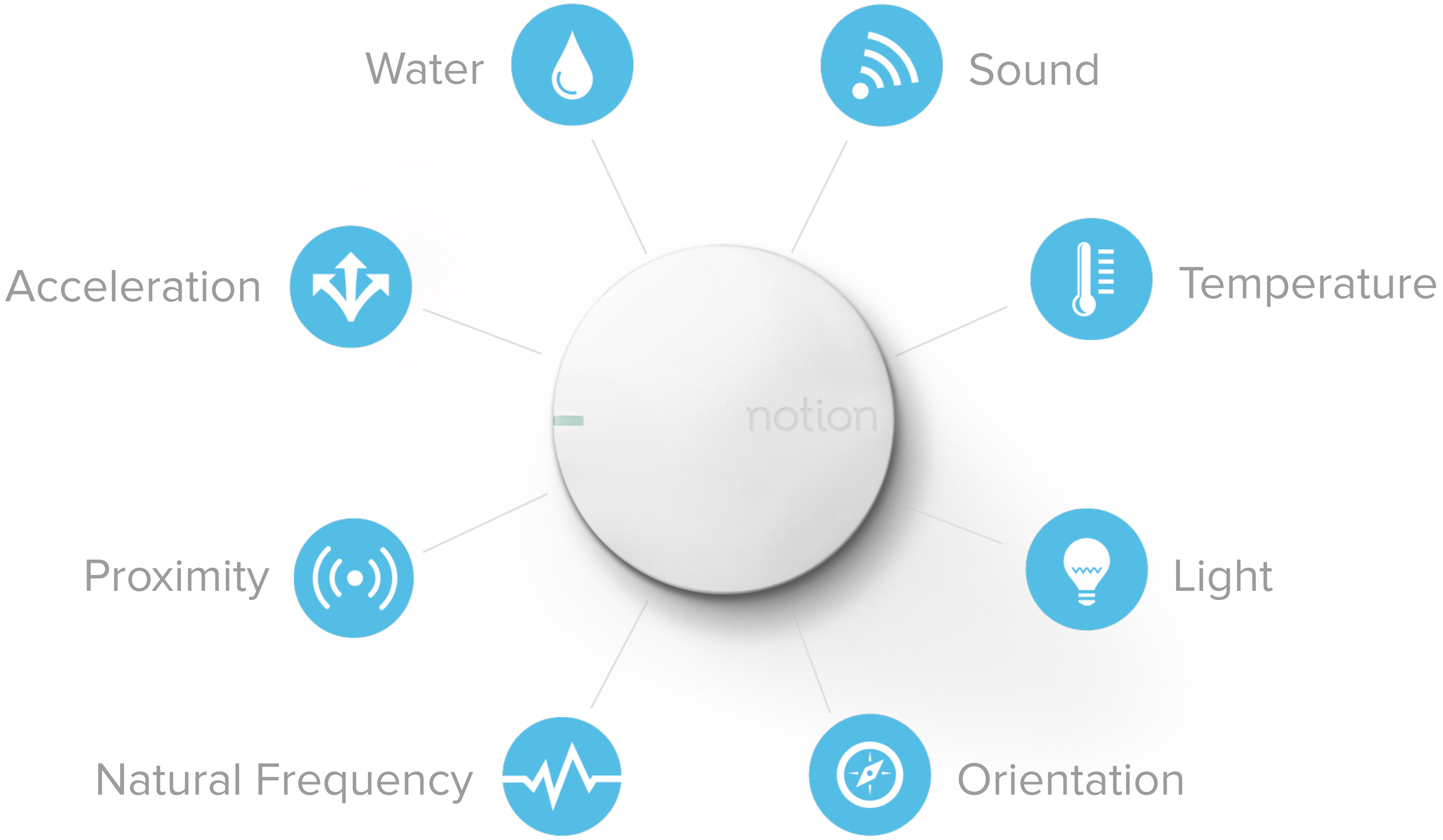
Building an Autoscaler for DigitalOcean

Jordan Stone
Notion

What We'll Talk About and In What Order

- What is Notion?
- Why DigitalOcean is awesome
- Why DigitalOcean is less awesome than other cloud hosting solutions
- Introducing autoscaler
- Deep dive into autoscaler
- Q & A


Water
Sound
Acceleration
Temperature
notion
Proximity
Light
Natural Frequency
Orientation

@notion

Why DigitalOcean is Awesome

- Cheap
- Fast
- Growing (quickly)
- Pre-built images
- Techstars company

Why DigitalOcean is less awesome than other cloud hosting solutions

Feature	DigitalOcean	AWS	Rackspace
File Storage	✗	✓	✓
CDN	✗	✓	✓
Autoscaling	✗	✓	✓
Load Balancing	✗	✓	✓
Security/Access Control	✗	✓	✓

Autoscaler’s Architecture

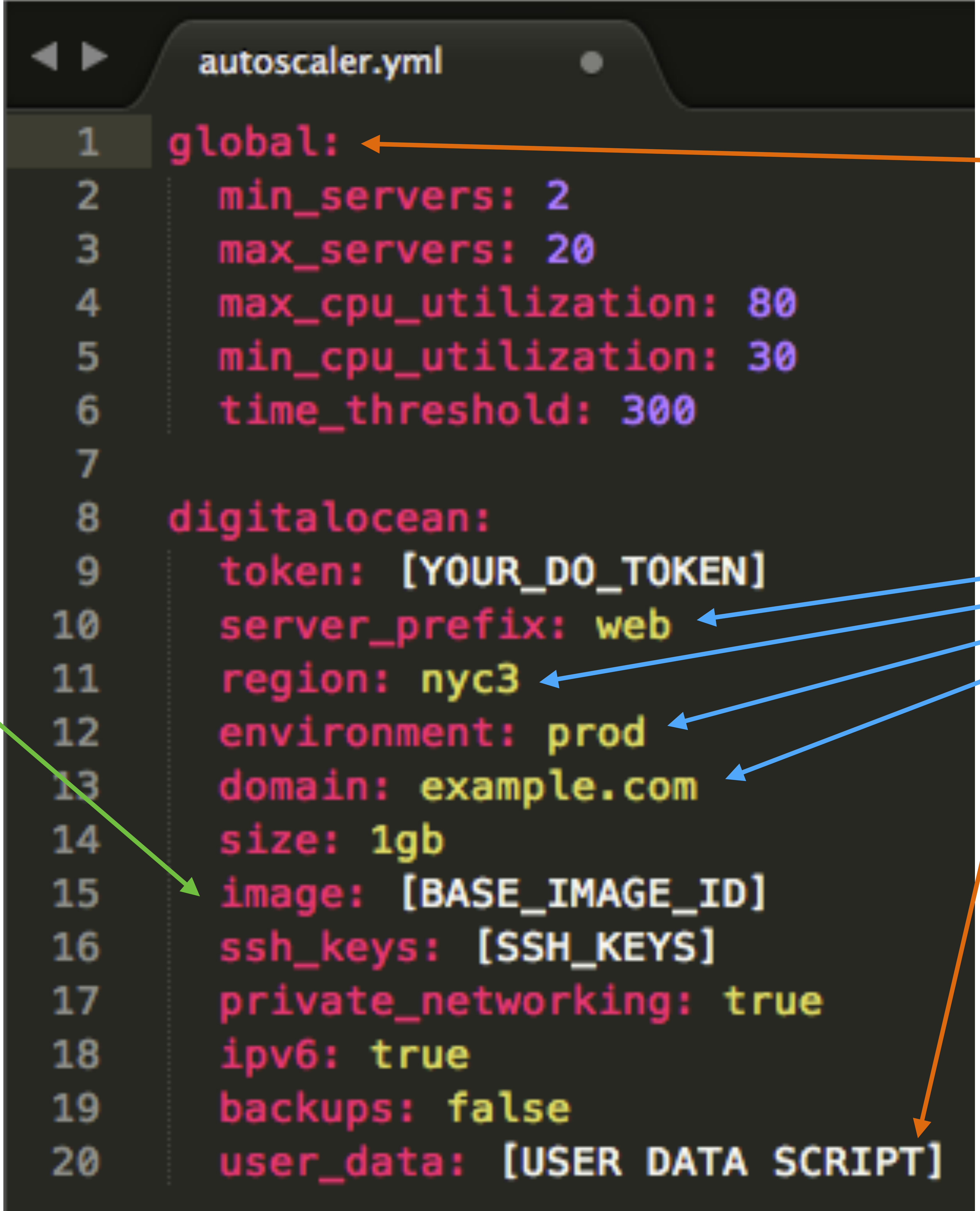
- Consists of two parts
    - One runs on the same server as HAProxy
    - The other runs on a “primary” app server
- Uses RabbitMQ as a messaging queue
- Uses the DigitalOcean API and a standard server naming scheme to add/remove servers from HAProxy

In order to automatically scale, we need to…

- Define which application to scale
- Define a threshold that triggers scaling out
- Define a threshold that triggers scaling in
- Define how long the threshold should be passed before we scale
- Define a minimum and maximum number of servers
- Add/remove a server to our load balancer

Defining which application to scale

- How do others do it?
  - Launch Configurations and Autoscaling Groups
- Is there a DigitalOcean equivalent?
  - Sort of yes, sort of no


autoscaler.yml

```
global:
  min_servers: 2
  max_servers: 20
  max_cpu_utilization: 80
  min_cpu_utilization: 30
  time_threshold: 300

digitalocean:
  token: [YOUR_DO_TOKEN]
  server_prefix: web
  region: nyc3
  environment: prod
  domain: example.com
  size: 1gb
  image: [BASE_IMAGE_ID]
  ssh_keys: [SSH_KEYS]
  private_networking: true
  ipv6: true
  backups: false
  user_data: [USER DATA SCRIPT]
```

Mimicking a Launch Configuration

- DigitalOcean server snapshots
  - User configuration
  - Common packages (fail2ban, linus, etc.)
- Not the easiest to find
- Difficult to maintain

Mimicking an Autoscale Group

- DigitalOcean has no concept of grouping or tagging servers
- We can use a common server naming scheme to group servers
  - server_prefix: Which application to deploy
  - region: Where to deploy it
  - environment, domain: more precise groupings

Mimicking Scaling Policies

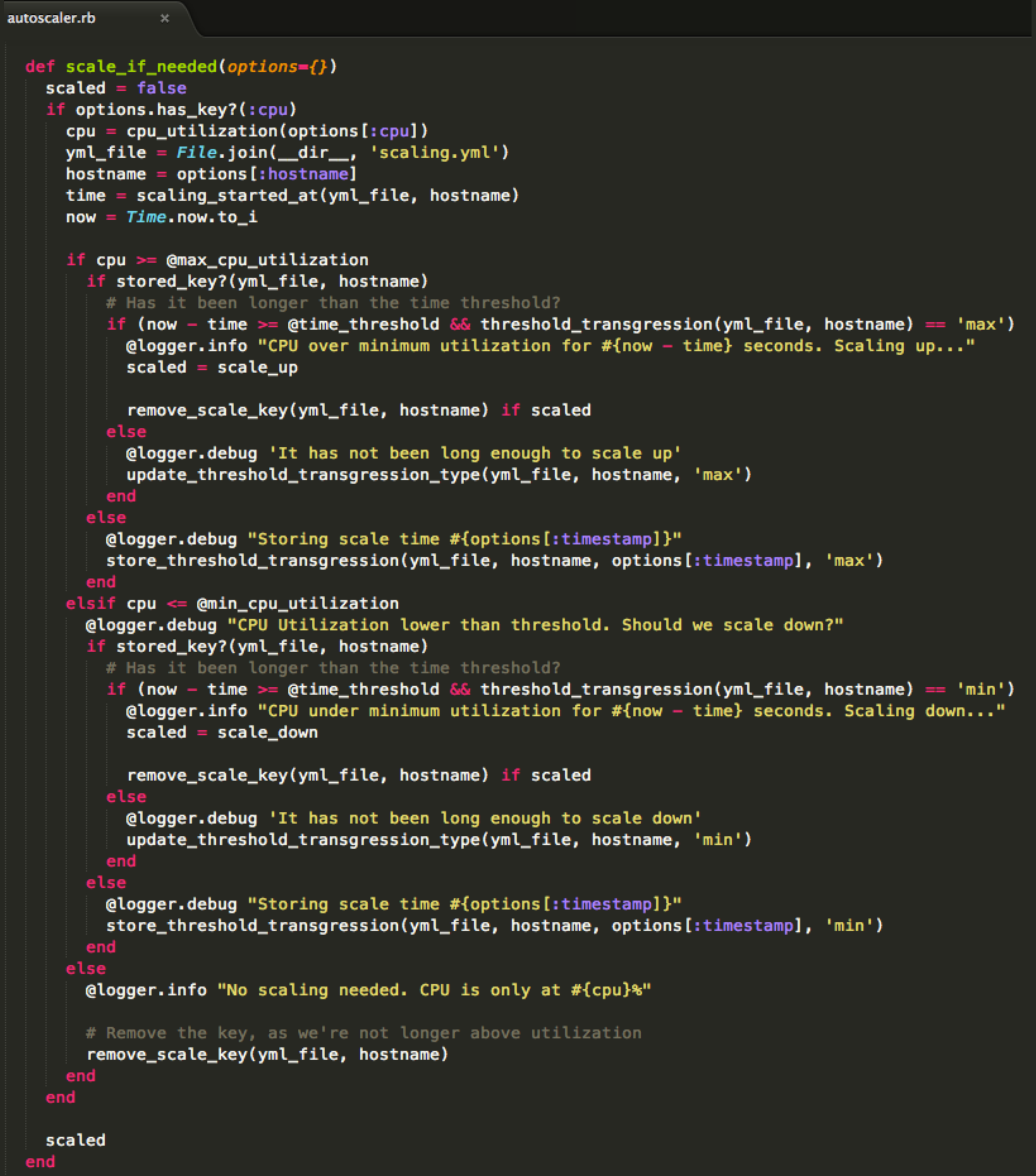
- Minimum/Maximum number of servers
- Define an arbitrary amount of scaling criteria
  - Currently, just CPU utilization
- Define what to do after a new server is provisioned
  - User data

Notifications via Messaging Queue

- We use RabbitMQ, but easy to swap
- Servers report metrics via topics
    - `cpu`, `response_time`, etc.
- Routing key is server hostname
    - Helps determine which scaling group to check

```ruby
require 'bunny'
require 'socket'

class CpuMonitor
  def self.monitor

    conn = Bunny.new(user: 'USER', password: 'PASSWORD', vhost: 'VHOST', host: 'HOST')
    conn.start

    ch = conn.create_channel
    exchange = ch.topic("cpu")

    cpu = `(vmstat|tail -1|awk '{print $15}')`.strip.to_i
    hostname = `hostname`.strip
    exchange.publish(cpu.to_s, routing_key: hostname, timestamp: Time.now.to_i)

    puts " [x] Sent CPU Utilization #{cpu} to routing key #{hostname}"
    conn.close
  end
end

CpuMonitor.monitor
```

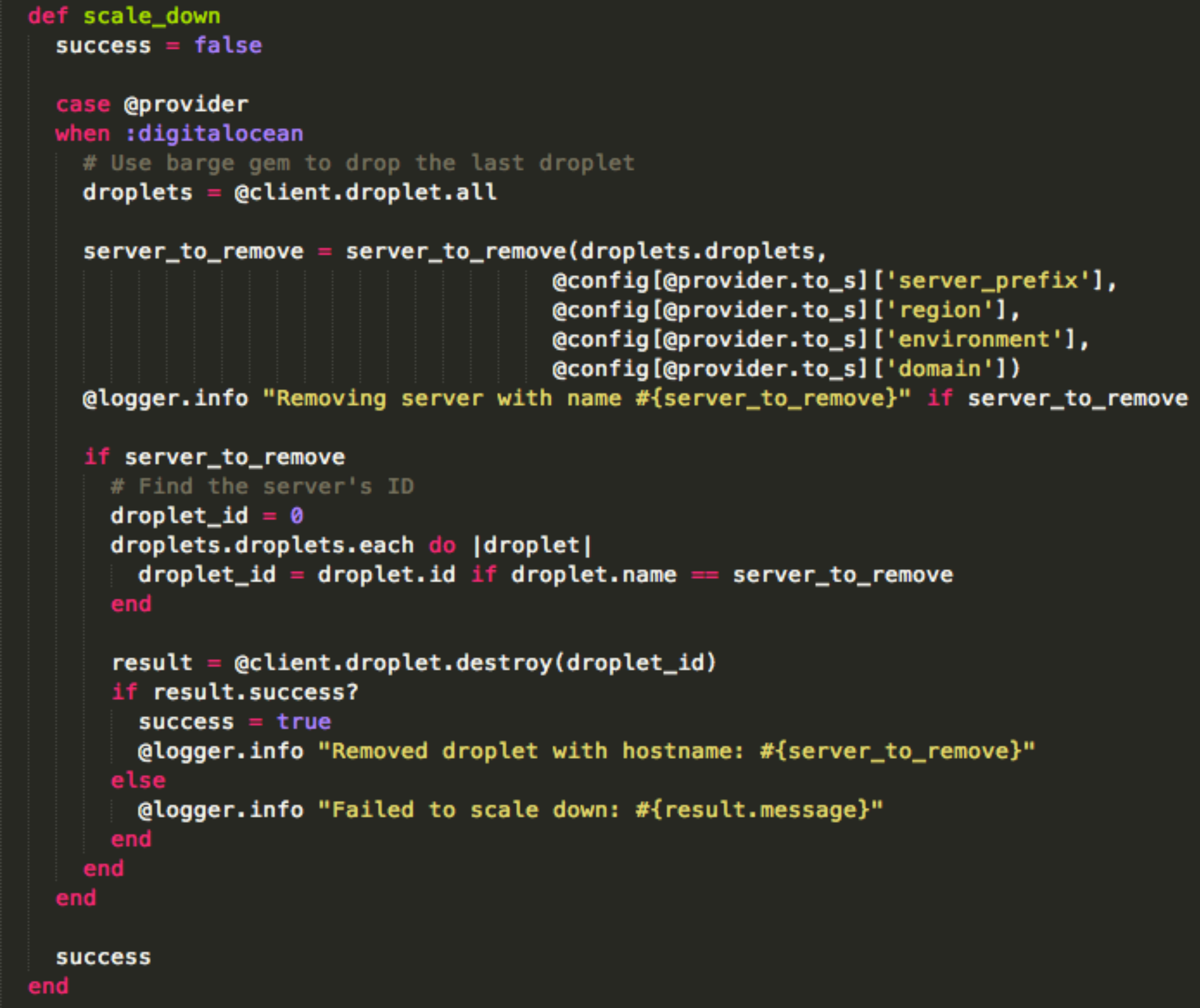
```ruby
class Monitor
  def self.monitor(options={})
    # TODO: Check if the file exists at lb
    # TODO: Raise an error if it doesnt
    # TODO: Actually check the load balancer file.
    autoscaler = Autoscaler.new(options[:host], options[:config])
    conn = ::Bunny.new(host: options[:rhost], user: options[:user], password: options[:password], vhost: options[:vhost])
    conn.start

    channel = conn.create_channel

    exchange = channel.topic('cpu')

    puts " [*] Waiting for messages in #{exchange}. To exit press CTRL+C"

    channel.queue('', exclusive: true).bind(exchange, routing_key: "SOME_REGEX").subscribe(block: true) do |delivery_info, metadata, payload|
      puts "Received message #{payload}, routing key is #{delivery_info.routing_key}, exchange is #{delivery_info.exchange}, sent at time #{metadata.timestamp.to_i}"
      scaled = autoscaler.scale_if_needed(cpu: payload.to_i, hostname: delivery_info.routing_key, timestamp: metadata.timestamp.to_i)

      if scaled
        # Wait 30 seconds for our new instance to be spun up/down
        sleep 30

        # Reload the template HAProxy configuration file
        fn = File.dirname(File.expand_path(__FILE__)) + "/templates/#{options[:lb_config_template]}"
        puts fn
        bn = binding
        bn.local_variable_set(:servers, autoscaler.private_server_ips)
        hap_config = ERB.new(File.read(fn), nil, '-').result(bn)
        # Write the file to the config location
        res = File.open(lb_config, 'w') do |f|
          f << hap_config
        end

        puts "Config is at #{options[:lb_config]}"
        # Restart the haproxy service
        `sudo service haproxy restart`
      end
    end
  end
end
```



autoscaler.rb ×

```ruby
def scale_if_needed(options={})
  scaled = false
  if options.has_key?(:cpu)
    cpu = cpu_utilization(options[:cpu])
    yml_file = File.join(__dir__, 'scaling.yml')
    hostname = options[:hostname]
    time = scaling_started_at(yml_file, hostname)
    now = Time.now.to_i

    if cpu >= @max_cpu_utilization
      if stored_key?(yml_file, hostname)
        # Has it been longer than the time threshold?
        if (now - time >= @time_threshold && threshold_transgression(yml_file, hostname) == 'max')
          @logger.info "CPU over minimum utilization for #{now - time} seconds. Scaling up..."
          scaled = scale_up

          remove_scale_key(yml_file, hostname) if scaled
        else
          @logger.debug 'It has not been long enough to scale up'
          update_threshold_transgression_type(yml_file, hostname, 'max')
        end
      else
        @logger.debug "Storing scale time #{options[:timestamp]}"
        store_threshold_transgression(yml_file, hostname, options[:timestamp], 'max')
      end
    elsif cpu <= @min_cpu_utilization
      @logger.debug "CPU Utilization lower than threshold. Should we scale down?"
      if stored_key?(yml_file, hostname)
        # Has it been longer than the time threshold?
        if (now - time >= @time_threshold && threshold_transgression(yml_file, hostname) == 'min')
          @logger.info "CPU under minimum utilization for #{now - time} seconds. Scaling down..."
          scaled = scale_down

          remove_scale_key(yml_file, hostname) if scaled
        else
          @logger.debug 'It has not been long enough to scale down'
          update_threshold_transgression_type(yml_file, hostname, 'min')
        end
      else
        @logger.debug "Storing scale time #{options[:timestamp]}"
        store_threshold_transgression(yml_file, hostname, options[:timestamp], 'min')
      end
    else
      @logger.info "No scaling needed. CPU is only at #{cpu}%"

      # Remove the key, as we're not longer above utilization
      remove_scale_key(yml_file, hostname)
    end
  end

  scaled
end
```

```ruby
def scale_up
  success = false

  case @provider
  when :digitalocean
    # Use barge gem to create a new droplet
    options = @config[@provider.to_s].reject do |k, v|
      k == 'token' || k == 'server_prefix' || k == 'environment' || k == 'domain'
    end

    options['name'] = build_hostname(@config[@provider.to_s]['server_prefix'],
                                     @config[@provider.to_s]['region'],
                                     @config[@provider.to_s]['environment'],
                                     @config[@provider.to_s]['domain'])

    @logger.debug "Built options hash #{options.inspect}"
    result = @client.droplet.create(options)
    if result.success?
      success = true
      @logger.info "Built new droplet with hostname #{options['name']}"
    else
      @logger.info "Failed to scale up: #{result.message}"
    end
  end

  success
end
```



```ruby
def scale_down
  success = false

  case @provider
  when :digitalocean
    # Use barge gem to drop the last droplet
    droplets = @client.droplet.all

    server_to_remove = server_to_remove(droplets.droplets,
                                  @config[@provider.to_s]['server_prefix'],
                                  @config[@provider.to_s]['region'],
                                  @config[@provider.to_s]['environment'],
                                  @config[@provider.to_s]['domain'])
    @logger.info "Removing server with name #{server_to_remove}" if server_to_remove

    if server_to_remove
      # Find the server's ID
      droplet_id = 0
      droplets.droplets.each do |droplet|
        droplet_id = droplet.id if droplet.name == server_to_remove
      end

      result = @client.droplet.destroy(droplet_id)
      if result.success?
        success = true
        @logger.info "Removed droplet with hostname: #{server_to_remove}"
      else
        @logger.info "Failed to scale down: #{result.message}"
      end
    end
  end

  success
end
```

Start autoscaler

- Install required gems, RabbitMQ Server
  - See README for instructions
- Configure RabbitMQ server
  - See README for instructions
- Copy `central/` to HAProxy server
- Copy `send_cpu_util.rb` to primary server
- Copy `autoscaler.yml.example` to `autoscaler.yml`
- Run

```
/path/to/ruby /path/to/monitor_balancer.rb -h
digitalocean -l haproxy -h 1.1.1.1 -u user -p password -o
vhost
```

@notion

To Do

- Further break apart and containerize services
- Support server monitoring API’s
- Support > 1 app server
- Support multiple cloud hosting providers

Thanks!

Jordan Stone
Chief Software Architect
jordan@getnotion.com
@cheddz

https://github.com/LoopLabsInc/autoscalr

@notion