# Rube Goldberg Architecture

# Building a Better Mousetrap for the Cloud

Ezra Zygmuntowicz

# 3 Pillars of a good Cloud Infrastructure

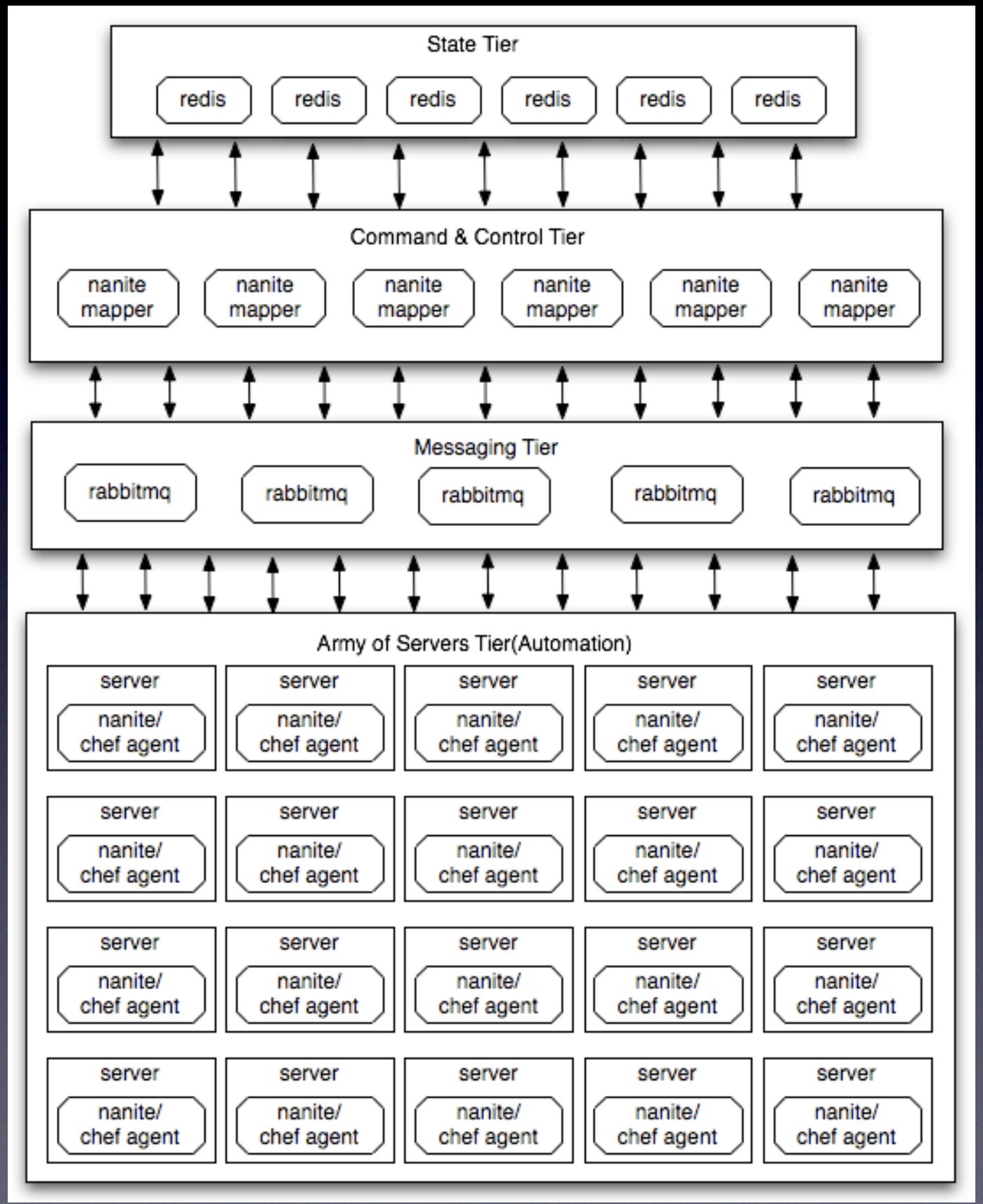
## Automation

## Command & Control

## Scalable State Storage

Automation:
Chef
Command & Control:
Nanite
Scalable State Storage:
Redis

# Go big or go home...



# Automation: Chef

- Idempotent configuration management
- Embed-able and flexible
- Heavy lifting behind Solo/Flex
- No more “every server is a unique snowflake”
- Badass

# Basic Concepts:

Resources

Recipes

Providers

# Resources:

apt_package
bash
cron
csh
directory
execute
file
gem_package
group
http_request
link
mount

```
directory "/db/mysql" do
  owner "mysql"
  group "mysql"
  mode 0755
  recursive true
end
```

```
execute "do-init-mysql" do
  command %Q{
    mysql_install_db
  }
  not_if { File.directory?('/db/mysql/mysql')  }
end
```

package
perl
portage_package
python
remote_directory
remote_file
route
ruby
script
service
template
user

# Recipes:

```
#
# Cookbook Name:: haproxy
# Recipe:: default
#
# Copyright 2009, Engine Yard, Inc.
#
# All rights reserved - Do Not Redistribute
#

package "net-proxy/haproxy" do
  action :install
end

template "/etc/haproxy.cfg" do
  owner 'root'
  group 'root'
  mode 0644
  source "haproxy.cfg.erb"
  variables({
    :backends => node[:members]
  })
end

execute "add-haproxy-to-init" do
  command "rc-update add haproxy default"
  not_if "rc-status | grep haproxy"
end
```

```
gem_package 'foca-integrity-email' do
  action :install
  source "http://gems.github.com"
end

gem_package 'foca-sinatra-ditties' do
  action :install
  source "http://gems.github.com"
end

gem_package 'do_sqlite3' do
  action :install
end

gem_package 'integrity' do
  action :install
  version '0.1.9.0'
end

if_app_needs_recipe("integrity") do |app,data,index|

  execute "install integrity" do
    command "integrity install --passenger /data/#{app}/current"
  end

  template "/data/#{app}/current/config.ru" do
    owner node[:owner_name]
    group node[:owner_name]
    mode 0655
    source "config.ru.erb"
  end

  template "/data/#{app}/current/config.yml" do
    owner node[:owner_name]
    group node[:owner_name]
    mode 0655
    source "config.yml.erb"
    variables({
      :app  => app,
      :domain  => data[:vhosts].first[:name],
    })
  end
end
```

# Providers:

```
class Chef
  class Provider
    class Directory < Chef::Provider::File
      def load_current_resource
        @current_resource = Chef::Resource::Directory.new(@new_resource.name)
        @current_resource.path(@new_resource.path)
        if ::File.exist?(@current_resource.path) && ::File.directory?(@current_resource.path)
          cstats = ::File.stat(@current_resource.path)
          @current_resource.owner(cstats.uid)
          @current_resource.group(cstats.gid)
          @current_resource.mode("%o" % (cstats.mode & 007777))
        end
        @current_resource
      end

      def action_create
        unless ::File.exists?(@new_resource.path)
          Chef::Log.info("Creating #{@new_resource} at #{@new_resource.path}")
          if @new_resource.recursive == true
            ::FileUtils.mkdir_p(@new_resource.path)
          else
            ::Dir.mkdir(@new_resource.path)
          end
          @new_resource.updated = true
        end
        set_owner if @new_resource.owner != nil
        set_group if @new_resource.group != nil
        set_mode if @new_resource.mode != nil
      end

      def action_delete
        if ::File.directory?(@new_resource.path) && ::File.writable?(@new_resource.path)
          if @new_resource.recursive == true
            Chef::Log.info("Deleting #{@new_resource} recursively at #{@new_resource.path}")
            FileUtils.rm_rf(@new_resource.path)
          else
            Chef::Log.info("Deleting #{@new_resource} at #{@new_resource.path}")
            ::Dir.delete(@new_resource.path)
          end
          @new_resource.updated = true
        else
          raise RuntimeError, "Cannot delete #{@new_resource} at #{@new_resource_path}!" if ::File.exists?(@new_resource.path)
        end
      end
    end
  end
end
```

# Converging

- Recipes are loaded in specified order
- Resources are compiled into objects and stored in a ResourceCollection
- ResourceCollection is iterated and the right Provider for each Resource is invoked
- The Providers runs the specified action on each Resource

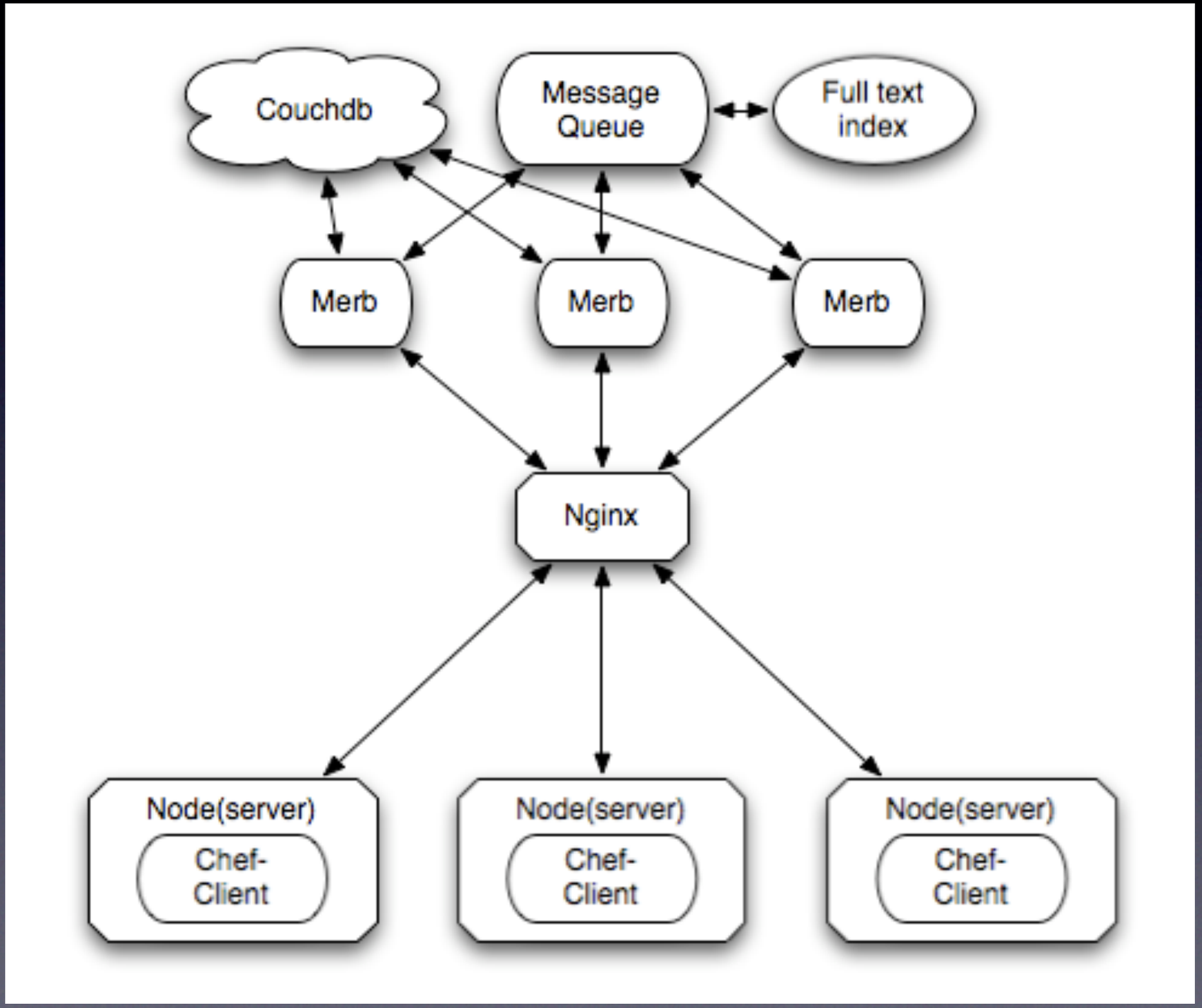
# Chef Solo

- chef-solo -r http://foo.com/recipes.tgz
- Downloads a tarball of recipes and runs them
- Dead simple to get started with

# Chef Server

- chef-client
- Communicates with the chef-server to get recipes and JSON data
- Allows for searching across other nodes attributes
- Uses open-id for node authentication

# Chef Server


Couchdb
Message Queue
Full text index
Merb
Merb
Merb
Nginx
Node(server)
Chef-Client
Node(server)
Chef-Client
Node(server)
Chef-Client

# State Storage: Redis

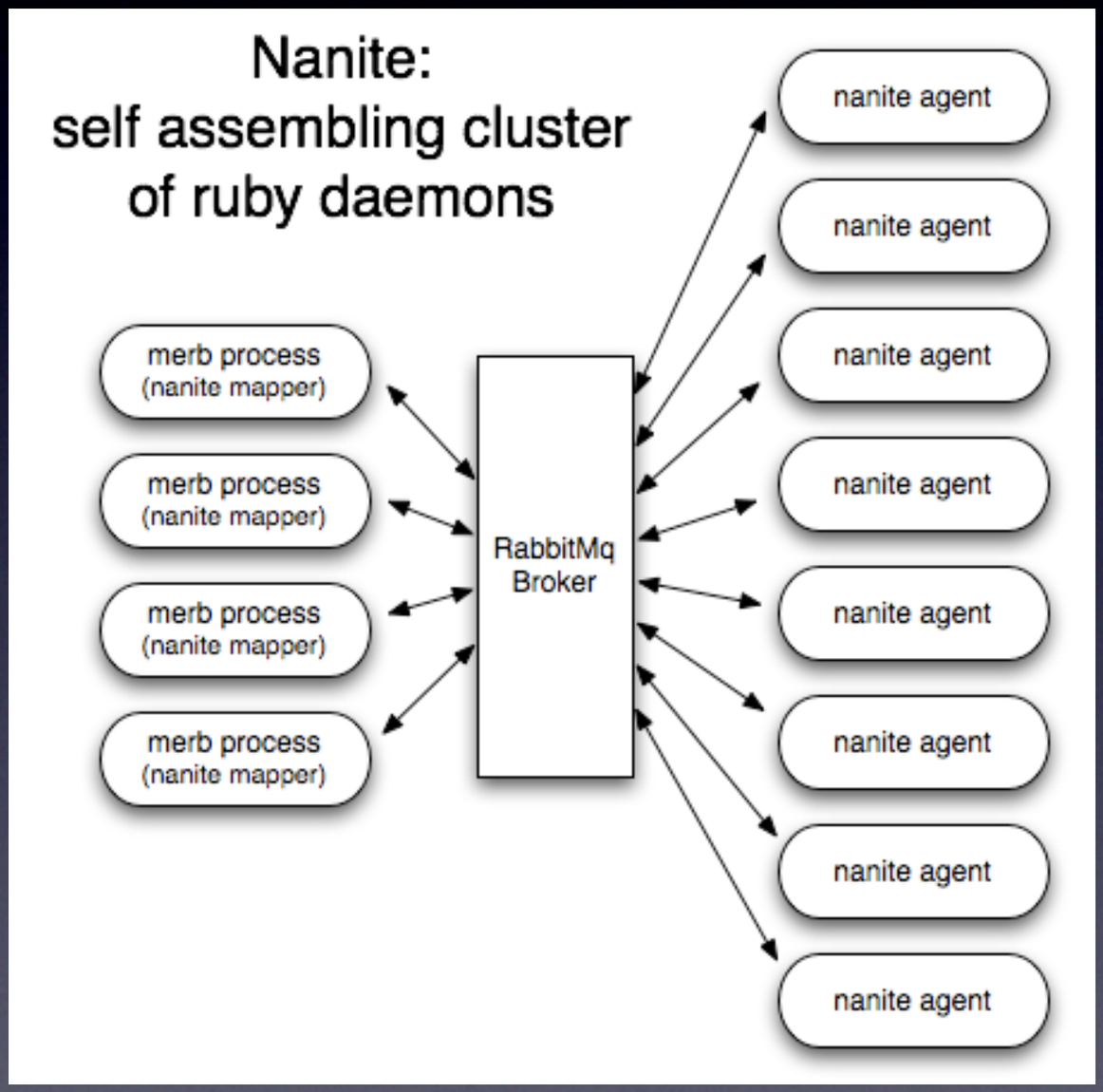

persistent memcached on steroids

# Redis Features

- Asynchronous Persist to disk
- Horizontally scalable
- Values can have types
- STRING's, LIST's and SET's
- Atomic Operations (push, pop, incr, decr, set intersection)

# Let’s see a demo

# Nanite


Nanite:
self assembling cluster
of ruby daemons
merb process
(nanite mapper)
merb process
(nanite mapper)
merb process
(nanite mapper)
merb process
(nanite mapper)
RabbitMq
Broker
nanite agent
nanite agent
nanite agent
nanite agent
nanite agent
nanite agent
nanite agent
nanite agent

# Built around RabbitMQ

- Written in erlang, cluster-able, highly scalable, fast as hell.
- AMQP protocol provides many nice features
- Transient, Persistent and Transactional semantics

# <3 RabbitMQ

# Nanite agents consist of multiple Actors

```
class Feeds < Nanite::Actor
  expose :crawl

  def crawl(urls)
    failed = []
    urls.each do |url|
      failed << url unless process_url(url)
    end
    if failed.empty?
      return :success
    else
      return { :failed => failed }
    end
  end

  def process_url(url)
    # get and process url
  end
end

Nanite::Dispatcher.register(Feeds.new)
```

Nanite agents advertise their services and status

Feeds#crawl

advertises:
/feeds/crawl

Load average is advertised as default status

# Nanite Mappers

Track nanites and their advertised services and status

Can do dispatch based on a number of factors

Run inside your Merb or Rails app or as a separate service

State of all nanites is replicated across all mappers in memory *or* stored in Redis

# Multiple Dispatch Styles

```ruby
urls = %w[http://foo.com http://bar.com http://qux.com]

# send request to the least loaded nanite
Nanite.request('/feeds/crawl', urls, :least_loaded) do |res|
  #do something with result of feed crawling
end

# send request to a random nanite
Nanite.request('/feeds/crawl', urls, :random) do |res|
  #do something with result of feed crawling
end

# send request to *all* nanites
Nanite.request('/feeds/crawl', urls, :all) do |res|
  #do something with result of feed crawling
end
```

# Least loaded dispatch and the fitness function

```ruby
# default based on load average
Nanite.status_proc = lambda{ parse_uptime(`uptime`) }

# You can advertise anything you want that is comparable <=>
 Nanite.status_proc = lambda{ MyApp.calculate_load }
```

Agents ping the mapper exchange every @ping_time seconds. Mappers track the state of all nanites and remove them from mapping if they haven't reported in within a timeout

```
{"fred"=>
  {:status=>0.756666666666667,
   :services=>["/feeds/crawl", "/indexer/index"],
   :timestamp=>"Mon Oct 06 20:39:26 -0700 2008"},
 "barney"=>
  {:status=>0.753333333333333,
   :services=>["/feeds/crawl", "/images/process"],
   :timestamp=>"Mon Oct 06 20:39:21 -0700 2008"}}
```

# Nanite gives us:

- Presence, we know when nanites are ready for requests or not.
- Self assembly, nanites can come and go and can run anywhere with zero configuration in the mappers.
- Dispatch based on load or any fitness function that suits your app
- Easily take advantage of cloud

# Nanite Demo

Please download and run this shell script:

https://cloud.engineyard.com/ete.sh

# Questions?