Serverspec in cloud provision

Serguei Kouzmine
kouzmine_serguei@yahoo.com

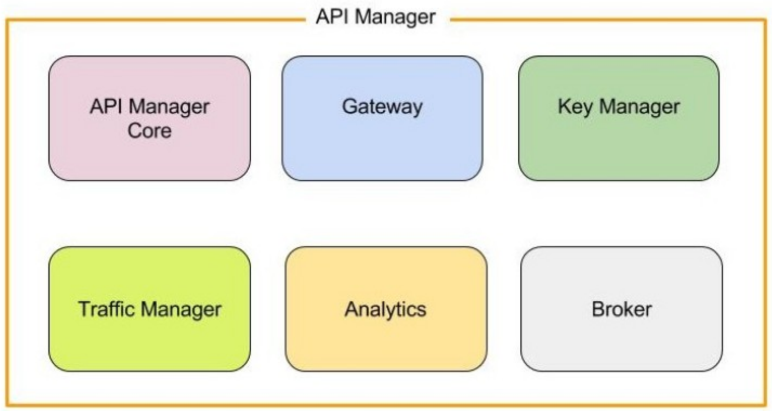
Serverspec in cloud provision

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

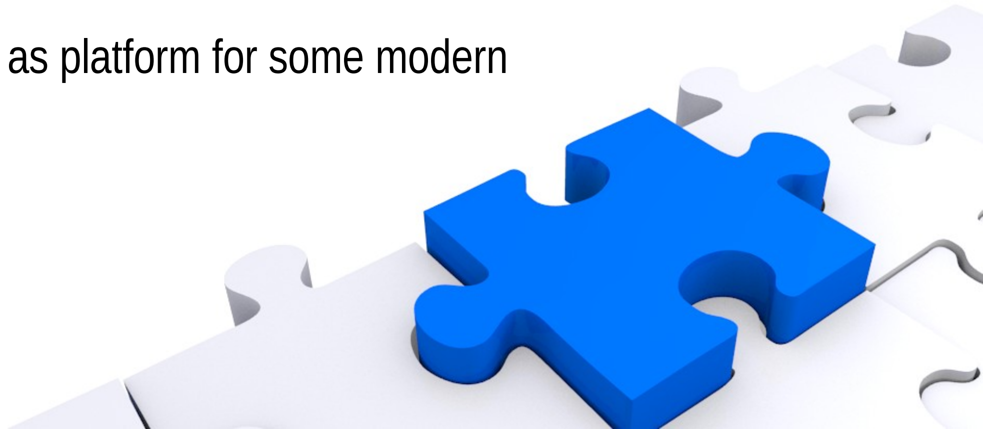
Often more critical than provision engine (Puppet, Chef, Ansible, DSC learning curve.

QA engineer using Selenium or Katalon often is or willing to grow as Web developer but unlikely ever interested in the code base of those tools.

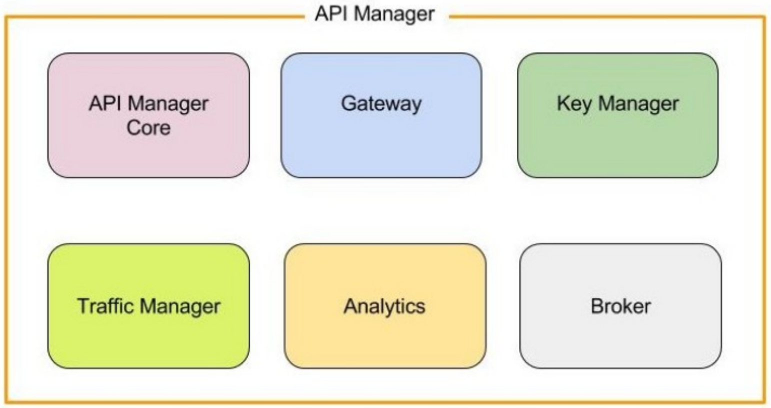
Serverspec in cloud provision

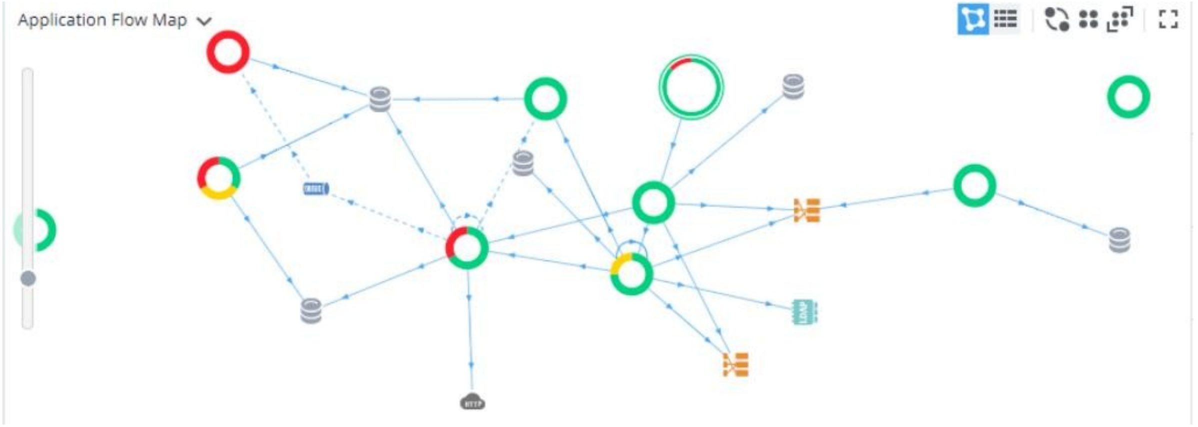


Cloud cluster provisioned by some automation workflow serves as platform for some modern application stack

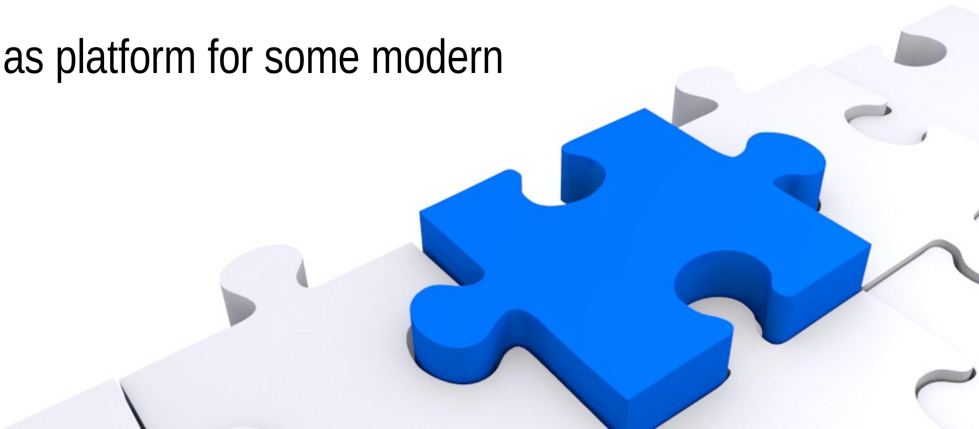


Serverspec in cloud provision

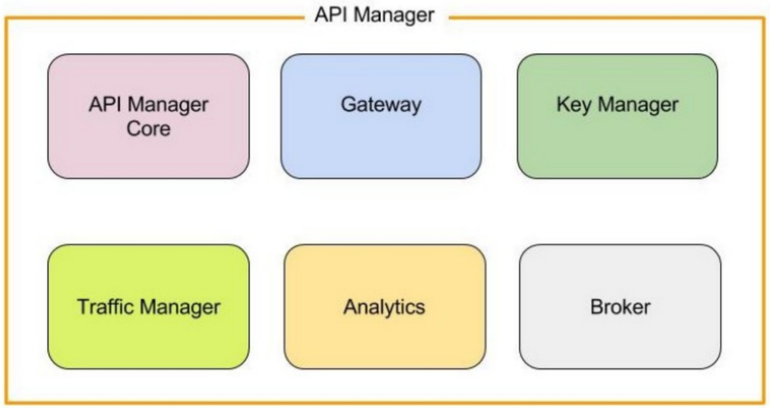


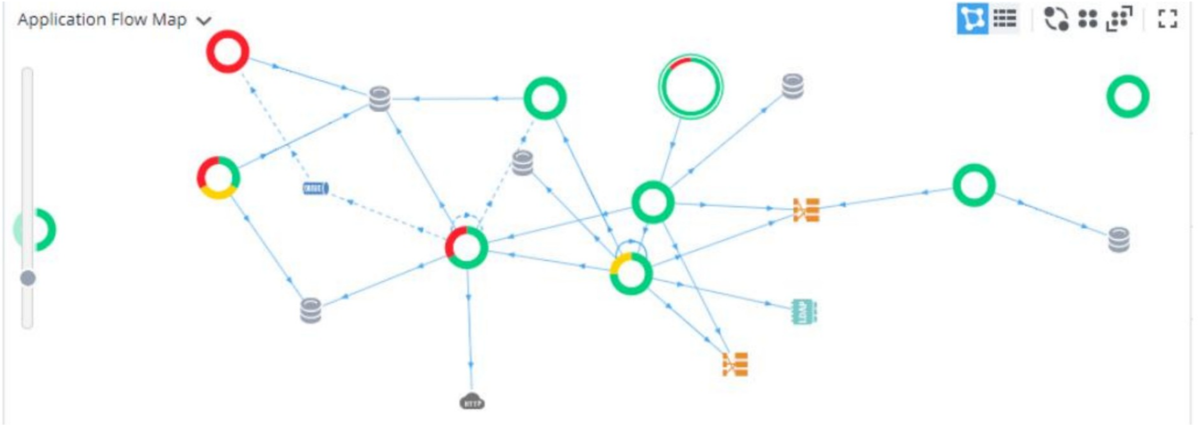


Cloud cluster provisioned by some automation workflow serves as platform for some modern application stack



Serverspec in cloud provision





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

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:

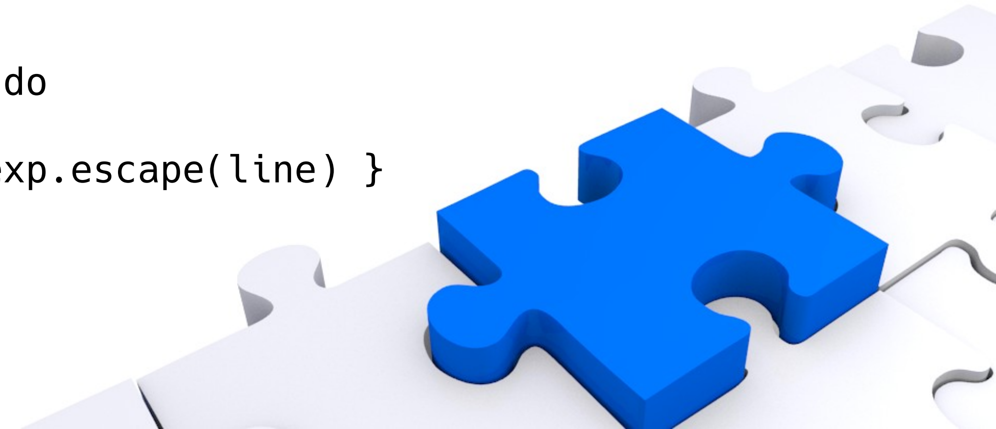
```
describe service(‘tomcat’) do
  it {should be_running }
end

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
```

It is shorter then the underlying command

```
describe command("/bin/gem list --local #{package_name}") do
     its(:stderr) { should be_empty }
     its(:exit_status) {should eq 0 }
     its(:stdout) { should match Regexp.new(package_version) }
end
```

A slighly less magical expectation

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

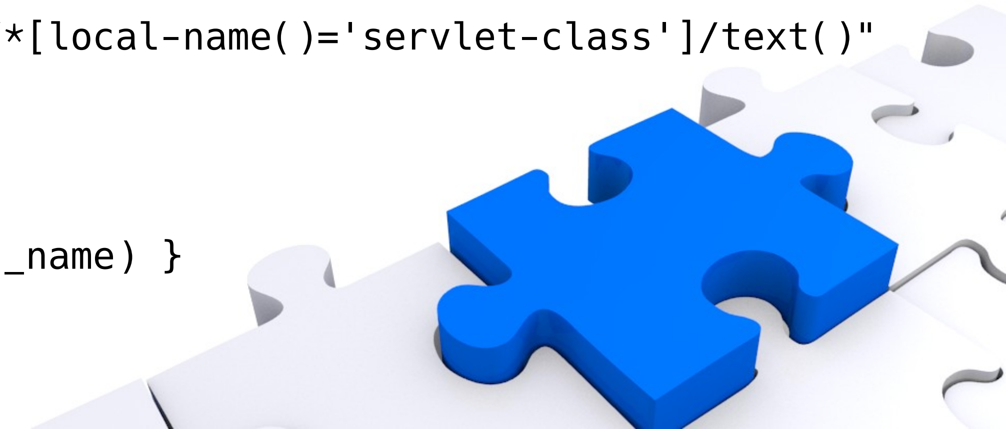


Eventually the *command* is where the tricky part is:

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

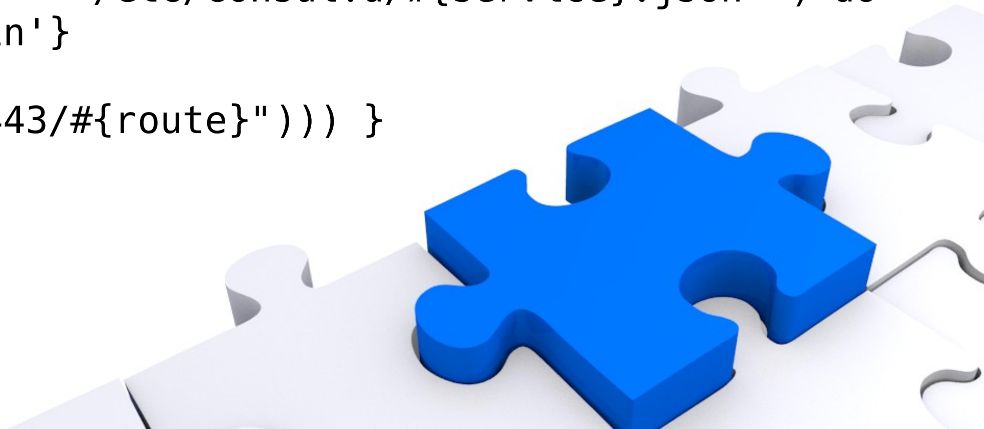
```
context 'Tomcat web.xml configuration' do
  class_name = 'com.mycompany.mypackage.ControllerServlet',
  describe command(<<-EOF
   xmllint --xpath "//*[local-name()='servlet']/*[local-name()='servlet-class']/text()"
#{web_xml}
  EOF
  ) do
    its(:exit_status) { should eq 0 }
    its(:stdout) { 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:

```
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

Serverspec resources

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

It is hosted on github in mizzy/serverspec, mizzy/specinfra ,vvchik/vagrant-serverspec, A very similar inspec/inspec framework exists for Chef.

A handful of active projects hosted in github present extended types, and app stack specific spec.

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 bundled with provision run making time consuming to rerun

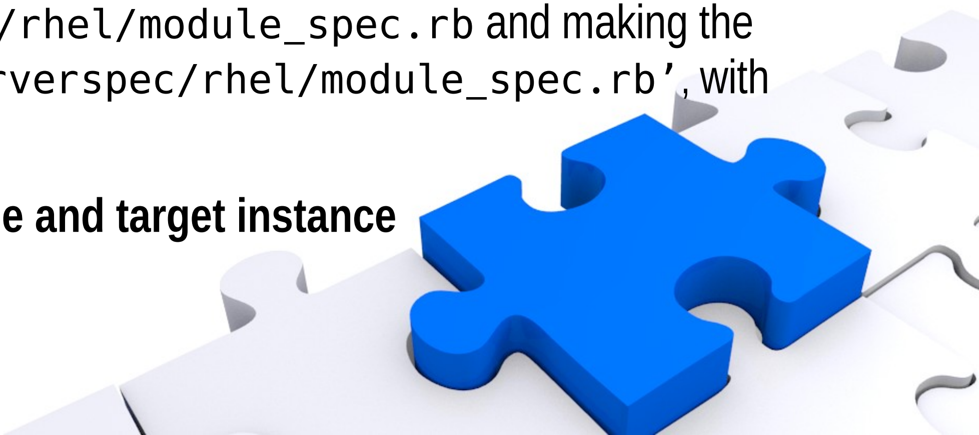
default output is super extra verbose

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 in cloud provision