# Title: DevOps: Creating new docker setup in staging

1. Install required gems, Update freshsales\_source\_code\_path in the script, and create a directory to save the output files and update its path in the script, save this file
2. **docker-setup.rb**

* require 'aws-sdk-s3'  
  require 'aws-sdk-acm'  
  require 'aws-sdk-route53'  
  require 'aws-sdk-opsworks'  
  require 'aws-sdk-cloudfront'  
  require 'json'  
  require 'httparty'  
  require 'byebug'  
  require 'active\_support/all'  
    
  STATUS = { INTIATE: 0, CREATED\_S3\_BUCKET: 1, CREATED\_CLOUDFRONT\_DISTRIBUTION: 2, CREATED\_SSL\_CERTIFICATE: 3, CREATED\_STACK: 4, BOOTED\_SPOTINST: 5, BOOTSTRAPED: 6, CREATED\_REDSHIFT\_DB: 7, MANUAL\_COMMANDS: 8}.freeze  
    
  @options = {  
   stack\_for: 'harsh',  
   email: 'harsh.kumar@freshworks.com',  
   reference\_docker\_prefix: 'dwarak',  
   stack\_region: 'us-east-1',  
   freshsales\_source\_code\_path: '/Users/⁨dwarak/freshsales',  
   output\_folder\_path: '/Users/dwarak/output',  
   base\_pgsql\_path: '/Users',  
   redshift\_port: 5439, #reference  
   redshift\_db: 'dwarak', #reference  
   redshift\_user: 'fs\_usr\_admin',  
   redshift\_host: 'staging-report.cdyawqmnysyw.us-east-1.redshift.amazonaws.com',  
   access\_key\_id: nil,  
   secret\_access\_key: nil,  
   route\_53\_hosted\_zone\_id: 'Z1AJ3BVLC16748', #freshmarketeer.com  
   status: 0  
  }  
    
    
  def signin\_into\_opsworks  
   puts "Signing to OpsWorks...Please Wait..."  
   @opsworks = Aws::OpsWorks::Client.new(:region => @options[:stack\_region], :credentials => fetch\_credentials)  
  end  
    
  def fetch\_credentials  
   @creds ||= Aws::Credentials.new(@options[:access\_key\_id], @options[:secret\_access\_key])  
  end  
    
  def fetch\_all\_stacks(refresh = false)  
   if refresh  
   @all\_stacks = @opsworks.describe\_stacks[:stacks]  
   end  
   @all\_stacks ||= @opsworks.describe\_stacks[:stacks]  
  end  
    
  def check\_stack\_already\_exists\_or\_not  
   puts "Stack already exists with same name or not..."  
   current\_stacks = fetch\_all\_stacks.select { |stack| stack.name.include?("#{@options[:stack\_for]}-") || stack.name.include?("-#{@options[:stack\_for]}") }.map(&:name)  
   if current\_stacks.any?  
   puts "Cannot create stack for : #{@options[:stack\_for]}. Conflicting stacks : #{current\_stacks.inspect}"  
   update\_status(STATUS[:CREATED\_STACK])  
   end  
   puts "No name conflicts...Can create a stack with this name..."  
   # press\_enter  
  end  
    
  def create\_new\_bucket  
   s3 = Aws::S3::Client.new(region: @options[:stack\_region], :credentials => fetch\_credentials)  
   puts "Listing all buckets in AWS..."  
    
   bucket\_name = "freshsales-assets-#{@options[:stack\_for]}"  
   already\_present = s3.list\_buckets({}).buckets.select { |bucket| bucket.name.eql?(bucket\_name) }.map(&:name)  
    
   unless already\_present.empty?  
   puts "Bucket Already Exists With name : #{bucket\_name}"  
   else  
   puts "Creating asset bucket for app : #{bucket\_name}..."  
   s3.create\_bucket(bucket: bucket\_name)  
   puts "Updating CORS for asset bucket : #{bucket\_name}..."  
   s3.put\_bucket\_cors({  
   bucket: bucket\_name,  
   cors\_configuration: { cors\_rules: [{ allowed\_headers: ["Content-\*"], allowed\_methods: ["GET"], allowed\_origins: ["\*"], max\_age\_seconds: 3000 }]}  
   })  
   # press\_enter  
   end  
  end  
    
  def create\_new\_cloudfront\_distribution  
   cloudfront = Aws::CloudFront::Client.new(region: @options[:stack\_region], :credentials => fetch\_credentials)  
   puts "Listing all cloudfront distributions..."  
    
   already\_present = cloudfront.list\_distributions.distribution\_list.items.select { |distribution| distribution.origins.items[0].domain\_name.include?("freshsales-assets-#{@options[:stack\_for]}.s3.amazonaws.com") }.first  
    
   if already\_present  
   puts "Using Previously Created CloudFront distribution itself..."  
   @cloudfront\_details = already\_present  
   else  
   puts "Creating CloudFront distribution..."  
   created\_distribution = cloudfront.create\_distribution({  
   distribution\_config: {  
   caller\_reference: "dist-for-#{@options[:stack\_for]}",  
   origins: {  
   quantity: 1,  
   items: [{ id: "S3-freshsales-assets-#{@options[:stack\_for]}", domain\_name: "freshsales-assets-#{@options[:stack\_for]}.s3.amazonaws.com", s3\_origin\_config: {origin\_access\_identity: ""} }],  
   },  
   default\_cache\_behavior: {  
   target\_origin\_id: "S3-freshsales-assets-#{@options[:stack\_for]}",  
   forwarded\_values: { query\_string: false, cookies: { forward: "none" }, headers: { quantity: 1, items: ["Origin"] } },  
   trusted\_signers: { enabled: false, quantity: 0 },  
   viewer\_protocol\_policy: "allow-all",  
   min\_ttl: 0,  
   allowed\_methods: { quantity: 2, items: ["HEAD", "GET"], cached\_methods: { quantity: 2, items: ["HEAD", "GET"] } },  
   smooth\_streaming: false,  
   default\_ttl: 86400,  
   max\_ttl: 31536000,  
   compress: false,  
   },  
   comment: "",  
   price\_class: "PriceClass\_All",  
   enabled: true  
   }  
   })  
   @cloudfront\_details = created\_distribution[:distribution]  
   end  
   # press\_enter  
  end  
    
  def create\_new\_certificate  
   acm = Aws::ACM::Client.new(:region => @options[:stack\_region], :credentials => fetch\_credentials)  
   domain\_name = "\*.#{@options[:stack\_for]}.freshmarketeer.com"  
    
   puts "Listing all Certificates..."  
   already\_present = acm.list\_certificates({ certificate\_statuses: ["PENDING\_VALIDATION", "ISSUED"] }).certificate\_summary\_list.select { |cert| cert.domain\_name == domain\_name }.first  
    
   if already\_present  
   puts "Certificate already exists with domain : '#{domain\_name}'...Please Check manually..."  
   else  
   resp = acm.request\_certificate(domain\_name: domain\_name, validation\_method: 'DNS')  
   puts "Certificate for #{@options[:stack\_for]} is requested...Please approve it..."  
   sleep(5)  
   requested\_cert = acm.describe\_certificate(certificate\_arn: resp.certificate\_arn)  
   ar53 = Aws::Route53::Client.new(region: @options[:stack\_region], :credentials => fetch\_credentials)  
   verification\_record = requested\_cert.certificate.domain\_validation\_options[0]  
   ar53.change\_resource\_record\_sets(  
   {change\_batch: {  
   changes: [{action: 'CREATE',  
   resource\_record\_set:  
   {name: verification\_record.resource\_record.name,  
   resource\_records: [{value: verification\_record.resource\_record.value}],  
   ttl: 300,  
   type: 'CNAME'}}]  
   },  
   hosted\_zone\_id: @options[:route\_53\_hosted\_zone\_id]  
   })  
   puts 'Added domain verification records in route 53...'  
   end  
   # press\_enter  
  end  
    
  def create\_new\_stack  
   puts "Getting stack config from '#{@options[:reference\_docker\_prefix]}-docker'..."  
   reference\_stack = fetch\_all\_stacks.select { |stack| stack.name == "#{@options[:reference\_docker\_prefix]}-docker" }.first  
   custom\_json = JSON.parse(reference\_stack.custom\_json.gsub(@options[:reference\_docker\_prefix], @options[:stack\_for]))  
    
   puts "Updating stack settings json with new details..."  
   custom\_json["freshsales"]["config/asset\_sync.yml"]["staging"]["asset\_host\_url\_https"] = "https://#{@cloudfront\_details.domain\_name}"  
   custom\_json["freshsales"]["config/asset\_sync.yml"]["staging"]["opsworks\_stack\_name"] = "#{@options[:stack\_for]}-docker"  
   custom\_json["assets"]["https\_url"] = @cloudfront\_details.domain\_name  
   custom\_json["assets"]["cloudfront\_distribution"] = @cloudfront\_details.id  
    
   puts "Cloning stack..."  
   @stack = @opsworks.clone\_stack({  
   source\_stack\_id: reference\_stack.stack\_id,  
   name: "#{@options[:stack\_for]}-docker",  
   attributes: {"Color"=>"rgb(57, 131, 94)"},  
   service\_role\_arn: reference\_stack.service\_role\_arn,  
   custom\_json: JSON.pretty\_generate(custom\_json),  
   clone\_permissions: true,  
   clone\_app\_ids: @opsworks.describe\_apps(stack\_id: reference\_stack.stack\_id)[:apps].map(&:app\_id)  
   })  
   @layer\_id = @opsworks.describe\_layers(stack\_id: @stack.stack\_id).layers.first.layer\_id  
   # press\_enter  
  end  
    
  def delete\_stack  
   return unless @stack.present?  
   puts 'Do you want to delete stack? (y/n)'  
   return unless check\_yes  
   @opsworks.describe\_apps(stack\_id: @stack.stack\_id)[:apps].map(&:app\_id).each { |a| @opsworks.delete\_app(app\_id: a) }  
   @opsworks.delete\_stack(stack\_id: @stack.stack\_id)  
   puts 'Stack deleted...'  
  end  
    
    
    
  def create\_and\_boot\_spot\_inst  
   elastigroup\_config = spotinst\_request('get\_elastigroup', "#{@options[:reference\_docker\_prefix]}-docker")  
   puts "Using reference docker spotinst configuration..."  
   elastigroup\_config = replace\_params(elastigroup\_config, @stack.stack\_id, @layer\_id)  
   spotinst\_request('create\_elastigroup', elastigroup\_config)  
   puts "Created elastigroup in spotinst..."  
   # press\_enter  
  end  
    
  def replace\_params(existing\_params, stack\_id, layer\_id)  
   existing\_params.except!('id','createdAt','updatedAt')  
   existing\_params['name'] = "#{@options[:stack\_for]}-docker"  
   existing\_params['compute']['launchSpecification']['userData'] = Base64.encode64(get\_user\_data(stack\_id, layer\_id)).gsub(/\n/,'')  
   existing\_params['thirdPartiesIntegration']['opsWorks']['layerId'] = layer\_id.to\_s  
   existing\_params['compute']['availabilityZones'] = existing\_params['compute']['availabilityZones'].map {|zone| zone.slice('name','subnetIds')}  
   { 'group' => existing\_params }  
  end  
    
  def get\_user\_data(stack\_id, layer\_id)  
   "#!/bin/bash  
   curl -fsSL https://s3.amazonaws.com/spotinst-public/integrations/opsworks/spotinst\_aws\_opsworks\_v5.sh | \  
   OPSWORKS\_STACK\_TYPE=\"CLASSIC\" \  
   OPSWORKS\_STACK\_ID=\"#{stack\_id}\" \  
   OPSWORKS\_LAYER\_ID=\"#{layer\_id}\" \  
   bash  
   sudo mount /dev/sdb /data"  
  end  
    
  def spotinst\_request(method, param)  
   case method  
    
   when 'get\_elastigroup'  
   r = HTTParty.get('https://api.spotinst.io/aws/ec2/group',  
   query: { 'accountId' => 'act-4d7f5f6d', 'name' => param},  
   headers: {'Authorization' => 'Bearer f3efa47a85a306befae57d7debbe614b9e11339d45e698e2e57aa6fafcb3b022',  
   'Content-Type' => 'application/json'}  
   )  
   r['response']['items'].first  
   return r['response']['items'].first  
   when 'create\_elastigroup'  
   r = HTTParty.post('https://api.spotinst.io/aws/ec2/group',  
   query: { 'accountId' => 'act-4d7f5f6d'},  
   headers: {'Authorization' => 'Bearer f3efa47a85a306befae57d7debbe614b9e11339d45e698e2e57aa6fafcb3b022',  
   'Content-Type' => 'application/json'},  
   body: param.to\_json  
   )  
   if r["response"]["status"]["code"] == '200'  
   puts "Created #{r["response"]["status"]["code"]}"  
   end  
   end  
  end  
    
  def create\_redshift\_db  
   puts "\nReading from #{@options[:freshsales\_source\_code\_path]}...\nPlease Wait.."  
   pgsql\_content = "CREATE DATABASE #{@options[:stack\_for]};\n\\c #{@options[:stack\_for]};\n"  
   Dir["#{@options[:freshsales\_source\_code\_path]}/config/redshift/\*.sql"].each do |file\_name|  
   pgsql\_content += "\n" + File.open(file\_name).read.gsub('freshsales', @options[:stack\_for])  
   end  
    
   pgsql\_content = pgsql\_content.sub(" contact\_status\_id NUMERIC(20, 0) ENCODE MOSTLY32,\n", '')  
   pgsql\_content += "GRANT ALL ON TABLE contact TO fs\_usr\_admin;\nGRANT ALL ON TABLE deal TO fs\_usr\_admin;\nGRANT ALL ON TABLE lead TO fs\_usr\_admin;\nGRANT ALL ON TABLE sales\_account TO fs\_usr\_admin;\nGRANT ALL ON TABLE contact\_deal\_association TO fs\_usr\_admin;\nGRANT SELECT ON TABLE contact TO fs\_usr\_readonly;\nGRANT SELECT ON TABLE deal TO fs\_usr\_readonly;\nGRANT SELECT ON TABLE lead TO fs\_usr\_readonly;\nGRANT SELECT ON TABLE sales\_account TO fs\_usr\_readonly;\nGRANT SELECT ON TABLE contact\_deal\_association TO fs\_usr\_readonly;\nCOMMIT;\n"  
    
   pgsql\_file\_path = "#{@options[:output\_folder\_path]}/#{@options[:stack\_for]}.pgsql"  
   File.open(pgsql\_file\_path, 'w+') do |file|  
   file.write(pgsql\_content)  
   file.close  
   end  
    
   command = "PGPASSWORD=freshreportAdmin321 psql --host=#{@options[:redshift\_host]} --port=#{@options[:redshift\_port]} --user=#{@options[:redshift\_user]} --dbname=#{@options[:redshift\_db]} < #{pgsql\_file\_path}"  
   puts "executing command - #{command}"  
   `#{command}`  
   press\_enter  
  end  
    
  def generate\_other\_needed\_commands  
   puts "Generating needed Commands ...\nReading from #{@options[:freshsales\_source\_code\_path]}...\nPlease Wait..."  
    
    
   file\_content = ""  
   file\_content += "\n\n\*\*\*\*\*\*\*\*\*\*\* Gmail Auth Callbacks Setup \*\*\*\*\*\*\*\*\*\*\*\n\n"  
   file\_content += "Login to - https://console.developers.google.com/apis/credentials/oauthclient/1017055921322-k7i8o85lld9s1uv250phb55l93e79bos.apps.googleusercontent.com?project=freshmarketeer-86\n"  
   file\_content += "Add these urls in the client-\n"  
   file\_content += "https://login.#{@options[:stack\_for]}.freshmarketeer.com/auth/google\_oauth2/callback\n"  
   file\_content += "https://login.#{@options[:stack\_for]}.freshmarketeer.com/integrations/oauth2/google\_calendar/callback\n"  
   file\_content += "https://login.#{@options[:stack\_for]}.freshmarketeer.com/integrations/oauth2/google\_contacts/callback\n\n"  
    
   file\_content += "Login to - https://console.developers.google.com/apis/credentials/oauthclient/167958100302-71mg7bcv1lje40mp9ej1scv7jb0d0qdm.apps.googleusercontent.com?project=freshsales-admin-dev\n"  
   file\_content += "Add these urls in the client-\n"  
   file\_content += "https://admin.#{@options[:stack\_for]}.freshmarketeer.com:4000/users/auth/google\_oauth2/callback\n\n"  
    
   file\_content += "Login to - https://console.developers.google.com/apis/credentials/domainverification?project=freshmarketeer-86\n"  
   file\_content += "Add these urls in the client-\n"  
   file\_content += "gcalendar.#{@options[:stack\_for]}.freshmarketeer.com\n"  
   file\_content += "https://console.aws.amazon.com/route53/home?#resource-record-sets:Z1AJ3BVLC16748"  
    
   file\_content += "\n\n\*\*\*\*\*\*\*\*\*\* All Urls \*\*\*\*\*\*\*\*\*\*\*\n\n"  
   file\_content += "Email Address: #{@options[:email]}\n"  
   file\_content += "Freshsales Application Development Account URL : "  
   file\_content += "https://development.#{@options[:stack\_for]}.freshmarketeer.com/\n"  
   file\_content += "Admin Console URL : https://admin.#{@options[:stack\_for]}.freshmarketeer.com:4000/\n"  
    
   File.open("#{@options[:output\_folder\_path]}/#{@options[:stack\_for]}.txt", 'w+') do |file|  
   file.write(file\_content)  
   file.close  
   end  
    
   puts "file store in #{@options[:output\_folder\_path]}#{@options[:stack\_for]}.txt"  
   puts File.read("#{@options[:output\_folder\_path]}/#{@options[:stack\_for]}.txt")  
  end  
    
  def initialize\_stack\_and\_instance\_details  
   @stack = fetch\_all\_stacks(true).select { |stack| stack.name.include?("#{@options[:stack\_for]}-docker") }.first  
   if @stack.nil?  
   puts "Stack not Found with name : #{@options[:stack\_for]}"  
   exit (0)  
   end  
   wait\_till\_instance\_has\_ip  
  end  
    
  def fetch\_staging\_instance  
   return @staging\_instance unless @staging\_instance.nil?  
   staging\_stack = fetch\_all\_stacks.select { |stack| stack.name.include?("staging-rails") }.first  
   @staging\_instance = @opsworks.describe\_instances(stack\_id: staging\_stack.stack\_id)[:instances].select { |instance| instance.hostname.include?("appserver") }.first  
  end  
    
  def wait\_till\_instance\_has\_ip(retry\_time = 0)  
   puts  
   loop do  
   @instance = @opsworks.describe\_instances(stack\_id: @stack.stack\_id)[:instances].first  
   if !(@instance.present? && @instance.public\_ip.present? && @instance.status == 'online')  
   sleep(2)  
   print '.'  
   else  
   puts "\nWaiting for docker containers to boot..."  
   sleep(600) # wait for docker containers to boot  
   return  
   end  
   end  
  end  
    
  def run\_bootstrap\_recipe  
   initialize\_stack\_and\_instance\_details if @instance.nil?  
   fetch\_staging\_instance if @staging\_instance.nil?  
    
   "Running bootstrap recipe in the instance..."  
   resp = @opsworks.create\_deployment({stack\_id: @stack.stack\_id, instance\_ids: [@instance.instance\_id],  
   command: { name: "execute\_recipes", args: { "recipes" => ["docker::db\_bootstrap"] } },  
   comment: "Bootstrap",  
   custom\_json: JSON.pretty\_generate({bootstrap: { apps: ["freshsales","freshsales\_admin"], domain: "development", email: @options[:email]}}),  
   })  
    
  end  
    
  def show\_and\_get\_input  
   loop do  
   puts 'Current Configuration'  
   puts JSON.pretty\_generate(@options)  
    
   puts "\nDo you wish to change configurations? (y/n)"  
   return unless check\_yes  
    
   @options.keys.each do |k|  
   if @options[k]  
   puts "\n#{k}: #{@options[k]}\n('ENTER' to proceed / 'y' to change): "  
    
   next unless check\_yes  
   end  
    
   puts "\nEnter value for #{k}:"  
   new\_value = gets  
   @options[k] = new\_value.strip  
   end  
   end  
  end  
    
  def check\_yes  
   response = gets  
   response.strip!  
   response == 'y' || response == 'Y' || response == 'Yes' || response == 'yes' || response == 'yeah' || response == 'Yeah'  
  end  
    
  def press\_enter  
   puts "\n\npress ENTER to proceed"  
   gets  
  end  
    
  def wait\_for\_confirmation(t)  
   loop do  
   sleep(1)  
   3.times.each do |i|  
   print '.'  
   sleep(1)  
   end  
   puts t  
   return if check\_yes  
   end  
  end  
    
  def update\_status(status)  
   @options[:status] = status if @options[:status] < status  
  end  
    
  def docker\_create\_interaction  
    
   show\_and\_get\_input  
    
   signin\_into\_opsworks  
    
   check\_stack\_already\_exists\_or\_not if @options[:status].zero?  
    
   create\_new\_bucket if @options[:status] < STATUS[:CREATED\_S3\_BUCKET]  
   update\_status(STATUS[:CREATED\_S3\_BUCKET])  
    
   create\_new\_cloudfront\_distribution if @options[:status] < STATUS[:CREATED\_CLOUDFRONT\_DISTRIBUTION]  
   update\_status(STATUS[:CREATED\_CLOUDFRONT\_DISTRIBUTION])  
    
   create\_new\_certificate if @options[:status] < STATUS[:CREATED\_SSL\_CERTIFICATE]  
   update\_status(STATUS[:CREATED\_SSL\_CERTIFICATE])  
    
   create\_new\_stack if @options[:status] < STATUS[:CREATED\_STACK]  
   update\_status(STATUS[:CREATED\_STACK])  
    
   create\_and\_boot\_spot\_inst if @options[:status] < STATUS[:BOOTED\_SPOTINST]  
   update\_status(STATUS[:BOOTED\_SPOTINST])  
   puts "Stack created successfully... Wait for instance to boot"  
    
   # wait\_for\_confirmation("\nHas the instance created under stack? (y/n)")  
   # wait\_for\_confirmation("\nIs the instance 'setup' over? (y/n)")  
    
   run\_bootstrap\_recipe if @options[:status] < STATUS[:BOOTSTRAPED]  
   update\_status(STATUS[:BOOTSTRAPED])  
    
   create\_redshift\_db if @options[:status] < STATUS[:CREATED\_REDSHIFT\_DB]  
   update\_status(STATUS[:CREATED\_REDSHIFT\_DB])  
    
   generate\_other\_needed\_commands  
  rescue => e  
   puts "Error: #{e.message}\n #{e.backtrace.take(10).join("\n")}"  
   delete\_stack  
  end  
  # COMMAND TO RUN: ruby create\_docker\_with\_spotinst.rb  
    
  docker\_create\_interaction

1. Update the "stack-for" and "email" values and run the script.
2. The above script will create docker stack, and will also add and boot an instance.
3. The script would also generate a file with path - 'output\_folder\_path/stack\_for.txt' - ex - '/Users/reshmauma/output/reshma.txt',  
   which would contain other necessary commands to be run manually, such as  
   registering domains in google dev console.