



OPSCODE

# *Understanding LWRP*

*Speaker:*

**Joshua Timberman** Technical Evangelist

- ▶ [joshua@opscode.com](mailto:joshua@opscode.com)
- ▶ [@jtimberman](https://twitter.com/jtimberman)
- ▶ [www.opscode.com](http://www.opscode.com)



Copyright © 2010 Opscode, Inc - All Rights Reserved

link remote\_file  
cookbook\_file service  
template ruby\_block  
execute

# Chef manages **Resources** on Nodes

package bash git log  
user deploy http\_request



Resources are an abstraction we feed data into. When you write recipes in Chef, you create resources of things you want to configure. You describe the state of some part of the system. We're going to talk about how this works in depth.

```
gem_package "bluepill"
```



```
package "bluepill" do
  provider Chef::Provider::Package::Rubygems
end
```

```
package "bluepill" do
  provider gem_package
end
```



```
template "/etc/bluepill/dnscache.pill" do
  source "dnscache.pill.erb"
  mode 0644
end
```

```
bluepill_service "dnscache" do  
  action [:enable, :load, :start]  
end
```



# Resources take *action* through Providers



```
service "dnscache" do
  provider bluepill_service
  action [:enable, :load, :start]
end
```



# Chef's **Recipe DSL** creates resources



Recipes evaluated in **two phases**

**Compile Phase**  
**Execution Phase**



# Compile Phase



# Resources added to ResourceCollection



# Chef::ResourceCollection

**Resources and definitions**  
**Hash of indexed resources**  
**Definitions are replaced**



Chef “recognizes” resources and definitions through `method_missing`. These are created as a hash of numerically indexed resources. Definitions are replaced entirely by the resources they contain.

```
@resources_by_name={"file[/tmp/something]"=>0}
```



## Loads cookbook components

- ▶ **Libraries**
- ▶ **Providers**
- ▶ **Resources**
- ▶ **Attributes**
- ▶ **Definitions**
- ▶ **Recipes**

# Execution Phase



Once everything is loaded up and the recipes have been parsed for the resources they contain, the runner starts the execution phase.



# Chef::Runner converges the node



# Chef::Resource calls run\_action



# Provider is chosen by **Chef::Platform**



# Chef::Platform finds the provider

**Specifically named parameter**  
**Platform assigned providers**  
**Matching the resource name**



Specifically named providers are like we saw earlier where the resource itself had the provider parameter.

Platform assigned are the ones like apt/yum for packages.

Matching the resource name is how LWRPs get chosen.

# Cookbook LWRP



# Ruby DSL for writing Resources & Providers



# Resources have states



# Resources have states

## Resource DSL describe states

## Two possible states

- ▶ **Current (@current\_resource)**
- ▶ **Desired (@new\_resource)**

## Providers load\_current\_resource



The resource describes the state that some aspect of the system should be in by passing parameters that configure it for that state.

Current and desired are the two states that a resource can be in.

Providers load\_current\_resource determines what state the resource is in.



```

def load_current_resource
  @bp = Chef::Resource::BluepillService.new(new_resource.name)
  @bp.service_name(new_resource.service_name)

  Chef::Log.debug("Checking status of service #{new_resource.service_name}")

  begin
    if run_command_with_systems_locale(:command => "#{node['bluepill']['bin']} status #{
new_resource.service_name}") == 0
      @bp.running(true)
    end
  rescue Chef::Exceptions::Exec
    @bp.running(false)
  end
  nil
end

if ::File.exists?("#{node['bluepill']['conf_dir']}/#{new_resource.service_name}.pill")
  && ::File.symlink?("#{node['bluepill']['init_dir']}/#{new_resource.service_name}")
  @bp.enabled(true)
else
  @bp.enabled(false)
end
end
end

```

The code is not important. What is important that we're checking for some various states on the system by running commands or code and setting "state" parameters.

# Resource DSL

```
actions :start, :stop, :enable, :disable, :load, :restart

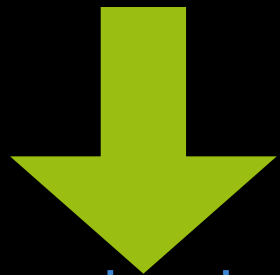
attribute :service_name, :name_attribute => true
attribute :enabled, :default => false
attribute :running, :default => false
attribute :variables, :kind_of => Hash
attribute :supports, :default => { :restart => true, :status => true }
```

# Provider DSL

```
action :start do
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

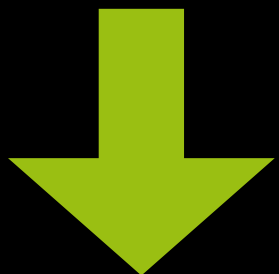


The provider DSL at a minimum has action methods. The action methods handle doing whatever is needed to configure the resource to be in the declared state. Earlier we set @bp\_running to true or false depending on the current state, here we check.



```
actions :start, :stop, :enable, :disable, :load, :restart
```

```
attribute :service_name, :name_attribute => true
attribute :enabled, :default => false
attribute :running, :default => false
attribute :variables, :kind_of => Hash
attribute :supports, :default => { :restart => true, :status => true }
```



```
action :start do
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

```
actions :start, :stop, :enable, :disable, :load, :restart
attribute :service_name, :name_attribute => true
attribute :enabled, :default => false
attribute :running, :default => false
attribute :variables, :kind_of => Hash
attribute :supports, :default => { :restart => true, :status => true }
```

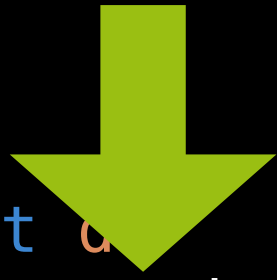
```
def load_current_resource
  @bp = Chef::Resource::BluepillService.new(new_resource.name)
  @bp.service_name(new_resource.service_name)

  Chef::Log.debug("Checking status of service #{new_resource.service_name}")

  begin
    if run_command_with_systems_locale(:command => "#{node['bluepill']['bin']} status #{new_resource.service_name}") == 0
      @bp.running(true)
    end
  rescue Chef::Exceptions::Exec
    @bp.running(false)
    nil
  end

  if ::File.exists?("#{node['bluepill']['conf_dir']}/#{new_resource.service_name}.pill")
    && ::File.symlink?("#{node['bluepill']['init_dir']}/#{new_resource.service_name}")
    @bp.enabled(true)
  else
    @bp.enabled(false)
  end
end
```





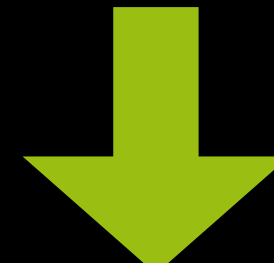
```
action :start do
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

# Chef Resources!



```
action start do
  unless @bp.running
    execute "/usr/bin/bluepill start #{new_resource.service_name}"
  end
end
```

# New resource parameters



# LWRPs in Opscode Cookbooks

**aws\_ebs\_volume**

**aws\_elastic\_ip**

**bluepill\_service**

**daemontools\_service**

**dynect\_rr**

**mysql\_database**

**pacman\_aur**

**pacman\_group**

**samba\_user**



built from the name of the cookbook and the ruby file in the resources/providers directory. these are the aws, bluepill, daemontools, dynect, mysql, pacman and samba cookbooks respectively.