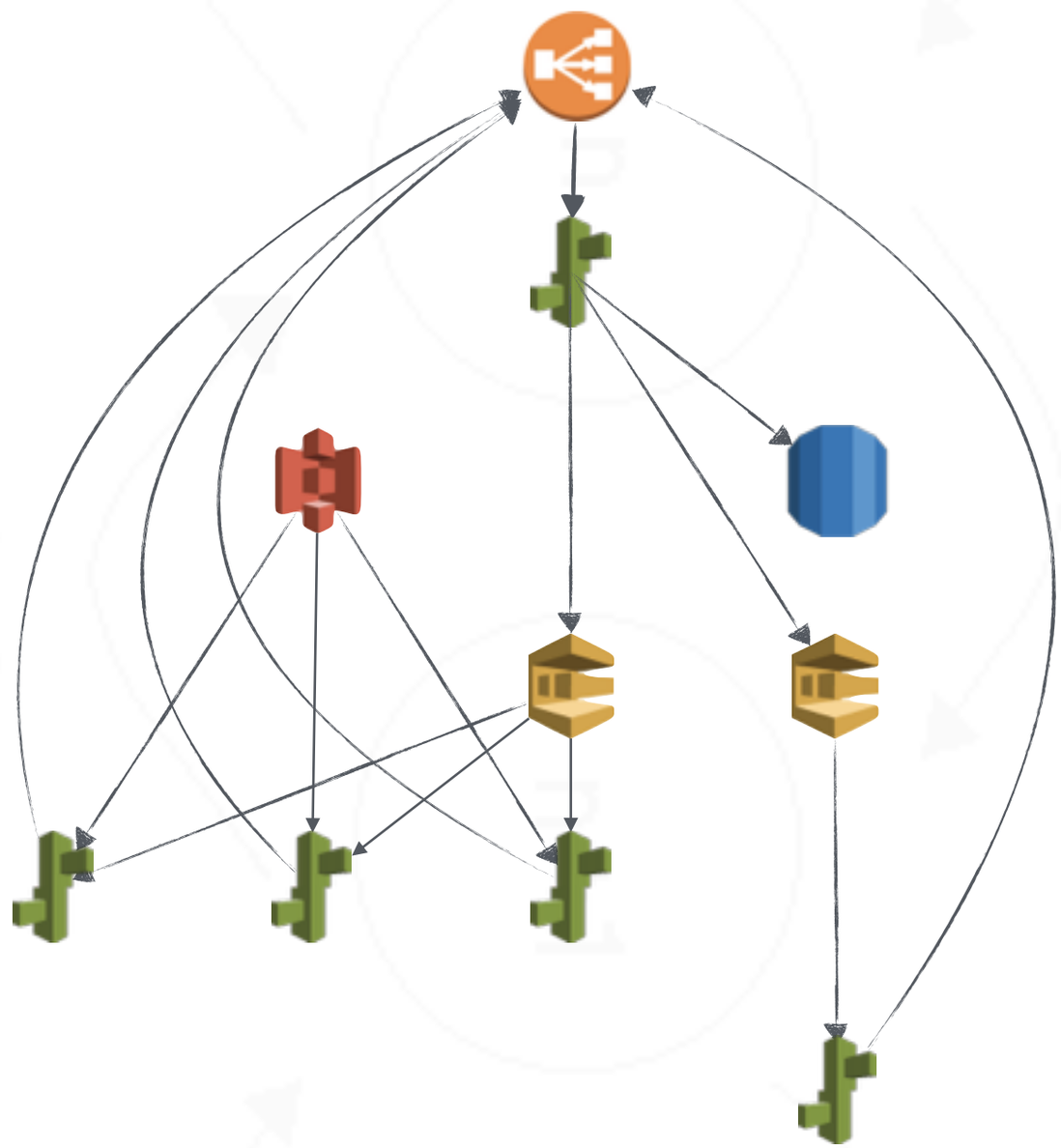
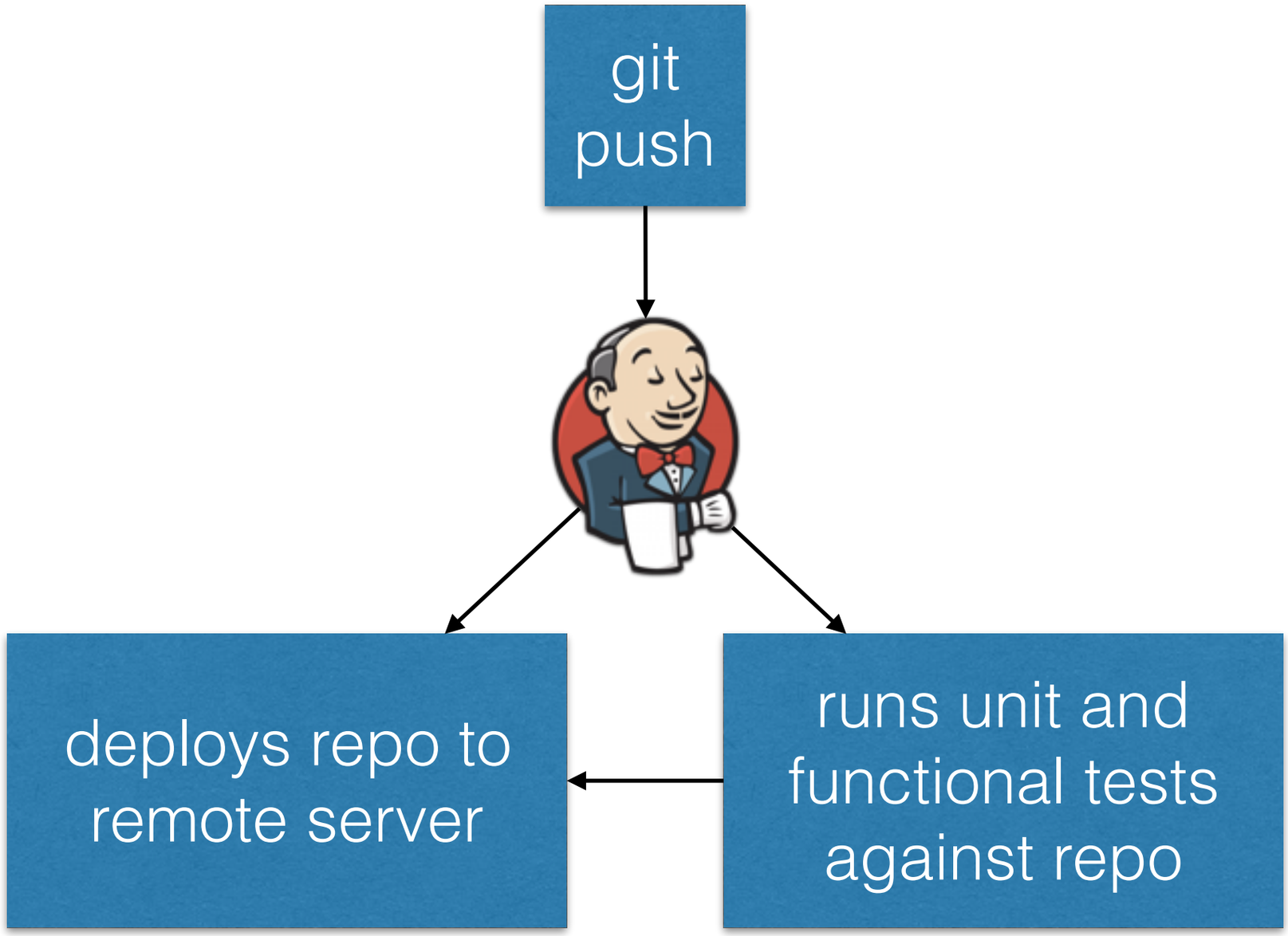
the Automated Testing Platform

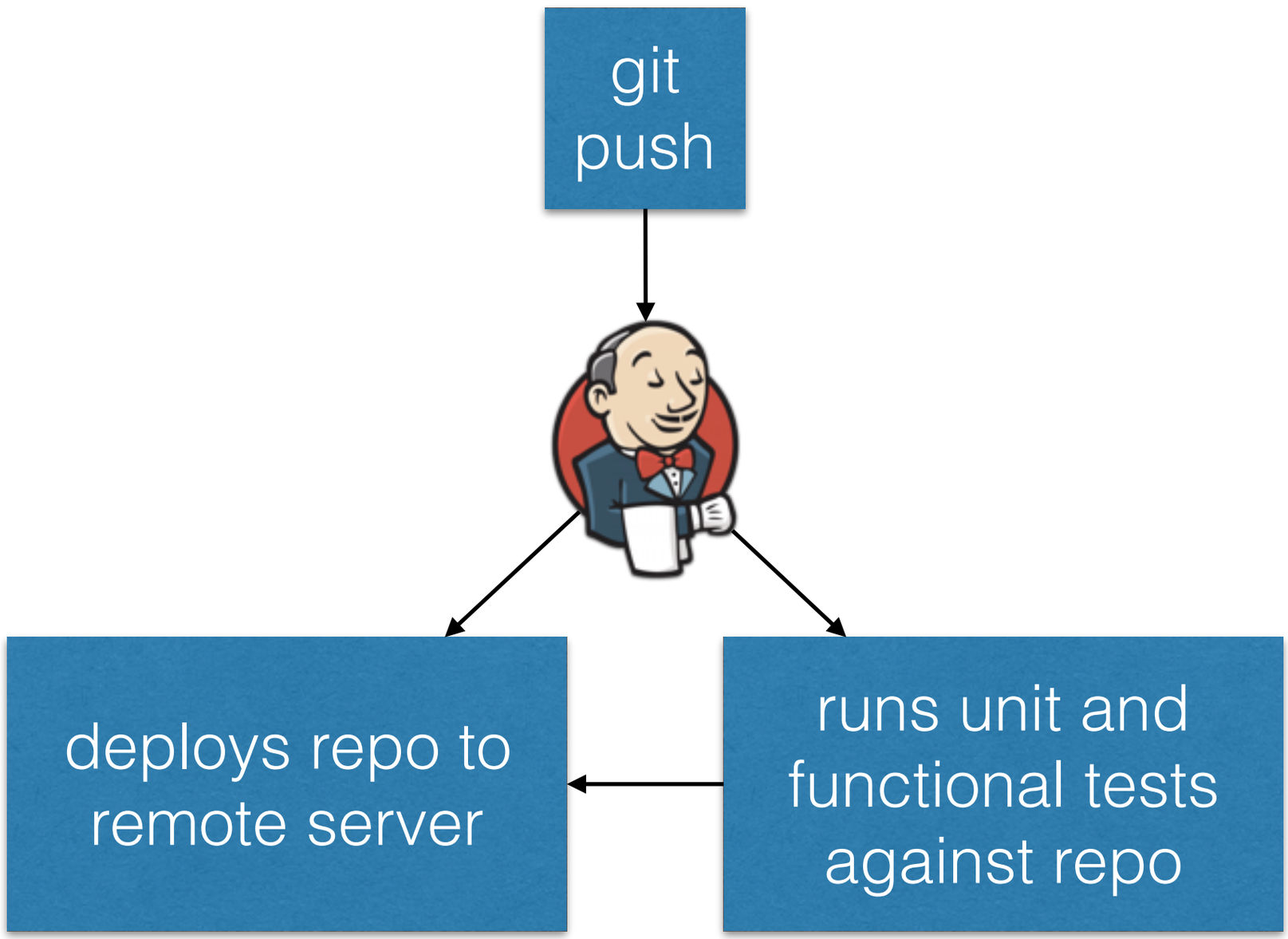


the Continuous Integration Pipeline

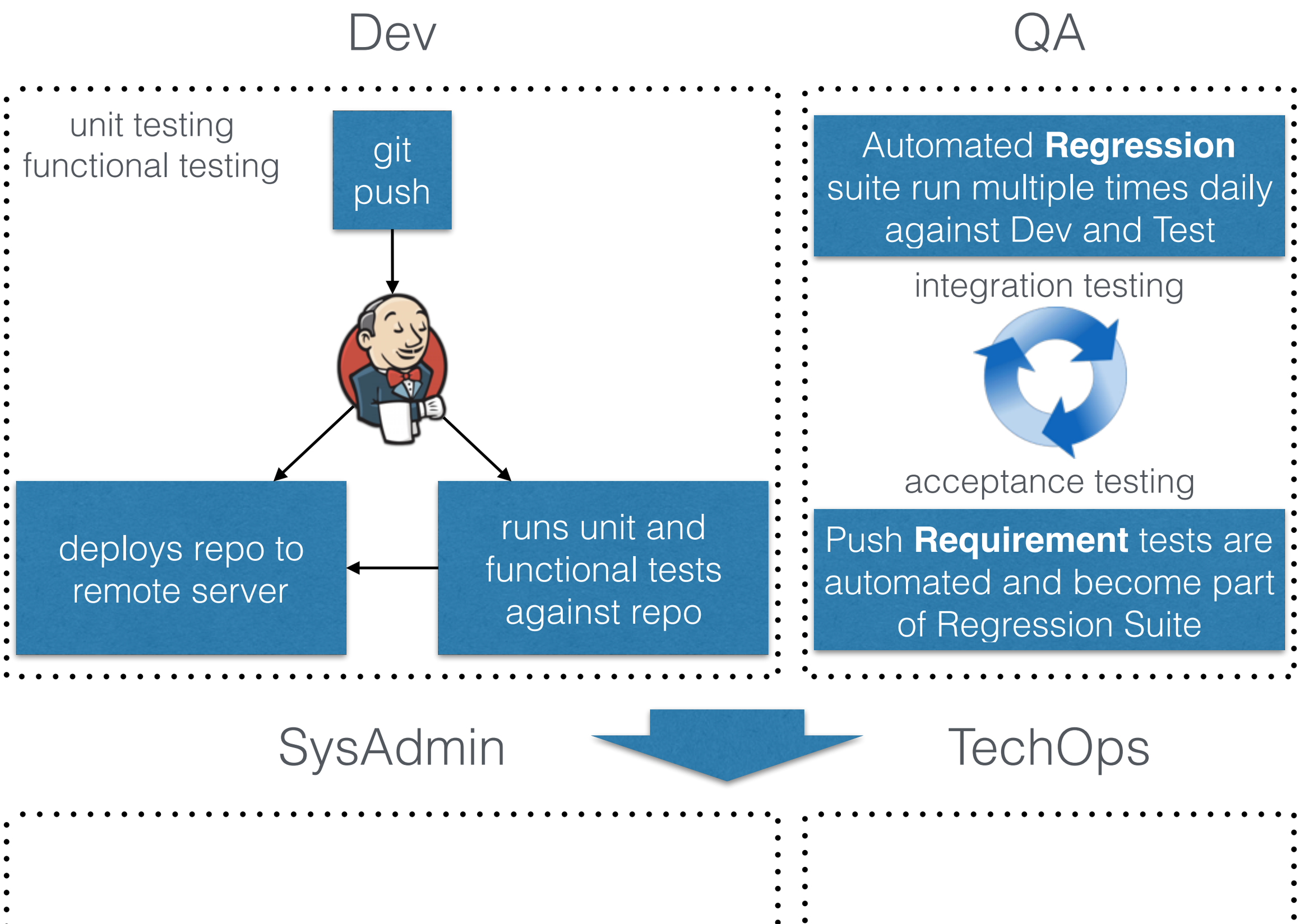


the Continuous Integration Pipeline

What about integration testing?
What about acceptance testing?



the Continuous Integration Pipeline



API Testing

- Do APIs return the correct response in the expected format for a broad range of feasible requests?
- Do APIs react properly to edge cases such as failures and unexpected extreme inputs?
- Do APIs deliver responses in an acceptable amount of time?
- Do APIs respond securely to potential security attacks?

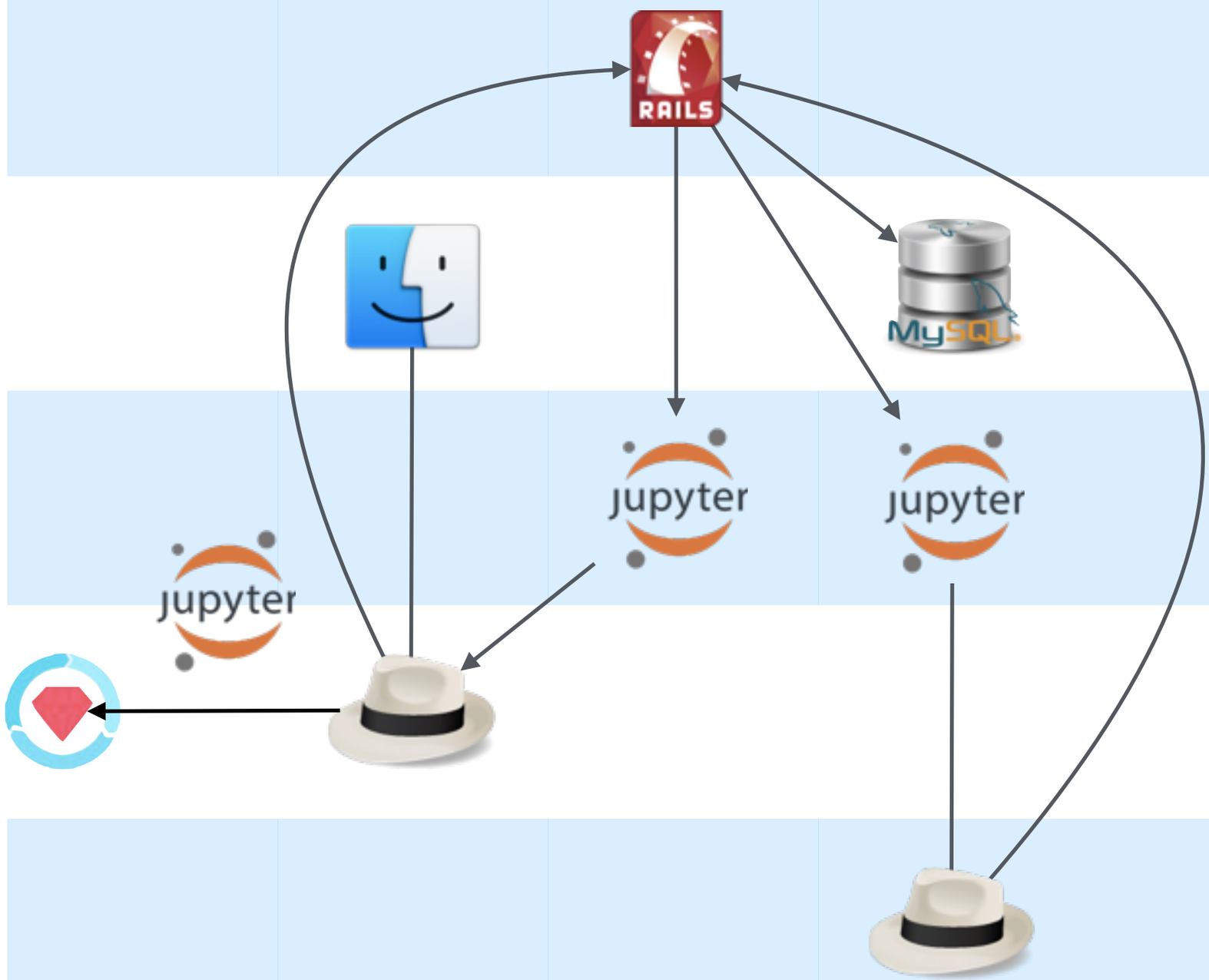
amazon web services

- Elastic Load Balancer
  - links front-end application to a DNS
- Elastic Beanstalk
  - used for building three types of application comprising the system
  - instantiates EC2, EBS, and other necessary services
- Simple Queue Service
  - used for passing messages between Elastic Beanstalk instances
- Relational Database Service
  - used by front-end to store admins, spec file information, and spec file executions
- Simple Storage Service
  - used to store media used in tests

open source software

- Ruby
  - high-level object-oriented language
  - lends itself to the creation of domain specific languages
- Rails
  - used to build our front-end
  - built-in support for views with Twitter Bootstrap support
- Sinatra
  - bare bones routing support, perfect for API only machines
- Rspec
  - Test Harness
  - Clear expectation vocabulary
- Jupyter
  - Sandbox for developing Spec Files

the Local Environment



web tier

RoR application serving front-end interface, connections to database and public api behind authentication. Schedules jobs for execution.

data tier

S3 stores assets used in tests.
RDS serves as database to web tier RoR application.

queue tier

Two instances of SQS sending requests to two types of worker tier applications.

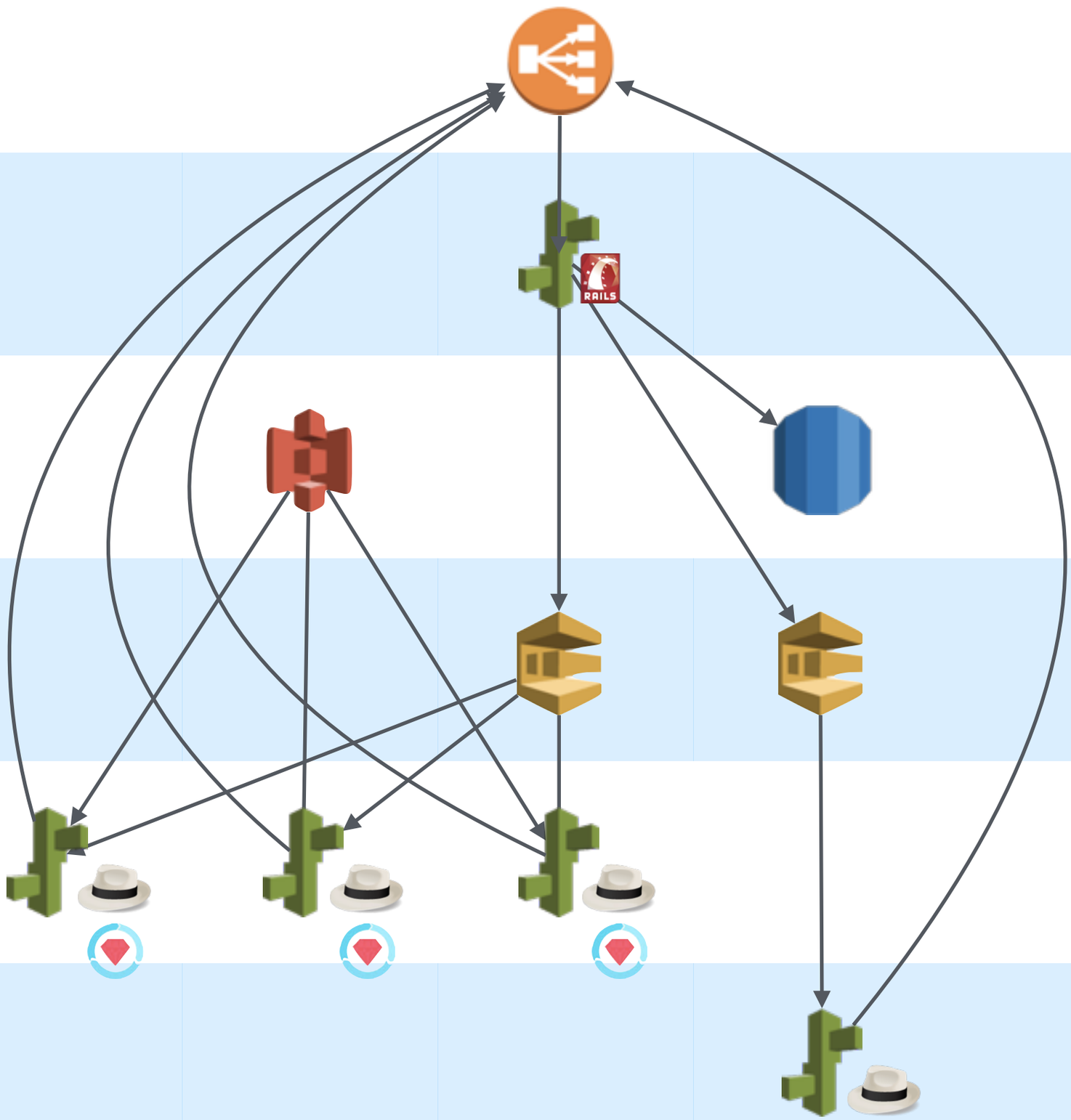
worker tier

Sinatra application executing Spec files against corporate APIs using RSpec.

heartbeat tier

Sinatra application queries front-end for jobs ready to be executed.

the Blueprint



load balancer

AWS service linking external domain to auto-scaling group containing web application.

web tier

RoR application serving front-end interface, connections to database and public api behind authentication. Schedules jobs for execution.

data tier

S3 stores assets used in tests.
RDS serves as database to web tier RoR application.

queue tier

Two instances of SQS sending requests to two types of worker tier applications.

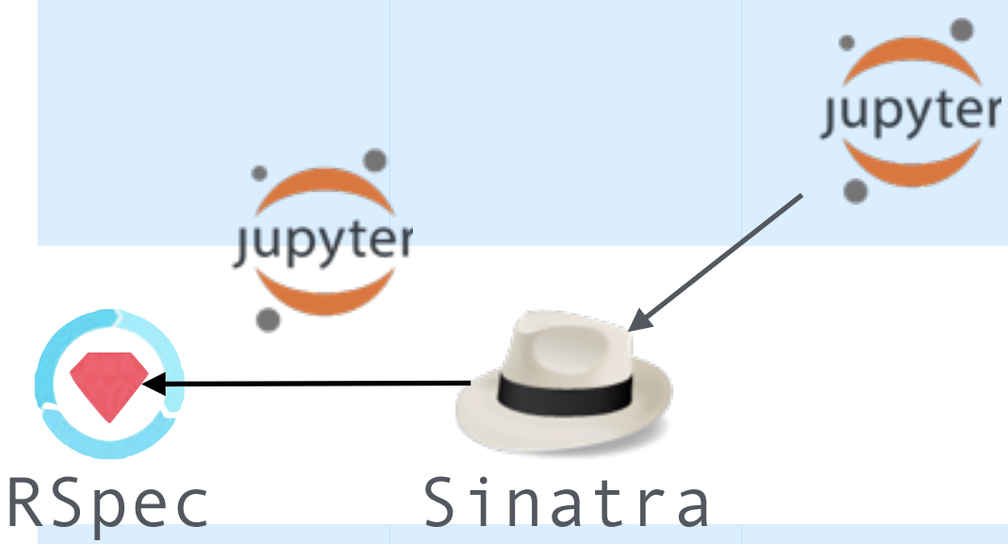
worker tier

Sinatra application executing Spec files against corporate APIs using RSpec.

heartbeat tier

Sinatra application queries front-end for jobs ready to be executed.

the Spec File



worker tier

Sinatra application executing Spec files against corporate APIs using RSpec.

the Spec File

- RSpec
  - DSL for spec files
    - (it … do, context … do)
- Jupyter Notebook
  - development environment for spec files
- Jupyter Notebook
  - API call simulates SQS
- Sinatra
  - web server for worker tier
  - executes rspec at command line
- RSpec
  - test harness executed at command line

the Spec File

- RSpec
  - platform_api_spec.rb

```ruby
describe "Platform API" do
  before :all do
    ENVIRONMENT = 'ec2_test'
    @session = PlatformAPI::Session.new ENVIRONMENT
  end

  context 'Session' do
    it 'can list campaigns' do
    it 'can get a single campaign' do
  end

  context 'Campaigns' do
    before :all do

    it 'can create a campaign' do

    it 'can modify an existing campaign'
    it 'can archive an existing campaign'
  end

  context 'Topics' do
    before :all do

    it 'can create a topic for a campaign' do
  end

  context 'Facebook Routing' do
    before :all do

    it 'can create a facebook routing for a campaign' do
  end

end
```

the Spec File

- Jupyter
  - development environment for spec

```
In [1]: Dir.chdir "/Users/joshuacook/src/telescope/automaton/worker"
        require './spec/spec_helper.rb'
        require 'rspec/expectations'
        include RSpec::Matchers

Out[1]: Object
```

```
In [6]: # @campaign_id    = CONFIG['live_poll_widget']['campaign_id']
        # @widget_id      = CONFIG['live_poll_widget']['widget_id']
        # @topic_id       = CONFIG['live_poll_widget']['topic_id']

        # @widget_id      = "f1d020226069a668"
        # @topic_id       = "1003532"
        @keyword_one    = "one"
        @keyword_two    = "two"
        @keyword_three  = "three"
        @base_url       = "http://on-air-api-test.telescope.tv"
        @poll_id        = 8

Out[6]: 4
```

```
module LivePollsApi
  def LivePollsApi.create_poll base_url, widget_id, topic_id
    uri = URI "#{base_url}/polls"
    live_poll_api = {"widget_id"=>widget_id, "topic_id"=>topic_id}
    request = Net::HTTP::Post.new uri, {'Content-Type' =>'application/json'}
    request.body = live_poll_api.to_json
    response = Net::HTTP.new(uri.host, uri.port).start do |http|
      http.request request
    end
    return response
  end
```

it 'can create a poll when it does not exist"

```
In [5]: response = LivePollsApi::create_poll @base_url, @widget_id, @topic_id
        response_hash = JSON.parse response.body
        response.body

Out[5]: "{\"poll_id\":4,\"widget_id\":\"f1d020226069a668\",\"topic_id\":\"1003532\",\"last_updated\":1459547457253,\"created
        \":1459547457253,\"expire\":1459806657253,\"keywords\":[],\"active\":true}"
```

the Spec File

- Jupyter
  - simulates SQS

Query worker

In [1]:
```
uri = URI "http://localhost:9292/worker"
params_api = {"id"=>46, "spec_file"=>"platform_api_spec.rb"}
request = Net::HTTP::Post.new uri, initheader = {'Content-Type' =>'application/json'}
request.body = params_api.to_json
response = Net::HTTP.new(uri.host, uri.port).start do |http|
  http.request request
end
```

Out[1]:
```
#<Net::HTTPOK 200 OK readbody=true>
```

the Spec File

- Sinatra
  - web server for worker tier

```
post '/worker' do
  request_payload       = Worker.receive_json request
  spec_file, job_id     = Worker.parse_payload request_payload
  spec_json             = Worker.run_spec spec_file
  spec_json_with_job_id = Worker.add_job_id_to_json spec_json, job_id
  response              = Worker.post_results spec_json_with_job_id, job_id
  status 200
  body response
end
```

```
def Worker.run_spec spec_file
  puts spec_file
  unless spec_file
    return 'spec_file is nil'
  else
    spec_json = `rspec spec/lib/#{spec_file}`
    if spec_json == ""
      return 'invalid spec file'
    else
      return spec_json
    end
  end
end
```

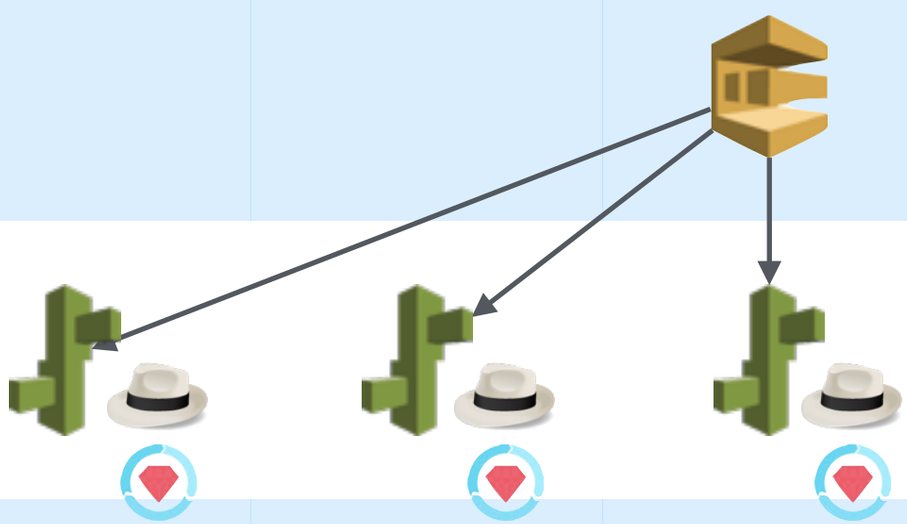
shell call

the Spec File

- RSpec
  - test harness

```
=> ~/src/telescope/ c automaton/worker/
AUTO-86_write_specfiles_for_livepolls_api_in_support_of_BACK-1065 => ~/src/telescope/automaton/worker/ rspec -f d spec/lib/platform_api_spec.rb
Run options: include {:focus=>true}

All examples were filtered out; ignoring {:focus=>true}

Platform API
  Session
    can list campaigns
    can get a single campaign
  Campaigns
    can create a campaign
    can modify an existing campaign (PENDING: Not yet implemented)
    can archive an existing campaign (PENDING: Not yet implemented)
  Topics
    can create a topic for a campaign
  Facebook Routing

    can create a facebook routing for a campaign

Pending: (Failures listed here are expected and do not affect your suite's status)

  1) Platform API Campaigns can modify an existing campaign
     # Not yet implemented
     # ./spec/lib/platform_api_spec.rb:45

  2) Platform API Campaigns can archive an existing campaign
     # Not yet implemented
     # ./spec/lib/platform_api_spec.rb:46


Finished in 35 seconds (files took 8.26 seconds to load)
7 examples, 0 failures, 2 pending
```

the Worker



queue tier

Two instances of SQS sending requests to two types of worker tier applications.

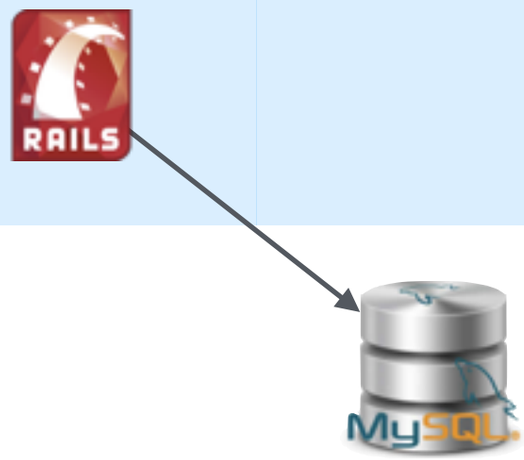
worker tier

Sinatra application executing Spec files against corporate APIs using RSpec.

the Worker

- SQS Queue
  - Has messages consisting of simple JSON e.g.
    - `{"job_id":1, "spec_file":"platform_api_spec.rb"}`
- Elastic Beanstalk Daemon
  - polls the SQS server
  - SQS server sends requested message to EB Worker as an API call to "/worker"
- Sinatra
  - Receives API Call
  - runs spec file and returns a 200 status
- SQS Queue removes message

the Rails App



web tier

RoR application serving front-end interface, connections to database and public api behind authentication. Schedules jobs for execution.

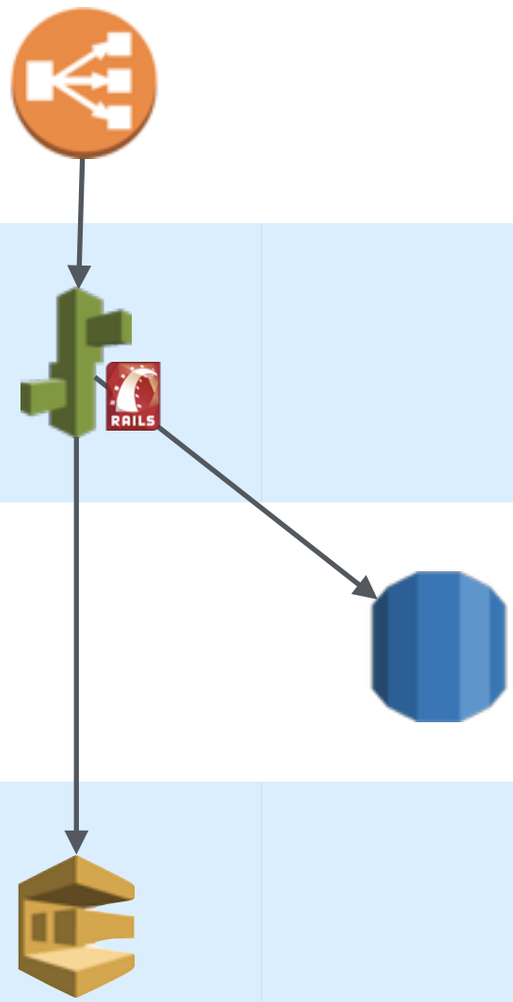
data tier

S3 stores assets used in tests.
RDS serves as database to web tier RoR application.

the Rails App

- Scaffolds (Models, Views, Controllers)
  - Admins
  - Spec Files
  - Spec File Executions
  - Behind email/password authentication
- API
  - "/api/heartbeat"
  - "/api/spec_file"
  - "/api/spec_file_executions"
  - Behind authentication token

the Front-end



load balancer

AWS service linking external domain to auto-scaling group containing web application.

web tier

RoR application serving front-end interface, connections to database and public api behind authentication. Schedules jobs for execution.

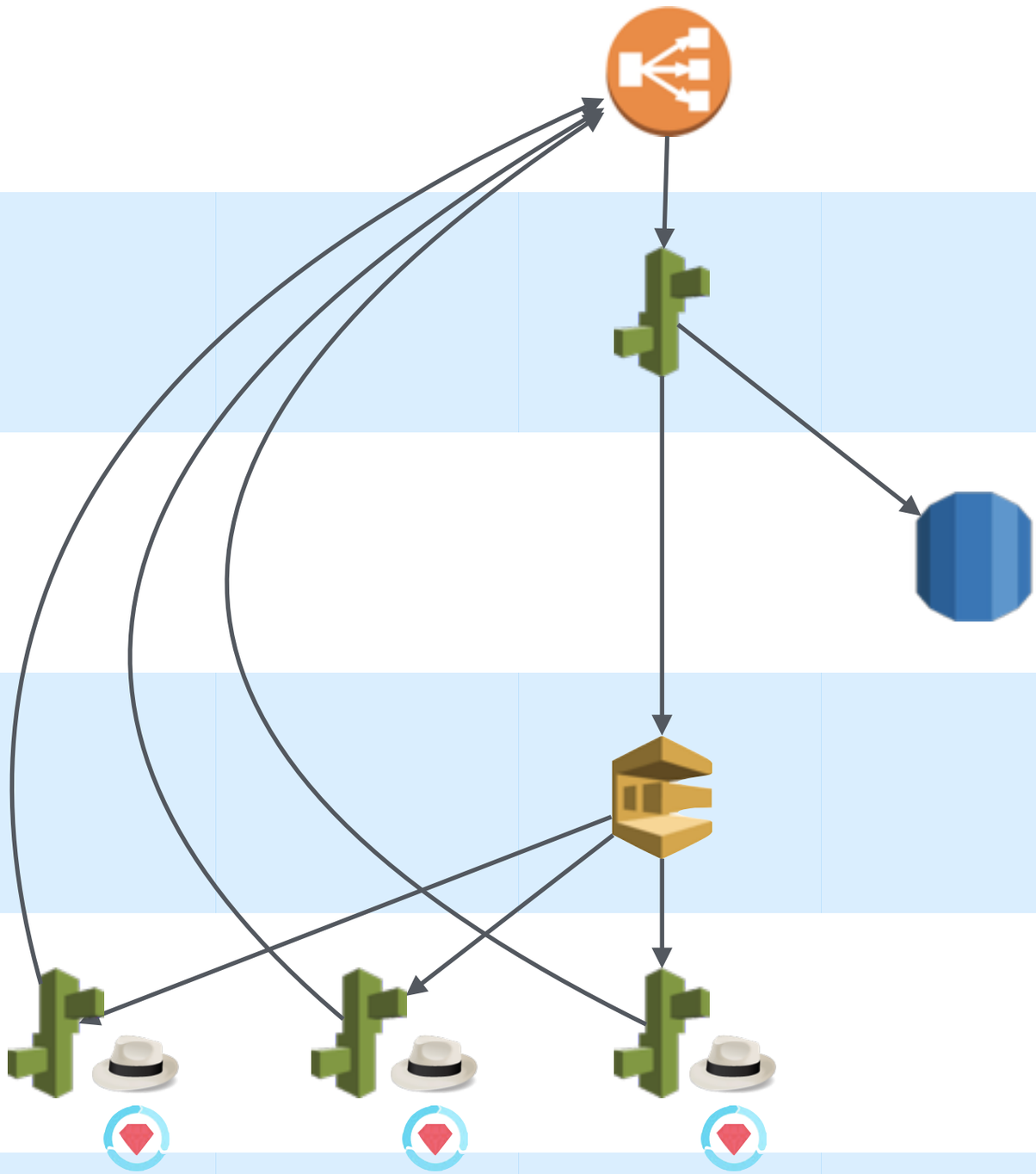
data tier

S3 stores assets used in tests.
RDS serves as database to web tier RoR application.

queue tier

Two instances of SQS sending requests to two types of worker tier applications.

the Job Runner



load balancer

AWS service linking external domain to auto-scaling group containing web application.

web tier

RoR application serving front-end interface, connections to database and public api behind authentication. Schedules jobs for execution.

data tier

S3 stores assets used in tests.
RDS serves as database to web tier RoR application.

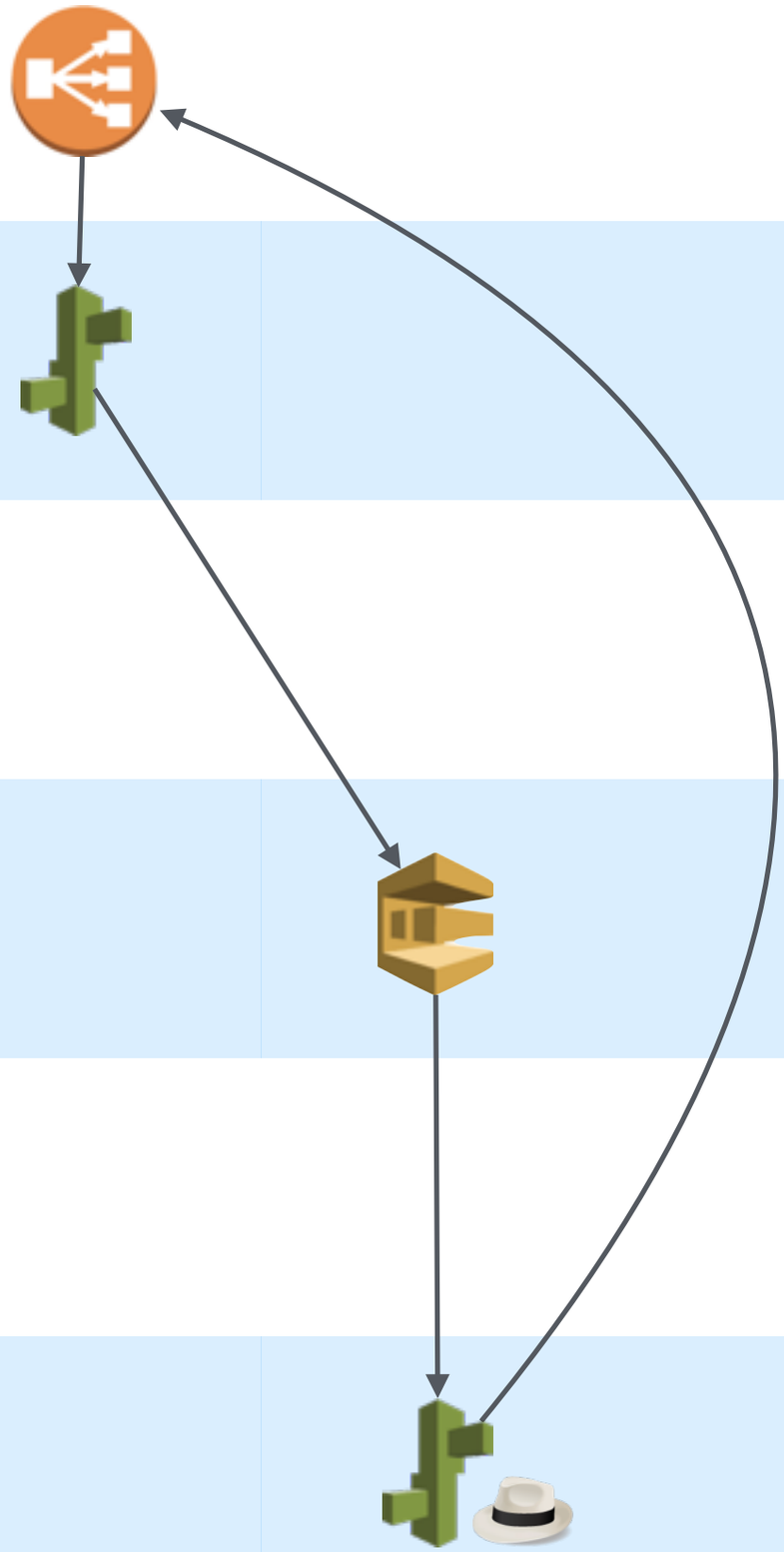
queue tier

Two instances of SQS sending requests to two types of worker tier applications.

worker tier

Sinatra application executing Spec files against corporate APIs.

the Heartbeat



load balancer

AWS service linking external domain to auto-scaling group containing web application.

web tier

RoR application serving front-end interface, connections to database and public api behind authentication. Schedules jobs for execution.

queue tier

Two instances of SQS sending requests to two types of worker tier applications.

heartbeat tier

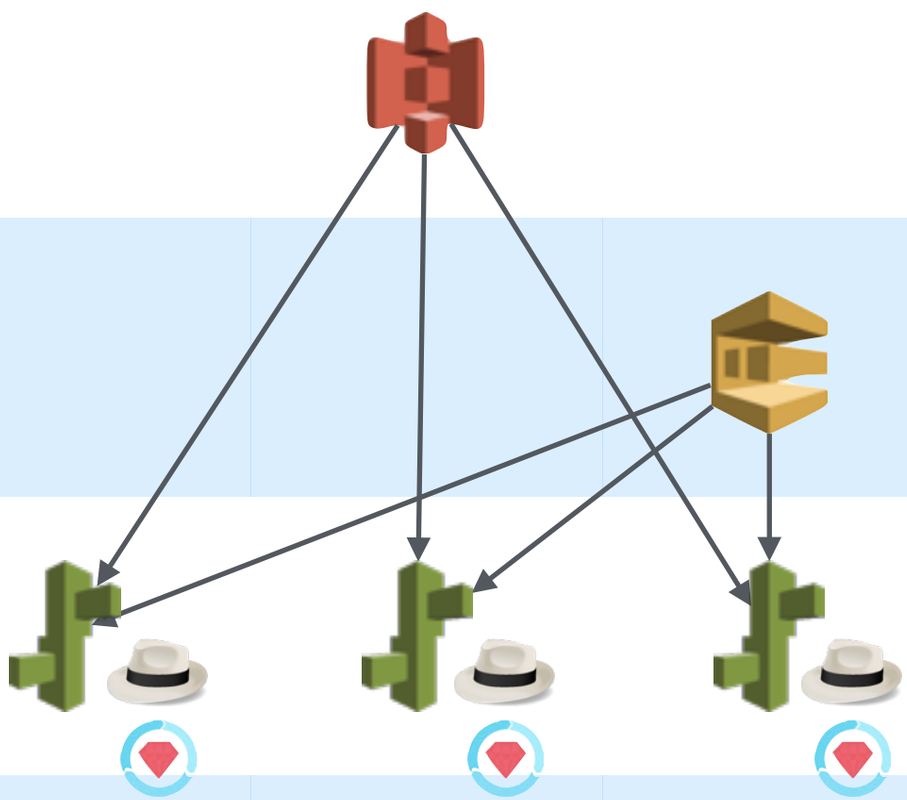
Sinatra application queries front-end for jobs ready to be executed.

the Heartbeat

- Sinatra
  - web server for worker tier

```
post '/heartbeat' do
  puts "Bump bump, #{Time.now}"

  # response Heartbeat.next_beat

  # puts response.body

  url = "#{ENV['FE_HOSTNAME']}/api/heartbeat?authentication_token=#{ENV['FE_AUTHENTICATION_TOKEN']}"

  curl_output = `curl -k -X GET #{url}`


  status 200
  body ''
```

the Media Server



data tier

S3 stores assets used in tests.
RDS serves as database to web tier
RoR application.

queue tier

Two instances of SQS sending requests
to two types of worker tier
applications.

worker tier

Sinatra application executing Spec
files against corporate APIs.

the Media Server

- Not Yet Implemented