))
  version_match[1].to_s
end
```



Serverspec expectation could be used for consul *“script-kind” service check* similar to `/etc/consul.d/mongodb.json`:

```
{
  "service": {
    "name": "mongo-db",
    "tags": ["mongo"],
    "address": "192.168.31.02",
    "port": 27017,
    "checks": [
      {
        "name": "Checking MongoDB"
        "script": "/usr/bin/check_mongo.py --host 192.168.31.02 --port 27017",
        "interval": "5s"
      }

    ]
}
```

https://www.consul.io/docs/agent/checks.html

Making Java Application consul-ready

The *http-kind service check* heartbeat-response is provided by Spring after annotation is provided and dependency added to `pom.xml`:

```
@SpringBootApplication
@EnableDiscoveryClient
public class DiscoverableApplication {
  public static void main(String[] args) {
    SpringApplication.run(DiscoverableApplication.class, args);
  }
}
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Serverspec Integration challenges

https://tomcat.apache.org/tomcat-8.5-doc/appdev/web.xml.txt

Vagrant Serverspec provisoner

serverspec vagrant provisioner is part of the Vagrant flow, a little bit of disadvantage so its `rake spec` is from a deep stack of Ruby calls

elementary tasks like `$DEBUG = ENV.fetch('DEBUG', false)` become a bit problematic

serverspec is scheduled afterprovision and rerun is time consuming - not really when module is idempotent

with default settings error stack is super extra verbose

spec file (`node_spec.rb`) is not visible to therefore can not be produced by Puppet module - solvable through relative reference placing under `files/serverspec/rhel/module_spec.rb` and making the legacy one simply `require_relative ‘../../files/serverspec/rhel/module_spec.rb’`, with the actual path determined by workspace directory layout

assumes the availability of ssh between developer machine and target instance

which may change during secure environment provision

`https://tomcat.apache.org/tomcat-8.5-doc/appdev/web.xml.txt`

Serverspec through own Puppet module

serverspec being just a handful of text files plus a Ruby runtime – calls to be provisioned (rvm-hosted) through Puppet from archive and templates and an exec for `rake spec` on the instance then updates Puppet and Vagrant logs with the result. This remediates limitations

`rake spec` is directly in console and can be run explicitly after provision and the spec file edited in the instance. Debugging is easy.

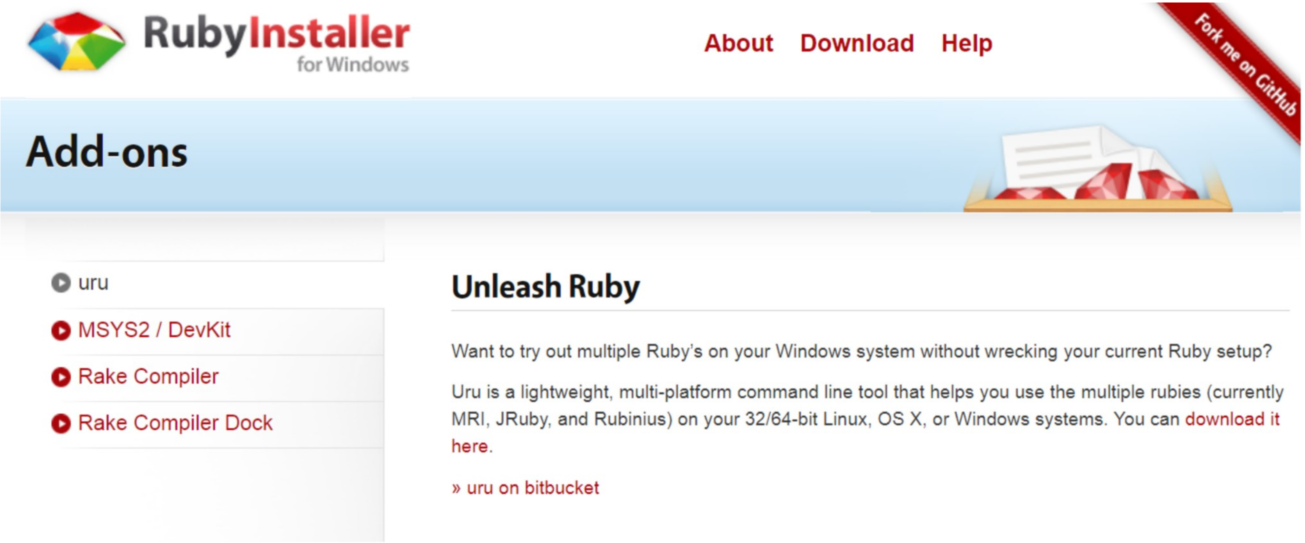
Spec file is generated by Puppet from template, hieradata etc. for version-sensitive portion
(one can also keep serverspec require relative for Vagrant runs)

Runs on DMZ machine after lockdown, the results pushed to the developer, CICD etc.

A little cumbersome to modify file locally and push to the vm to validate

`https://tomcat.apache.org/tomcat-8.5-doc/appdev/web.xml.txt`

Serverspec through own Puppet module



Add-ons

- uru
- MSYS2 / DevKit
- Rake Compiler
- Rake Compiler Dock

Unleash Ruby

Want to try out multiple Ruby's on your Windows system without wrecking your current Ruby setup?

Uru is a lightweight, multi-platform command line tool that helps you use the multiple rubies (currently MRI, JRuby, and Rubinius) on your 32/64-bit Linux, OS X, or Windows systems. You can download it here.

» uru on bitbucket

`https://rubyinstaller.org/`

Serverspec through own Puppet module

```
hierarchy:
  - name: 'Per-node data'
    path: "node/%{::trusted.certname}.yaml"
  - name: 'Per-role data'
    path: "role/%{trusted.extensions.pp_role}.yaml"
  - name: 'Shared data'
    path: common.yaml
```

```
context 'Packages' do
  {
    consul => nil,
    tomcat => '7.0.54-2',
    jdk    => '1.8.0.192'
  }. each do |name,version|
    describe package(name) do
      it { should be_installed.with_version(version)}
    end
  end
end
```

Puppet-RSpec

- Stubs target OS, environment facts, module parameters and hiera data
- Compiles and examines the 'Catalog'
- Asserts the specified actions are taken
- All that without requiring one to spawn the real instance
- Real provision will behave according to those specs

http://rspec-puppet.com
https://github.com/chefspec/chefspec/tree/master/examples

Puppet Rspec is useful with
module and profile development and refactoring
intelligent upgrade / downgrade logic is critical (present|latest|absent)
with complex module logic or generation of complex configurations

Chef Spec

- Performs the Puppet manifest / Chef cookbook compile check, allexceptions – will be also seen in real provision
- Dumps catalog through one of few available methods:

```
describe ‘module’ do
  it { should compile.with_all_deps}
  at_exit { RSPec::Puppet::Coverage.report! }
  it { p catalogue.resource}
end
```

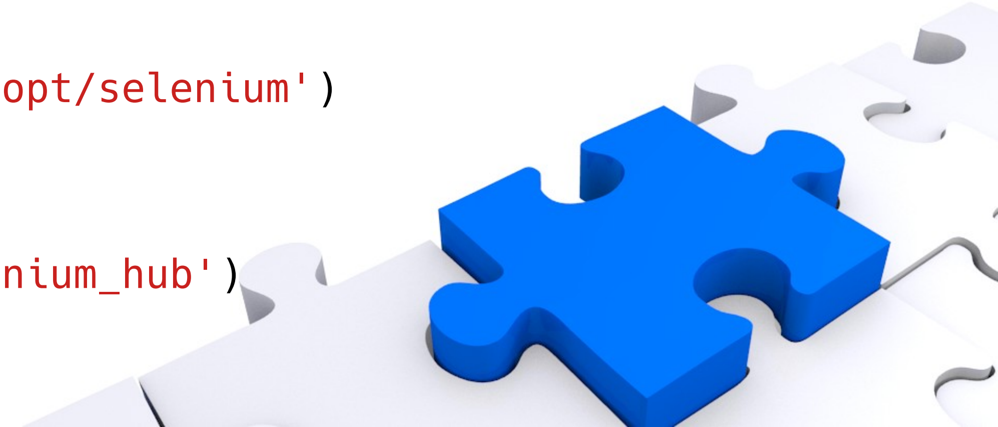
no example for Chef

Chef Spec

- Stubs target platform, ohai, cookbook attributes
- Performs ‘Converge’ and examines the catalog, asserts specified actions are taken

```
require 'chefspec'
require 'json'

describe 'selenium_hub::default' do
  let(:chef_run) { ChefSpec::SoloRunner.converge(described_recipe) }

  it 'creates selenium directory' do
    expect(chef_run).to create_directory('/opt/selenium')
  end

  it 'starts selenium hub service' do
    expect(chef_run).to start_service('selenium_hub')
  end
```

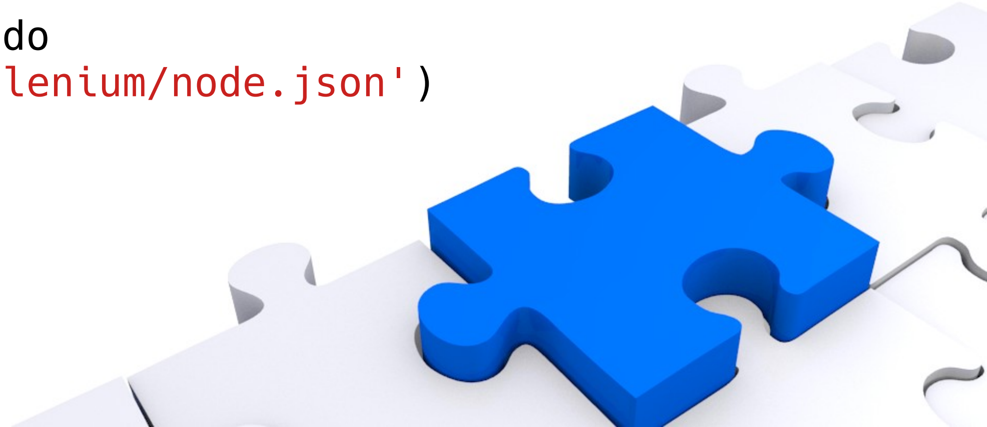
continued on the following slide



Chef Spec

Popular practice during module / cookbook development to run rspec to examine and fine tune resources produced from template, saving on a real Puppet / Chef run

```
describe 'selenium_hub::default' do
  let(:chef_run) { ChefSpec::SoloRunner.converge(described_recipe) }

  it 'generates init script configuration' do
    expect(chef_run).to render_file('/etc/init.d/selenium_hub')
  end
  it 'generates a valid json configuration' do
    expect(chef_run).to render_file('/opt/selenium/node.json')
    .with_content( satisfy do |content|
      JSON.parse(content) rescue nil
    end
    )
  end
end
```

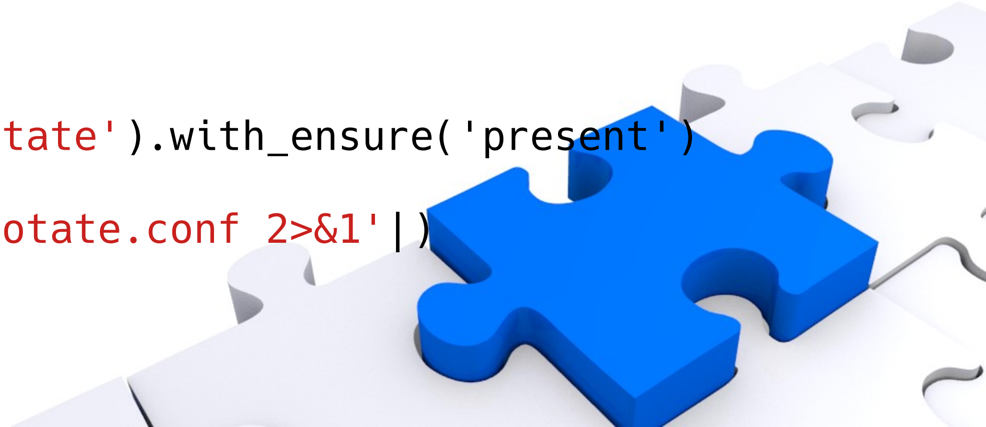


Pupper “defines” Spec

The Ruby DSL syntax is quite divergent

```
require 'spec_helper'
describe 'logrotate::cron' do
  # NOTE: verbatim Puppet code
  let(:pre_condition) { 'class { "::logrotate": }' }

  context 'with default params' do
    let(:title) { 'test' }
    let(:params) { { ensure: 'present' } }
      it {
        is_expected.to
          contain_file('/etc/cron.test/logrotate').with_ensure('present')
          .with_content(
           %r|'/usr/sbin/logrotate /etc/logrotate.conf 2>&1'|)
      }
  end
end
```

https://github.com/voxpupuli/puppet-logrotate



Pupper “defines” Spec

The Ruby DSL offers advanced ‘relationship’ expectation API, which unfortunately are not to be fully relied upon

```
it do
  should contain_file('c:/temp/log')
         .with(ensure => 'directory'
         .that_requires('c:/temp')
         .that_comes_before('Acl[c:/temp/log]')
end


it 'notifies service' do
    resource = chef_run.file(@setenv_file)
    expect(resource).to_not
      notify("service['#{@application_server}']").to(:restart).delayed
end
```

Questions?

Thank You

Wells Fargo Developer Gateway
developer.wellsfargo.com