Infrastructure Automation with Opscode Chef

http://opscode.com
@opscode
#opschef

OPSCODE | Chef

Who are we?

- Joshua Timberman
- Adam Jacob
- Christopher Brown
- Aaron Peterson
- Seth Chisamore
- Matt Ray

Who are you?

- System administrators?
- Developers?
- “Business” People?

http://www.flickr.com/photos/timyates/2854357446/sizes/l/

Hint, consultants, you’re “Business” people too.

OPSCODE | Chef

What are we talking about?

http://www.flickr.com/photos/peterkaminski/2174679908/

Managing infrastructure in the Cloud. With Chef, hopefully.

OPSCODE | Chef

Agenda

- How's and Why's
- Live Demo!
- Getting Started with Chef
- Anatomy of a Chef Run
- Managing Cloud Infrastructure
- Data Driven Shareable Cookbooks

http://www.flickr.com/photos/koalazymonkey/3590953001/

Tuesday, June 14, 2011

How's and why's of managing infrastructure with Chef.
We're running a live demo!
We'll walk through the things required to get started with Chef.
We will look at the anatomy of a Chef run in detail.
Since we've launched a cloud infrastructure, we'll want to know how we manage it.
We'll talk about our data driven sharable cookbooks.

Infrastructure as Code

The goal is fully automated infrastructure. In the cloud, anywhere. We get there with Infrastructure as Code.

OPSCODE | Chef

A technical domain revolving around building and managing infrastructure programmatically

Enable the reconstruction of the business from nothing but a source code repository, an application data backup, and bare metal resources.

OPSCODE | Chef

Configuration Management

Tuesday, June 14, 2011

Keep track of all the steps required to take bare metal systems to doing their job in the infrastructure.

It is all about the policy.

And this needs to be available as a service in your infrastructure.

System Integration

http://www.flickr.com/photos/opalsson/3773629074/

Taking all the systems that have been configured to do their job, and make them work together to actually run the infrastructure.

Tuesday, June 14, 2011

Introducing Chef.

Maybe you've already met!

Stephen Nelson-Smith has a great way to introducing Chef, so with apologies to him, I'm going to reuse his descriptions.

The Chef Framework

With thanks (and apologies) to Stephen Nelson-Smith

Chef provides a framework for fully automating infrastructure, and has some important design principles.

The Chef Framework

- Reasonability
- Flexibility
- Library & Primitives
- TIMTOWTDI

Chef makes it easy to reason about your infrastructure, at scale. The declarative Ruby configuration language is easy to read, and the predictable ordering makes it easy to understand what's going on.

Chef is flexible, and designed to allow you to build infrastructure using a sane set of libraries and primitives.

Just like Perl doesn't tell programmers how to program, Chef doesn't tell sysadmins how to manage infrastructure.

The Chef Tool(s)

With thanks (and apologies) to Stephen Nelson-Smith

Since Chef is a framework with libraries and primitives for building and managing infrastructure, it only makes sense that it comes with tools written for that purpose.

The Chef Tool(s)

- ohai
- chef-client
- knife
- shef

Ohai profiles the system to gather data about nodes and emits that data as JSON.
Chef client runs on your nodes to configure them.
Knife is used to access the API.
Shef is an interactive console debugger.

The Chef API

With thanks (and apologies) to Stephen Nelson-Smith

The Chef API provides a client/server service for configuration management in your infrastructure.

The Chef API

- RSA key authentication w/ Signed Headers
- RESTful API w/ JSON
- Search Service
- Derivative Services

The API itself is RESTful with JSON responses.

Part of the API is a dynamic search service which can be queried to provide rich data about the objects stored on the server.

Because it is flexible and built as a service, it is easy to build derivative services on top, including integration with other tools and services.

The Chef Community

With thanks (and apologies) to Stephen Nelson-Smith

As an Open Source project, the Chef community is critical.

The Chef Community

- Apache License, Version 2.0
- 360+ Individual contributors
- 70+ Corporate contributors
  - Dell, Rackspace, VMware, RightScale, Heroku, and more
- http://community.opscode.com
- 240+ cookbooks

Community is important.

http://apache.org/licenses/LICENSE-2.0.html
http://www.opscode.com/blog/2009/08/11/why-we-chose-the-apache-license/
http://wiki.opscode.com/display/chef/How+to+Contribute
http://wiki.opscode.com/display/chef/Approved+Contributors

OPSCODE | Chef

Chef Enables Infrastructure as Code

- Resources
- Recipes
- Roles
- Source Code

```
package "haproxy" do
  action :install
end

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => true
  action [:enable, :start]
end
```


Declare system configuration as idempotent resources.
Put resources together in recipes.
Assign recipes to systems through roles.
Track it all like source code.

Chef Resources

- Have a type.
- Have a name.
- Have parameters.
- Take action to put the resource in the declared state.
- Can send notifications to other resources.

```
package "haproxy" do
  action :install
end

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => true
  action [:enable, :start]
end
```

Resources take *action* through Providers

Providers know how to actually configure the resources to be in the declared state

package "haproxy"

```
yum install haproxy
apt-get install haproxy
pacman sync haproxy
pkg_add -r haproxy
```


The haproxy package resource may run any number of OS commands, depending on the node's platform.

Recipes are collections of Resources

Chef Recipes

- Recipes are evaluated for resources in the order they appear.
- Each resource object is added to the Resource Collection.

```
package "haproxy" do
  action :install
end

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  notifies :restart, "service[haproxy]"
end

service "haproxy" do
  supports :restart => true
  action [:enable, :start]
end
```

Chef Recipes

- Recipes can include other recipes.
- Included recipes are processed in order.

```
include_recipe "apache2"
include_recipe "apache2::mod_rewrite"
include_recipe "apache2::mod_deflate"
include_recipe "apache2::mod_headers"
include_recipe "apache2::mod_php5"
```


Just like recipes themselves are processed in order, the recipes included are processed in order, so when you include a recipe, all its resources are added to the resource collection, then Chef continues to the next.

Chef Recipes

- Extend recipes with Ruby.
- Iterate over an array of package names to install.

```
%w{ php5 php5-dev php5-cgi }.each do |pkg|

  package pkg do
    action :install
  end

end
```

Chef Recipes

```
template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  notifies :restart, "service[haproxy]"
end
```

- Good: Drop off a dynamic template.
- Better: Discover data through search.

```
pool_members = search("node", "role:mediawiki")

template "/etc/haproxy/haproxy.cfg" do
  source "haproxy.cfg.erb"
  owner "root"
  group "root"
  mode 0644
  variables :pool_members => pool_members
  notifies :restart, "service[haproxy]"
end
```

Chef Roles

- Roles describe nodes.
- Roles have a run list.
- Roles can have attributes.

```
name "mediawiki"
description "mediawiki app server"
run_list(
  "recipe[mysql::client]",
  "recipe[application]",
  "recipe[mediawiki::status]"
)
```

```
name "mediawiki_load_balancer"
description "mediawiki load balancer"
run_list(
  "recipe[haproxy::app_lb]"
)
override_attributes(
  "haproxy" => {
    "app_server_role" => "mediawiki"
  }
)
```

Track it like source code...

```
% git log
commit d640a8c6b370134d7043991894107d806595cc35
Author: jtimberman <joshua@opscode.com>

    Import nagios version 1.0.0

commit c40c818498710e78cf73c7f71e722e971fa574e7
Author: jtimberman <joshua@opscode.com>

    installation and usage instruction docs

commit 99d0efb024314de17888f6b359c14414fda7bb91
Author: jtimberman <joshua@opscode.com>

    Import haproxy version 1.0.1

commit c89d0975ad3f4b152426df219fee0bfb8eafb7e4
Author: jtimberman <joshua@opscode.com>

    add mediawiki cookbook

commit 89c0545cc03b9be26f1db246c9ba4ce9d58a6700
Author: jtimberman <joshua@opscode.com>

    multiple environments in data bag for mediawiki
```

LIVE DEMO!!!

git clone git://github.com/opscode/velocity2011-chef-repo

We thought we'd start with the live demo early on, since last year we were interrupted by a fire alarm.

Live Demo

- Behind the scenes we're building a new infrastructure
- Five nodes
  - Database master
  - Two App servers
  - Load Balanced
  - Monitored

http://www.flickr.com/photos/takomabibelot/3787425422

git clone git://github.com/opscode/velocity2011-chef-repo

During this workshop, we will build a cloud infrastructure before your very eyes (if we have multiple displays to show that while the slides are up.)

How did we get here?

git clone git://github.com/opscode/velocity2011-chef-repo

How did we get to the point where we can build a multi-tiered, monitored infrastructure?

Getting Started

- Opscode Hosted Chef
- Authentication Credentials
- Workstation Installation
- Source Code Repository

```
git clone git://github.com/opscode/velocity2011-chef-repo
```


We signed up for Opscode Hosted Chef, downloaded our authentication credentials (RSA private keys), installed Chef on our workstation and set up a source code repository.

Getting Started: Opscode Hosted Chef

- Sign up for Opscode Hosted Chef
  - https://community.opscode.com/users/new
- Sign into Management Console
  - https://manage.opscode.com
- Create an Organization

```
git clone git://github.com/opscode/velocity2011-chef-repo
```


The workshop installation instructions describe how to go about the process.

Getting Started: Authentication Credentials

- Download User Private Key
- Download Organization Validation Private Key
- Retrieve Cloud Credentials

```
git clone git://github.com/opscode/velocity2011-chef-repo
```


The signup process will provide instructions on how to retrieve your user private key and organization validation private key.

The examples in the chef repository will use Amazon EC2. You'll need the cloud credentials.

Getting Started: Workstation Installation

- Ruby (1.9.2 recommended)
- RubyGems 1.3.7+
- Chef
- Git

```
git clone git://github.com/opscode/velocity2011-chef-repo
```


Ruby 1.9.2 is recommended. It is higher performance, Chef works well with it and it comes with a reasonable, stable version of RubyGems, version 1.3.7.

Those that received the installation instructions will note that we're currently recommending RVM for workstation setup. This is not a recommendation for managed nodes.

We're working diligently on a full-stack installer for Chef, its in testing and will be done soon.

Getting Started: Source Code Repository

- Chef Repository for Velocity 2011
  - git://github.com/opscode/velocity2011-chef-repo
- Upload to Opscode Hosted Chef server
  - roles
  - data bags
  - cookbooks
  - environments

```
git clone git://github.com/opscode/velocity2011-chef-repo
```


The repository has a README-velocity.md file that describes how to Upload the Repository to the Opscode Hosted Chef server.

Working in the Repository

```
export ORGNAME="your_organization_name"
export OPSCODE_USER="your_opscode_username"
export AWS_ACCESS_KEY_ID="amazon aws access key id"
export AWS_SECRET_ACCESS_KEY="amazon aws secret access key"
export RACKSPACE_API_KEY="rackspace cloud api key"
export RACKSPACE_API_USERNAME="rackspace cloud api username"
% cd velocity2011-chef-repo
% cat .chef/knife.rb
% knife ec2 server list
% knife rackspace server list
% knife client list
```

git clone git://github.com/opscode/velocity2011-chef-repo

Export these variables with your cloud credentials.

The README in the repository contains these instructions too.

knife ec2 server create
OR!
knife rackspace server create

git clone git://github.com/opscode/velocity2011-chef-repo

With all that, we can run the series of knife ec2 server create commands. Nothing more than this to get fully automated infrastructure launched.

The file README-velocity.md contains all the commands needed to get started with launching infrastructure for yourself.

Anatomy of a Chef Run

```
% knife ec2 server create -G default -I ami-7000f019 -f m1.small \
  -S velocity-2011-aws -i ~/.ssh/velocity-2011-aws.pem -x ubuntu \
  -E production -r 'role[base],role[mediawiki_database_master]'
```


What happens when we run the knife command?

Anatomy of a Chef Run: EC2 Create

```
% knife ec2 server create -G default -I ami-7000f019 -f m1.small \
  -S velocity-2011-aws -i ~/.ssh/velocity-2011-aws.pem -x ubuntu \
  -E production -r 'role[base],role[mediawiki_database_master]'

Instance ID: i-8157d9ef
Flavor: m1.small
Image: ami-7000f019
Availability Zone: us-east-1a
Security Groups: default
SSH Key: velocity-2011-aws

Waiting for server.................................
Public DNS Name: ec2-50-17-117-98.compute-1.amazonaws.com
Public IP Address: 50.17.117.98
Private DNS Name: ip-10-245-87-117.ec2.internal
Private IP Address: 10.245.87.117

Waiting for sshd....done
Bootstrapping Chef on ec2-50-17-117-98.compute-1.amazonaws.com
```


The knife ec2 server create command makes a call to the Amazon EC2 API through fog[0] and waits for SSH.

There's a lot here to type, so you can copy/paste out of the README-velocity.md.

[0]: http://rubygems.org/gems/fog

Anatomy of a Chef Run: Bootstrap

```
Successfully installed mixlib-authentication-1.1.4
Successfully installed mime-types-1.16
Successfully installed rest-client-1.6.3
Successfully installed bunny-0.6.0
Successfully installed json-1.5.1
Successfully installed polyglot-0.3.1
Successfully installed treetop-1.4.9
Successfully installed net-ssh-2.1.4
Successfully installed net-ssh-gateway-1.1.0
Successfully installed net-ssh-multi-1.0.1
Successfully installed erubis-2.7.0
Successfully installed moneta-0.6.0
Successfully installed highline-1.6.2
Successfully installed uuidtools-2.1.2
Successfully installed chef-0.10.0
15 gems installed
```


After the system is available in EC2 and SSH is up, the “bootstrap” process takes over. Chef is installed.

Anatomy of a Chef Run: Validation

```
(
cat <<'EOP'
<%= validation_key %>
EOP
) > /tmp/validation.pem
awk NF /tmp/validation.pem > /etc/chef/validation.pem
rm /tmp/validation.pem
```


The bootstrap will write out the validation certificate from the local workstation to the target system.

Anatomy of a Chef Run: Configuration

```
(
cat <<'EOP'
<%= config_content %>
EOP
) > /etc/chef/client.rb
```


The chef client configuration file is written based on values from the local system.

The bootstrap is done from a template you can customize, so you can change the content in the EOP to whatever client.rb you want.

/etc/chef/client.rb

```
log_level        :info
log_location     STDOUT
chef_server_url  "https://api.opscode.com/organizations/velocitydemo"
validation_client_name "velocitydemo-validator"
node_name "i-138c137d"
```


For example, this is all it takes to configure the Chef Client on the new system.

Anatomy of a Chef Run: Run List

```
(
cat <<'EOP'
<%= { "run_list" => @run_list }.to_json %>
EOP
) > /etc/chef/first-boot.json
```

Anatomy of a Chef Run: chef-client

```
chef-client -j /etc/chef/first-boot.json

# run with debug output for full detail:

chef-client -j  /etc/chef/first-boot.json -l debug
```


Normally we just run chef-client with info level log output. To get more detail, I ran it with debug.

The -l debug option is available any time you want more detailed output from Chef.

Anatomy of a Chef Run: Ohai!

```
INFO: *** Chef 0.10.0 ***
DEBUG: Loading plugin os
DEBUG: Loading plugin kernel
DEBUG: Loading plugin ruby
DEBUG: Loading plugin languages
DEBUG: Loading plugin hostname
DEBUG: Loading plugin linux::hostname
...
DEBUG: Loading plugin ec2
DEBUG: has_ec2_mac? == true
DEBUG: can_metadata_connect? == true
DEBUG: looks_like_ec2? == true
DEBUG: Loading plugin rackspace
...
DEBUG: Loading plugin cloud
```


Chef runs ohai, the system profiling and data gathering tool. Ohai automatically detects a number of attributes about the system it is running on, including the kernel, operating system/platform, hostname and more.

Run Ohai

- Run `ohai | less` on your system.
- Marvel at the amount of data it returns.

You can run `ohai` on your local system with Chef installed to see what Chef discovers about it.

Anatomy of a Chef Run: Authenticate

```
INFO: Client key /etc/chef/client.pem is not present -
registering

DEBUG: Signing the request as velocitydemo-validator

DEBUG: Sending HTTP Request via POST to api.opscode.com:443/
organizations/velocitydemo/clients

DEBUG: Registration response: {"uri"=>"https://
api.opscode.com/organizations/velocitydemo/clients/
i-8157d9ef", "private_key"=>"SNIP!"}
```


If /etc/chef/client.pem is not present, the validation client is used to register a new client automatically.

The response comes back with the private key, which is written to /etc/chef/client.pem. All subsequent API requests to the server will use the newly created client, and the /etc/chef/validation.pem file can be deleted (we have chef-client::delete_validation for this).

Yes, the client's private key is displayed. Be mindful of this when pasting debug output.

* http://tickets.opscode.com/browse/CHEF-2238

Anatomy of a Chef Run: Build Node

```
DEBUG: Building node object for i-8157d9ef
DEBUG: Signing the request as i-8157d9ef
DEBUG: Sending HTTP Request via GET to api.opscode.com:443/
organizations/velocitydemo/nodes/i-8157d9ef
INFO: HTTP Request Returned 404 Not Found: Cannot load node
i-8157d9ef
DEBUG: Signing the request as i-8157d9ef
DEBUG: Sending HTTP Request via POST to api.opscode.com:443/
organizations/velocitydemo/nodes
DEBUG: Extracting run list from JSON attributes provided on
command line
INFO: Setting the run_list to ["role[base]", "role
[mediawiki_database_master]"] from JSON
DEBUG: Applying attributes from json file
DEBUG: Platform is ubuntu version 10.04
```


We have 3 important pieces of information about building the node object at this point. First, the instance ID is used as the node name. This is automatically set up as the default node name by knife ec2 server create.

Second, the JSON file passed into chef-client determines the run list of the node.

Finally, during the ohai data gathering, it determined that the platform of the system is Ubuntu 10.04. This is important for how our resources will be configured by the underlying providers.

Anatomy of a Chef Run: Sync Cookbooks

```
INFO: Run List is [role[base], role
[mediawiki_database_master]]

INFO: Run List expands to [apt, zsh, users::sysadmins, sudo,
git, build-essential, database::master]

INFO: Starting Chef Run for i-8157d9ef

DEBUG: Synchronizing cookbooks

INFO: Loading cookbooks [apt, aws, build-essential,
database, git, mysql, openssl, runit, sudo, users, xfs, zsh]
```


Once the run list is determined, it is expanded to find all the recipes that will be applied. The names of the recipes indicate which cookbooks are required, and those cookbooks are downloaded.

Cookbooks are like packages, so sometimes they depend on another which may not show up in the run list. Dependencies can be declared in cookbook metadata, similar to packaging system metadata for packages.

Anatomy of a Chef Run: Load Cookbooks

- Chef loads cookbook components after they are downloaded.
- Libraries
- Providers
- Resources
- Attributes
- Definitions
- Recipes

Once all the cookbooks have been downloaded, Chef will load the Ruby components of the cookbook. This is done in the order above.

Anatomy of a Chef Run: Load Recipes

```
DEBUG: Loading Recipe zsh via include_recipe
DEBUG: Found recipe default in cookbook zsh
DEBUG: Loading Recipe users::sysadmins via include_recipe
DEBUG: Found recipe sysadmins in cookbook users

DEBUG: Sending HTTP Request via GET to api.opscode.com:443/
organizations/velocitydemo/search/users
```


When recipes are loaded, the Ruby code they contain is evaluated. This is where things like search will hit the server API. We'll see more of this later on.

Chef is building what we call the "resource collection", an ordered list of all the resources that should be configured on the node.

Order Matters

The order of the run list and the order of resources in recipes is important, because it matters how your systems are configured. A half configured system is a broken system, and a system configured out of order may be a broken system. Chef's implicit ordering makes it easy to reason about the way systems are built, so you can identify and troubleshoot this easier.

Anatomy of a Chef Run: Convergence

```
user u['id'] do
  uid u['uid']
  gid u['gid']
  shell u['shell']
  comment u['comment']
  supports :manage_home => true
  home home_dir
end

directory "#{home_dir}/.ssh" do
  owner u['id']
  group u['gid'] || u['id']
  mode "0700"
end

template "#{home_dir}/.ssh/authorized_keys" do
  source "authorized_keys.erb"
  owner u['id']
  group u['gid'] || u['id']
  mode "0600"
  variables :ssh_keys => u['ssh_keys']
end
```


For example, our users::sysadmins recipe creates some resources for each user it finds from the aforementioned search.

These resources are added to the resource collection in the specified order. This is repeated for every user.

Anatomy of a Chef Run: Convergence

```
INFO: Processing user[velocity] action create
(users::sysadmins line 41)

INFO: Processing directory[/home/velocity/.ssh] action
create (users::sysadmins line 51)

INFO: Processing template[/home/velocity/.ssh/
authorized_keys] action create (users::sysadmins line 57)
```


Convergence is the phase when the resources in the resource collection are configured. Providers take the appropriate action. Users are created, packages are installed, services are started and so on.

Anatomy of a Chef Run: Save Node

```
DEBUG: Saving the current state of node i-8157d9ef
DEBUG: Signing the request as i-8157d9ef
DEBUG: Sending HTTP Request via PUT to api.opscode.com:443/
organizations/velocitydemo/nodes/i-8157d9ef
```


At the end of a run, the state of the node is saved, including all the attributes that were applied to the node from:

* ohai
* roles
* cookbooks
* environment

This data is also indexed by the server for search.

Anatomy of a Chef Run: Report Handlers

```
INFO: Running report handlers
INFO: Report handlers complete

... OR ...

ERROR: Running exception handlers
FATAL: Saving node information to /var/chef/cache/failed-run-data.json
ERROR: Exception handlers complete
FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
FATAL: Some unhandled Ruby exception message here.
```


At the end of the Chef run, report and exception handlers are executed.

Report handlers are executed on a successful run.

Exception handlers are executed on an unsuccessful run.

* stack trace data and state of the failed run are also saved to files on the filesystem, and reported.

I can haz cloud?

http://www.flickr.com/photos/felixmorgner/4347750467/

Tuesday, June 14, 2011

Once a node is saved on the server, it is considered a managed system. In Chef, nodes do all the heavy lifting. All the above happens on the node, the server just handles API requests and serves data/cookbooks.

knife node show

```
% knife node show i-cda03aa3
Node Name:   i-cda03aa3
Environment: production
FQDN:        ip-10-112-85-253.ec2.internal
IP:          10.112.85.253
Run List:    role[base], role[monitoring]
Roles:       monitoring, base
Recipes      apt, zsh, users::sysadmins, sudo, git, build-
essential, nagios::client, nagios::server
Platform:    ubuntu 10.04
% knife node show i-cda03aa3 -m # non-automatic attributes
% knife node show i-cda03aa3 -l # all attributes
% knife node show i-cda03aa3 -Fj # JSON output
```


We can show the nodes we have configured!

Data Driven

The deployment is data driven. Besides the data that came from the roles which we're about to see, we also have arbitrary data about our infrastructure, namely the application we're deploying and the users we're creating.

We didn't have to write or modify any code to get a fully functional infrastructure.

Writing Data Driven Cookbooks

- Focus on primitives.
- Apply the desired system state / behavior.
- Don't hardcode data.
  - Attributes
  - Data bags
  - Search

Data Driven Deployment

```
data_bags
├── apps
│   └── mediawiki.json
└── users
    ├── nagiosadmin.json
    └── velocity.json
```


We encapsulate all the information about our application, including environment-specific details. We also have two users we're creating.

Each Instance Has a Role

```
roles
├── base.rb
├── mediawiki.rb
├── mediawiki_database_master.rb
├── mediawiki_load_balancer.rb
└── monitoring.rb
```

Two app servers!

All Your Base...

Base Role

```
% knife role show base
chef_type:              role
default_attributes:     {}
description:            Base role applied to all nodes.
env_run_lists:          {}
json_class:             Chef::Role
name:                   base
override_attributes:
  authorization:
    sudo:
      passwordless:  true
      users:         ["ubuntu"]
  nagios:
    server_role:  monitoring
run_list:               recipe[apt], recipe[zsh], recipe
[users::sysadmins], recipe[sudo], recipe[git], recipe[build-
essential]
```


The base role is going to apply some settings that are common across the entire infrastructure. For example, apt ensures apt caches are updated, zsh installs the Z shell in case any users want it. Users::sysadmins creates all the system administrator users. Sudo sets up sudo permissions. Git ensures that our favorite version control system is installed. Build essential ensures that we can build our application, RubyGem native extensions, or other tools that should be installed by compilation.

Packages vs Source

Lean into it.

The base role installs build-essential. You may opt to only have packages. Build your infrastructure the way you want :).

We're not going to have a holy war of packages vs source.

Come to DevOpsDays Mountain View for a panel discussion on this topic.

Nagios Server

Every well built infrastructure needs monitoring. We've set up Nagios for our monitoring system. We could also add another tool such as munin to the mix if we wanted - there's a munin cookbook that is data driven too.

Nagios Server

```
% knife role show monitoring
chef_type:            role
default_attributes:
  nagios:
    server_auth_method:  htauth
description:          Monitoring Server
env_run_lists:        {}
json_class:           Chef::Role
name:                 monitoring
override_attributes:  {}
run_list:             recipe[nagios::server]
```


We've modified the default behavior of the cookbook to enable htauth authentication.

Load Balancer

Load Balancer

```
% knife role show mediawiki_load_balancer
chef_type:            role
default_attributes:   {}
description:          mediawiki load balancer
env_run_lists:        {}
json_class:           Chef::Role
name:                 mediawiki_load_balancer
override_attributes:
  haproxy:
    app_server_role:  mediawiki
run_list:             recipe[haproxy::app_lb]
```


We’re using haproxy, and we’ll search for a specific application to load balance. The recipe is written to search for the mediawiki role to find systems that should be pool members.

MediaWiki App Servers (two)

We actually have just the one system, we'll add another one shortly :).

MediaWiki App Servers

```
% knife role show mediawiki
chef_type:            role
default_attributes:   {}
description:          mediawiki front end application
server.
env_run_lists:        {}
json_class:           Chef::Role
name:                 mediawiki
override_attributes:  {}
run_list:             recipe[mysql::client], recipe
[application], recipe[mediawiki::status]
```


The main thing in this role is the application recipe.

The recipe will read in data from the data bag (in a predefined format) to determine what kind of application to deploy, the repository where it lives, details on where to put it, what roles to search for to find the database, and many more customizable properties.

We launched two of these to have something to load balance :).

Application Data Bag Item

```
{
  "id": "mediawiki",
  "server_roles": [
    "mediawiki"
  ],
  "type": {
    "mediawiki": [
      "php",
      "mod_php_apache2"
    ]
  },
  "database_master_role": [
    "mediawiki_database_master"
  ],
  "repository": "git://github.com/mediawiki/mediawiki-trunk-
phase3.git",
  "revision": {
    "production": "master",
    "staging": "master"
  },
...
```

Database Master

Every database backed application needs a master database. For this simple example we haven't done any complex setup of master/slave replication, but the recipes are built such that this would be relatively easy to add.

Database Master

```
% knife role show mediawiki_database_master
default_attributes:   {}
description:          database master for the mediawiki
application.
env_run_lists:        {}
json_class:           Chef::Role
name:                 mediawiki_database_master
override_attributes:  {}
run_list:             recipe[database::master]
```


The database master recipe will read the application information from the data bag and use it to create the database so the application can store its data.

Cookbooks are easy to share.

Tuesday, June 14, 2011

Chef is designed such that cookbooks are easy to share. Data is easy to separate from logic in recipes by using Attributes and Chef's rich data discovery and look up features such as data bags.

Data Driven Cookbooks

- application & database
- nagios
- users

Through data bag modification, role settings and Chef's search feature, these cookbooks are data driven. No code was modified. You didn't have to understand Ruby (though we think its a good idea :)), and you can deploy an infrastructure quickly and easily.

Open Source Cookbooks

```
knife cookbook site install nagios
knife cookbook site install git
knife cookbook site install application
knife cookbook site install database
knife cookbook site install haproxy
knife cookbook site install sudo
knife cookbook site install users
knife cookbook site install zsh
```


The cookbooks directory contains all the cookbooks we need.

These do all kinds of things we didn't have to write.

These cookbooks all came from community.opscode.com

Application-specific Cookbooks

```
knife cookbook create mediawiki

$EDITOR cookbooks/mediawiki/recipes/db_bootstrap.rb
```


Your application probably doesn't have a specific cookbook already shared by the community.

We create our mediawiki cookbook for application specific purposes.

mediawiki::db_bootstrap

```
app = data_bag_item("apps", "mediawiki")
dbm = search(:node, "role:mediawiki_database_master")
db = app['databases'][node.chef_environment]

execute "db_bootstrap" do
  command <<-EOH
  /usr/bin/mysql \
    -u #{db['username']} \
    -p#{db['password']} \
    -h #{dbm['fqdn']} \
    #{db['database']} \
    < #{Chef::Config[:file_cache_path]}/schema.sql"
  EOH
  action :run
end
```


We retrieve some data up front.

Then we use it to configure a resource.

Tuesday, June 14, 2011

The systems we manage are running their own services to fullfill their purpose in the infrastructure. Each of those services is network accessible, and by expressing our systems through rich metadata, we can discover the systems that fullfill each role through searching the chef server.

Search for Nodes with Knife

```
% knife search node role:mediawiki_database_master
1 items found

Node Name:   i-8157d9ef
Environment: production
FQDN:        ip-10-245-87-117.ec2.internal
IP:          10.245.87.117
Run List:    role[base], role[mediawiki_database_master]
Roles:       mediawiki_database_master, base
Recipes      apt, zsh, users::sysadmins, sudo, git, build-
essential, database::master
Platform:    ubuntu 10.04
```

Search for Nodes in Recipes

```
results = search (:node, "role:mediawiki_database_master")

template "/srv/mediawiki/shared/LocalSettings.php" do
  source "LocalSettings.erb"
  mode "644"
  variables(
    :path => "/srv/mediawiki/current",
    :host => results[0]['fqdn']
  )
end
```


You no longer need to track which system has an IP that should be applied as the database master. We can just use its fqdn from a search.

Managing Infrastructure: Knife SSH

```
% knife ssh 'role:mediawiki_database_master' 'sudo chef-client' -a ec2.public_hostname -x ubuntu
ec2-50-17-117-98 INFO: *** Chef 0.10.0 ***
ec2-50-17-117-98 INFO: Run List is [role[base], role[mediawiki_database_master]]
ec2-50-17-117-98 INFO: Run List expands to [apt, zsh, users::sysadmins, sudo, git, build-essential, database::master]
ec2-50-17-117-98 INFO: Starting Chef Run for i-8157d9ef
ec2-50-17-117-98 INFO: Loading cookbooks [apt, aws, build-essential, database, git, mysql, openssl, runit, sudo, users, xfs, zsh]
ec2-50-17-117-98 INFO: Chef Run complete in 9.471502 seconds
ec2-50-17-117-98 INFO: Running report handlers
ec2-50-17-117-98 INFO: Report handlers complete
```

What port is haproxy admin again?

```
% knife ssh role:mediawiki_load_balancer -a ec2.public_hostname \
  'netstat -an | grep LISTEN'
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22002           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5666            0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
```


Oh that's right. I always forget how many 2's and 0's.

Managing Nodes through an API

```
knife node run list add NODE "recipe[mediawiki::api_update]"
knife exec -E 'nodes.transform("role:mediawiki") \
  {|n| n.run_list << "recipe[mediawiki::api_update]"}'
knife ssh 'role:mediawiki' -x velocity 'sudo chef-client' \
  -a cloud.public_hostname
```


We can programmatically add a recipe to the run list of all our nodes through the server API.

Manage Infrastructure: Knife SSH

- "SSH In a For Loop" is bad right?
- Parallel command execution.
- SSH is industry standard.
- Use sudo NOPASSWD.

"Best practice" suggests that ssh in a for loop is bad, because the prevailing idea is we're doing "one-off" changes.

We're actually working toward parallel command execution. Kick off a chef-client run on a set of nodes, or gather some kind of command output.

SSH is an industry standard that everyone understands and knows how to set up.

A security best practice is to use sudo with NOPASSWD, which is e.g. how the Ubuntu AMIs are set up by Canonical.

Wrap-up

- Infrastructure as Code
- Getting Started with Chef
- Anatomy of a Chef Run
- Data Driven Shareable Cookbooks
- Managing Cloud Infrastructure

We've covered a lot of topics today! I'm sure you have questions...

FAQ: Chef vs [Other Tool]

http://www.flickr.com/photos/gesika22/4458155541/

Tuesday, June 14, 2011

We can have that conversation over a pint :).

FAQ: How do you test recipes?

FAQ: Testing

- You launch cloud instances and watch them converge.
- You use Vagrant with a Chef Provisioner

We test recipes by running chef-client. Chef environments prevent recipe errors from affecting production.

Or, you buy Stephen Nelson-Smith's book!

FAQ: Testing

- You buy Stephen Nelson-Smith's book!

FAQ: How does Chef scale?

FAQ: Scale

- The Chef Server is a publishing system.
- Nodes do the heavy lifting.
- Chef scales like a service-oriented web application.
- Opscode Hosted Chef was designed and built for massive scale.

http://www.flickr.com/photos/amagill/61205408/

OPSCODE | Chef

Questions?

- http://opscode.com
- http://wiki.opscode.com
- @opscode, #opschef
- irc.freenode.net, #chef, #chef-hacking
- http://lists.opscode.com
- We're in the exhibit hall this week.
- We'll be at DevOpsDays Mountain View.

http://www.flickr.com/photos/oberazzi/318947873/

Thanks!

http://opscode.com
@opscode
#opschef