



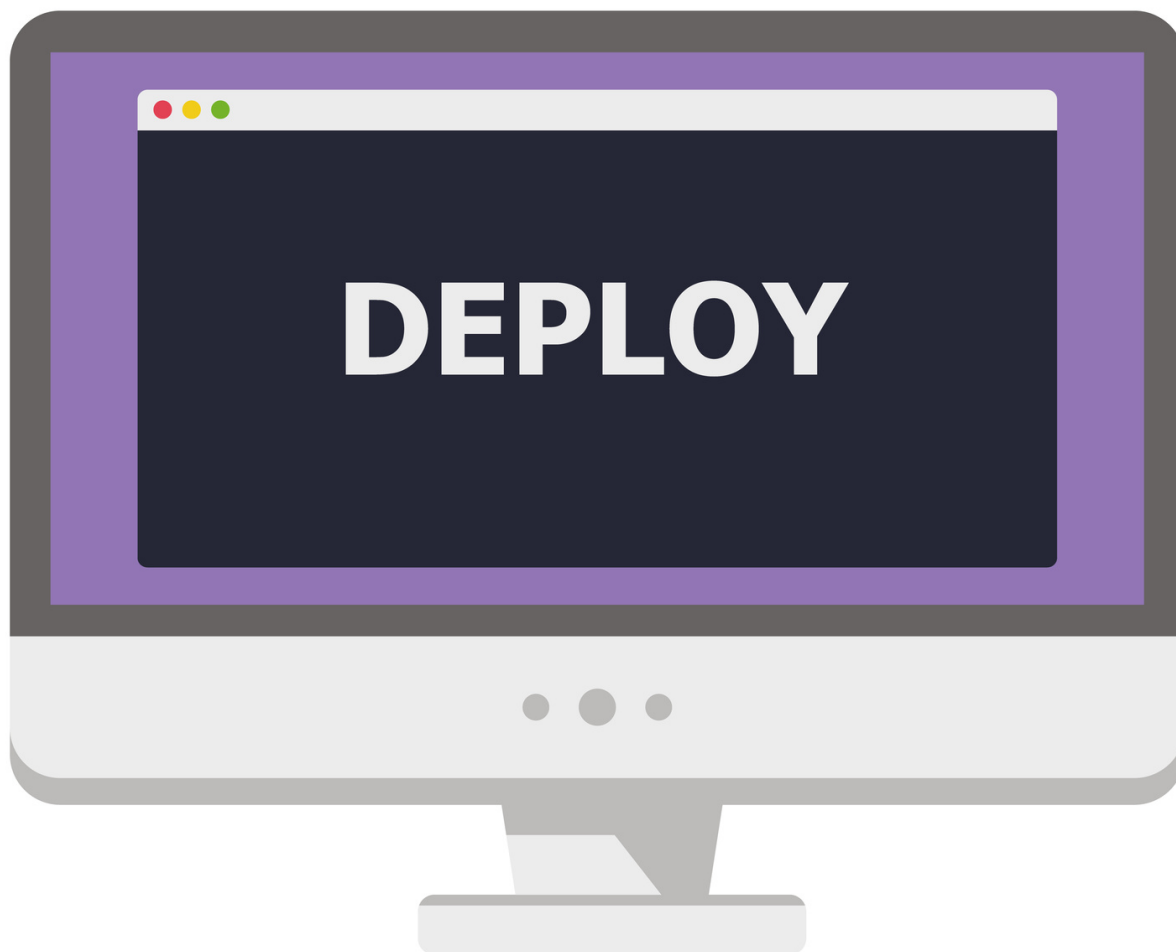
Deploy Your Rails App to AWS

By Devdatta Kane

Ruby

August 20, 2015

Share:



As developers, we are usually concerned about the development part of any application. We don't think much about the deployment part as we consider it to be a responsibility of the SysAdmins. But many times, we don't have a dedicated SysAdmin available, so we have to put on the SysAdmin hat and get things done. There are many options to deploy your Rails application. Today, I will cover how to deploy a Rails application to Amazon Web Services (AWS) using Capistrano.

We will use the Puma + Nginx + PostgreSQL stack. Puma will be the application server, Nginx the reverse proxy, and PostgreSQL is the database server. This stack can be used on MRI Ruby or JRuby as well. Most of the steps remain same for both rubies, but I'll highlight where they differ as well.

If you have an existing application, you can skip the following section and jump directly to next section.

Sample Rails Application

Let's create a sample Rails application with a contact model and CRUD. The app uses Rails 4.2 and PostgreSQL:

```
rails new contactbook -d postgresql
```

After the application is generated, create a Contact model and CRUD:

```
cd contactbook  
rails g scaffold Contact name:string address:string city:s
```

Setup your database username and password in `config/database.yml`, then create and migrate the database:

```
rake db:create && rake db:migrate
```

Let's check how its working:

```
rails s
```

Point your favorite browser to <http://localhost:3000/contacts> and check if everything is working properly.

Configuring Puma & Capistrano

We will now configure application for deployment. As previously mentioned, Puma is the application server and Capistrano as our deployment tool. Capistrano provides integration for Puma and RVM, so add those gems to the Gemfile. We will also use the figaro gem to save application configuration, such as the production database password and secret key:

```
gem 'figaro'
gem 'puma'
group :development do
  gem 'capistrano'
  gem 'capistrano3-puma'
  gem 'capistrano-rails', require: false
  gem 'capistrano-bundler', require: false
  gem 'capistrano-rvm'
end
```

Install the gems via bundler:

```
bundle install
```

It's time to configure Capistrano, first by generating the config file, as follows:

```
cap install STAGES=production
```

This will create configuration files for Capistrano at **config/deploy.rb** and **config/deploy/production.rb**. **deploy.rb** is the main configuration file and **production.rb** contains environment specific settings, such as server IP, username, etc.

Add the following lines into the **Capfile**, found in the root of the application. The Capfile includes RVM, Rails, and Puma integration tasks when finished:

```
require 'capistrano/bundler'
require 'capistrano/rvm'
require 'capistrano/rails/assets' # for asset handling add
require 'capistrano/rails/migrations' # for running migrations
require 'capistrano/puma'
```

Now, edit **deploy.rb** as follows:

```
lock '3.4.0'

set :application, 'contactbook'
set :repo_url, 'git@github.com:devdatta/contactbook.git' # Edit this
set :branch, :master
set :deploy_to, '/home/deploy/contactbook'
set :pty, true
set :linked_files, %w{config/database.yml config/application.yml}
set :linked_dirs, %w{bin log tmp/pids tmp/cache tmp/sockets vendor/}
set :keep_releases, 5
set :rvm_type, :user
set :rvm_ruby_version, 'jruby-1.7.19' # Edit this if you are using

set :puma_rackup, -> { File.join(current_path, 'config.ru') }
set :puma_state, "#{shared_path}/tmp/pids/puma.state"
set :puma_pid, "#{shared_path}/tmp/pids/puma.pid"
```

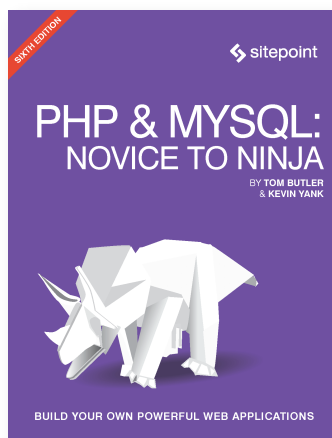
```
set :puma_bind, "unix://#{shared_path}/tmp/sockets/puma.sock" #c
set :puma_conf, "#{shared_path}/puma.rb"
set :puma_access_log, "#{shared_path}/log/puma_error.log"
set :puma_error_log, "#{shared_path}/log/puma_access.log"
set :puma_role, :app
set :puma_env, fetch(:rack_env, fetch(:rails_env, 'production'))
set :puma_threads, [0, 8]
set :puma_workers, 0
set :puma_worker_timeout, nil
set :puma_init_active_record, true
set :puma_preload_app, false
```

We will edit the **production.rb** later, since we don't know the server IP and other details yet.

Also, create **config/application.yml** to save any environment specific settings in the development environment. This file is used by the figaro gem to load the settings into environment variables. We will create the same file on the production server as well.

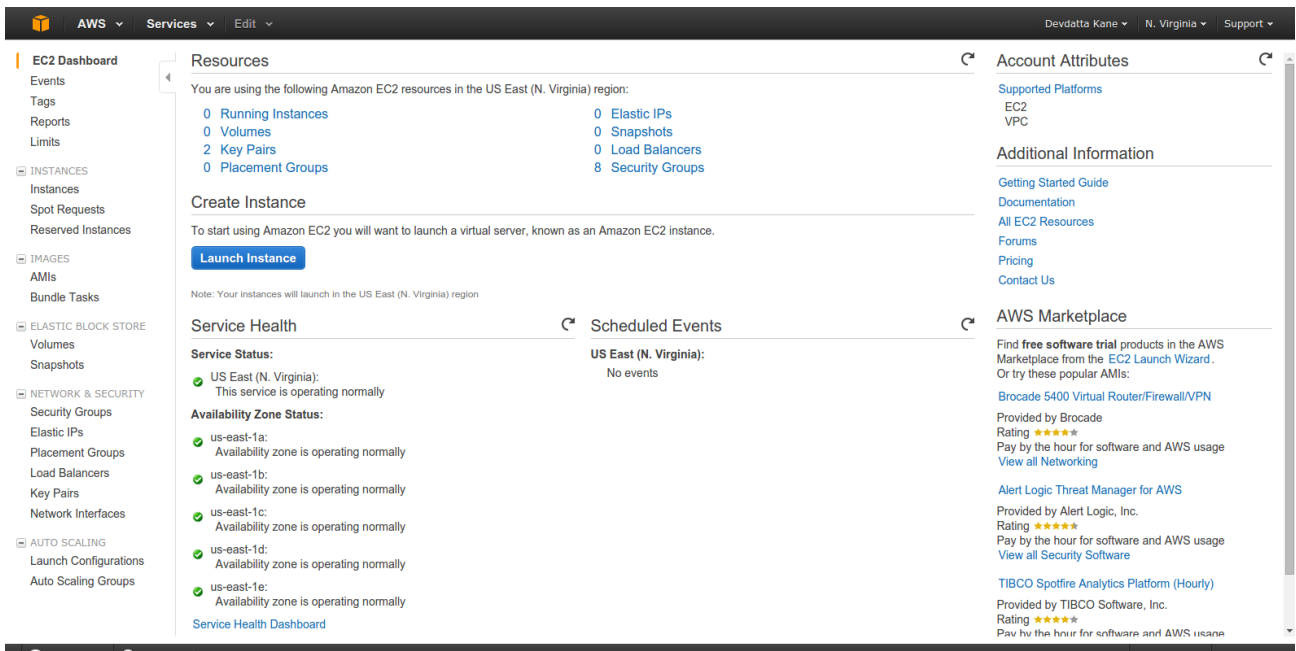
One thing to remember is to exclude **config/database.yml** and **config/application.yml** from the Git repository. Both files contain sensitive data which should not be checked into version control for obvious security concerns.

Creating an EC2 Instance

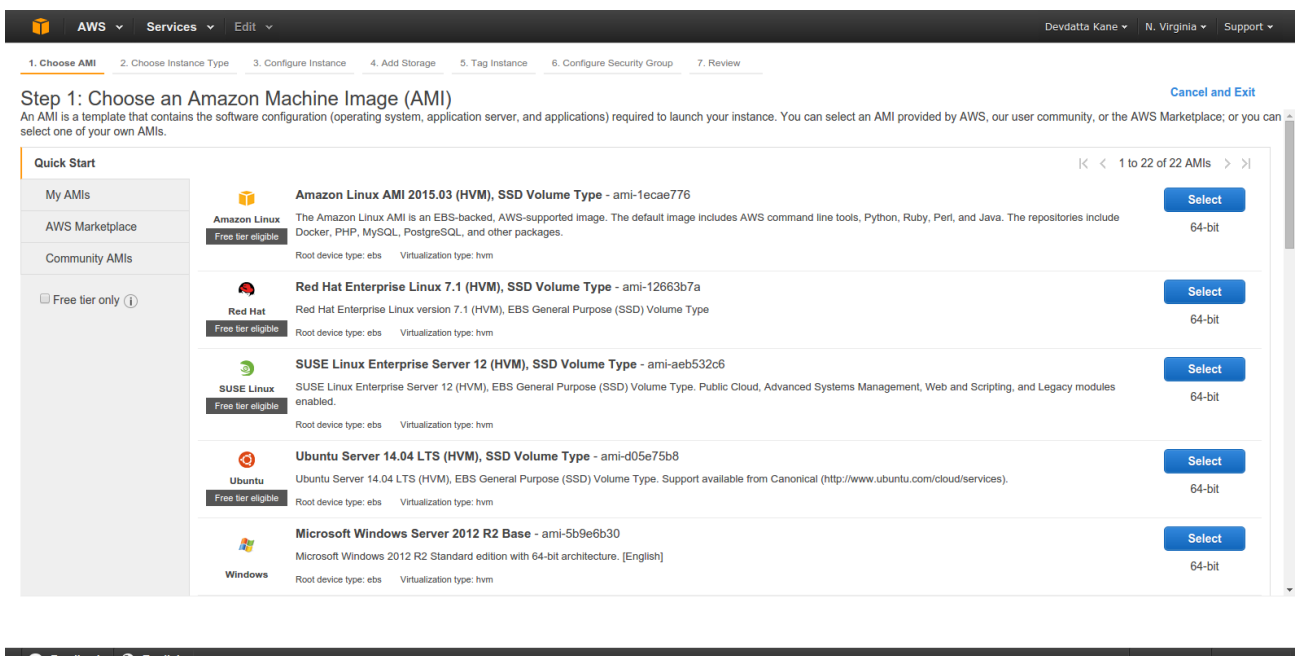


👉 If 35% of ALL websites are WordPress. is PHP

With the application configured and ready for deployment, it's time to launch a new EC2 instance. Log in to the [EC2 Management Console](#) (obviously, you'll need to sign up for an AWS account):



Click 'Launch Instance':



Select an Amazon Machine Image (AMI). We will be using 'Ubuntu Server 14.04 LTS':

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: **All instance types** **Current generation** [Show/Hide Columns](#)

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

T2 instances are VPC-only. Your T2 instance will launch into a VPC and subnet that we create for you. [Learn more](#) about T2 and VPC.

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
<input checked="" type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate
<input type="checkbox"/>	General purpose	m4.large	2	8	EBS only	Yes	Moderate
<input type="checkbox"/>	General purpose	m4.xlarge	4	16	EBS only	Yes	High
<input type="checkbox"/>	General purpose	m4.2xlarge	8	32	EBS only	Yes	High
<input type="checkbox"/>	General purpose	m4.4xlarge	16	64	EBS only	Yes	High

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Configure Instance Details](#)

Select the instance type as per your requirement. I am picking 't2.micro', because it is free/cheap. For a real production server, you'd want to go bigger. Click 'Next:Configure Instance Details' to continue.

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances: 1

Purchasing option: ☐ Request Spot Instances

Network: vpc-76869813 (172.30.0.0/16) [Create new VPC](#)

Subnet: subnet-ad243897 (172.30.4.0/24) | us-east-1e [Create new subnet](#)
251 IP Addresses available

Auto-assign Public IP: Use subnet setting (Enable)

IAM role: None [Create new IAM role](#)

Shutdown behavior: Stop

Enable termination protection: ☐ Protect against accidental termination

Monitoring: ☐ Enable CloudWatch detailed monitoring
Additional charges apply.

Tenancy: Shared tenancy (multi-tenant hardware)
Additional charges will apply for dedicated tenancy.

Network interfaces

Device	Network Interface	Subnet	Primary IP	Secondary IP addresses
eth0	New network interface	subnet-ad243897	Auto-assign	

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Add Storage](#)

The default settings are good for our tutorial. Click 'Next: Add Storage'.

The screenshot shows the 'Add Storage' step in the AWS Management Console. The top navigation bar includes the AWS logo, 'AWS' dropdown, 'Services' dropdown, and 'Edit' dropdown. The user is logged in as 'Devdatta Kane' in the 'N. Virginia' region. The progress bar shows seven steps: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage (active), 5. Tag Instance, 6. Configure Security Group, and 7. Review. The main heading is 'Step 4: Add Storage'. Below it, a paragraph explains that the instance will be launched with the following storage device settings and that additional EBS volumes can be attached after launching. A table lists the storage settings: Type (Root), Device (/dev/sda1), Snapshot (snap-4e3c6d2b), Size (8 GiB), Volume Type (General Purpose (SSD)), IOPS (24 / 3000), Delete on Termination (checked), and Encrypted (Not Encrypted). There is an 'Add New Volume' button. A blue information box states that free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. At the bottom, there are buttons for 'Cancel', 'Previous', 'Review and Launch', and 'Next: Tag Instance'.

Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-4e3c6d2b	8	General Purpose (SSD)	24 / 3000	<input checked="" type="checkbox"/>	Not Encrypted

[Learn more](#) about storage options in Amazon EC2.

[Free tier eligible customers can get up to 30 GB of EBS General Purpose \(SSD\) or Magnetic storage. \[Learn more\]\(#\) about free usage tier eligibility and usage restrictions.](#)

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Tag Instance](#)

The default storage is 8GB. Adjust as per your space requirement. Click 'Next: Tag Instance'

The screenshot shows the 'Tag Instance' step in the AWS Management Console. The top navigation bar is the same as the previous screenshot. The progress bar shows seven steps: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Tag Instance (active), 6. Configure Security Group, and 7. Review. The main heading is 'Step 5: Tag Instance'. Below it, a paragraph explains that a tag consists of a case-sensitive key-value pair and provides an example. A form with two input fields is shown: 'Key' (127 characters maximum) and 'Value' (255 characters maximum). The 'Key' field contains 'Name' and the 'Value' field contains 'rails-aws'. There is a 'Create Tag' button with the text '(Up to 10 tags maximum)'. At the bottom, there are buttons for 'Cancel', 'Previous', 'Review and Launch', and 'Next: Configure Security Group'.

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. [Learn more](#) about tagging your Amazon EC2 resources.

Key (127 characters maximum) **Value** (255 characters maximum)

Name rails-aws

[Create Tag](#) (Up to 10 tags maximum)

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Configure Security Group](#)

Enter instance name. Click 'Next: Configure Security Group'.

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source
SSH	TCP	22	Anywhere 0.0.0.0/0
HTTP	TCP	80	Anywhere 0.0.0.0/0

[Add Rule](#)

Warning

Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

[Cancel](#)
[Previous](#)
[Review and Launch](#)

Click 'Add Rule'. Select 'HTTP' from 'Type'. This is required to make nginx server accessible from the Internet. Click 'Review and Launch'

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

Improve your instances' security. Your security group, launch-wizard-1, is open to the world.

Your instances may be accessible from any IP address. We recommend that you update your security group rules to allow access from known IP addresses only. You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. [Edit security groups](#)

▼ **AMI Details** [Edit AMI](#)

Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-d05e75b8

Free tier eligible

Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
Root Device Type: ebs Virtualization type: hvm

▼ **Instance Type** [Edit instance type](#)

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

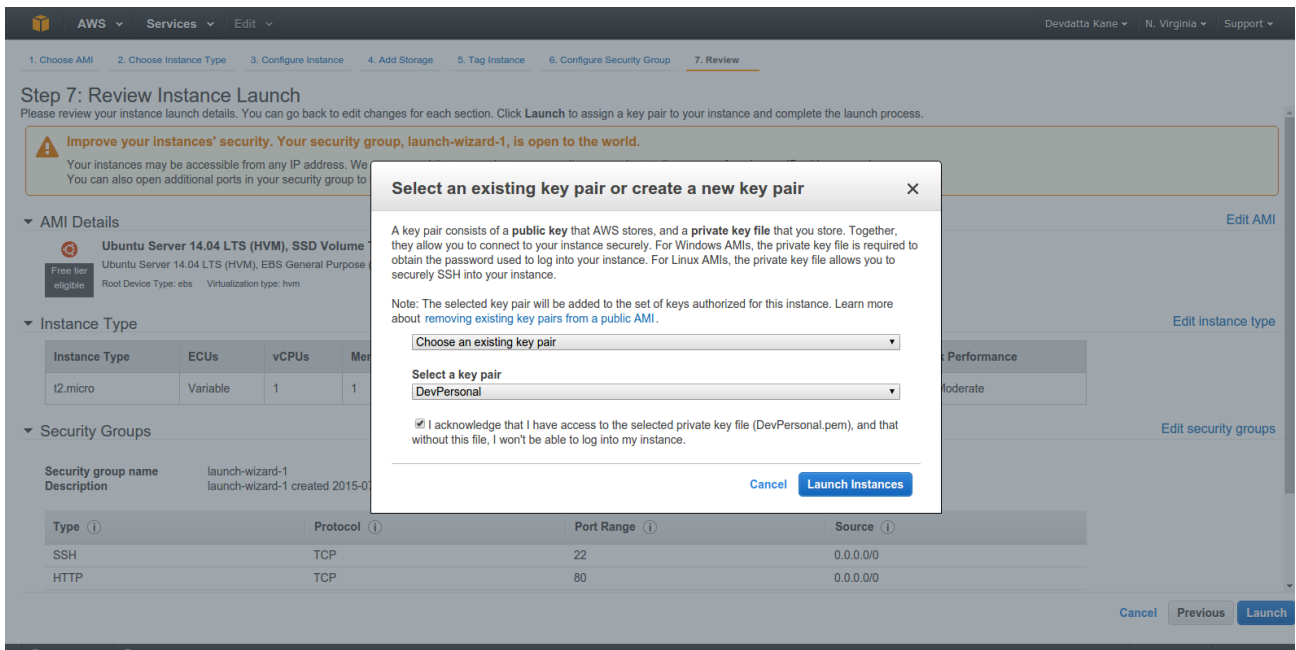
▼ **Security Groups** [Edit security groups](#)

Security group name: launch-wizard-1
Description: launch-wizard-1 created 2015-07-21T16:46:10.109+05:30

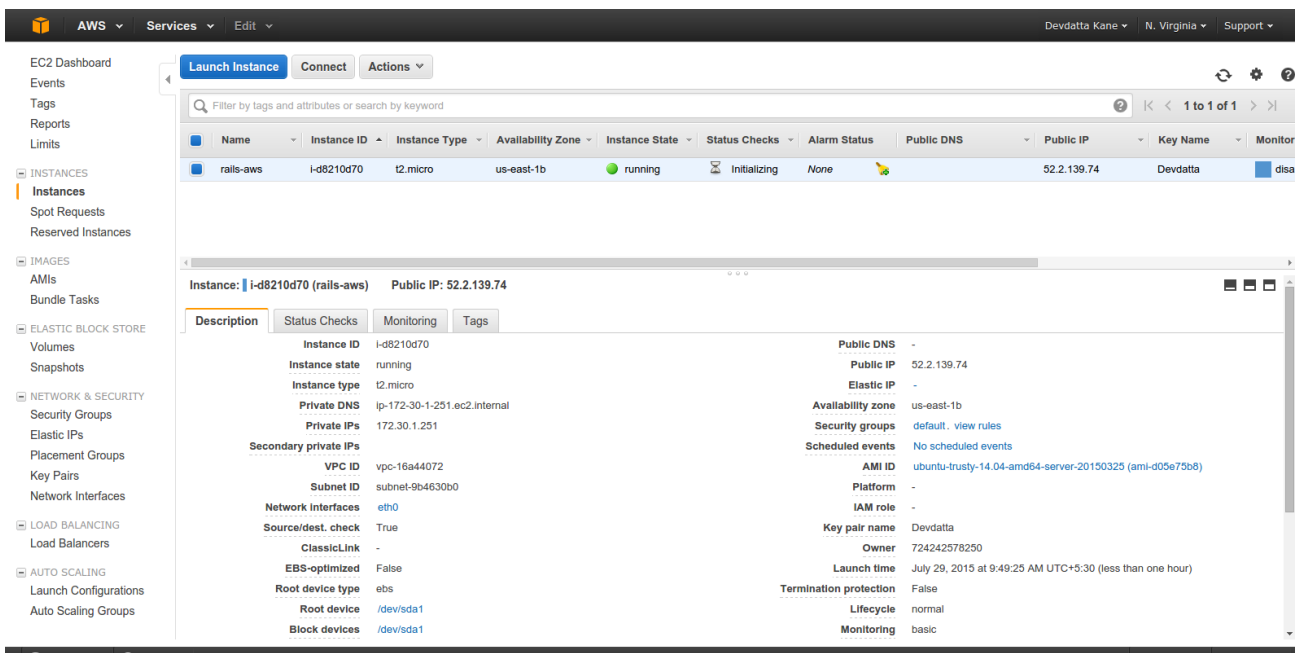
Type	Protocol	Port Range	Source
SSH	TCP	22	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0

[Cancel](#)
[Previous](#)
[Launch](#)

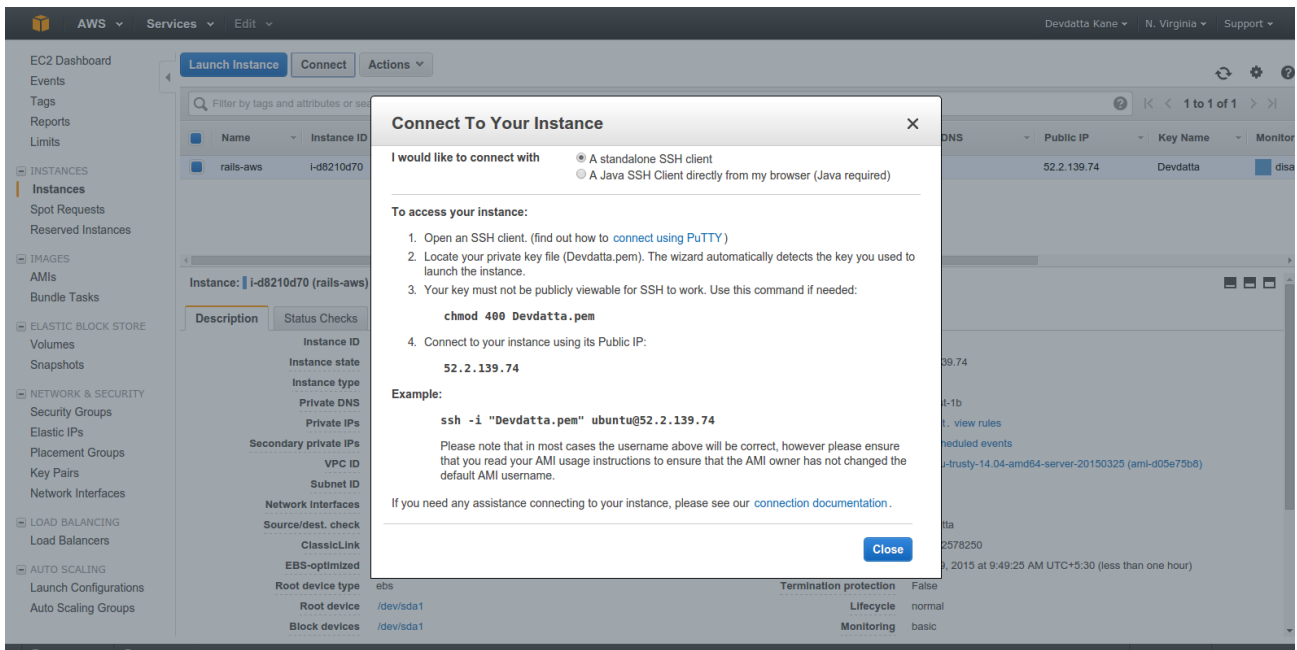
Check if all setting are correct. Click 'Launch'



Select or create a keypair to connect to instance. You **HAVE** to have the private key on your local box in order to ssh into the EC2 instance. The key should live in your `~/.ssh` directory. Click the checkbox 'I acknowledge that...' and click 'Launch Instance'. Wait for the instance to launch.



The instance should be in the 'running' state. Select the instance and click 'Connect'.



Note down the 'Public IP' address (It's 52.2.139.74 in the screenshot. Yours will be different.). We will need it to connect to the server.

Setup the Server

We have now provisioned the server and it's time to setup some basic things. First of all, SSH into the server with our selected private key. Replace 'Devdatta.pem' with the full path to your private key:

```
ssh -i "Devdatta.pem" ubuntu@52.2.139.74
```

You are logged into the brand new server. Update the existing packages first:

```
sudo apt-get update && sudo apt-get -y upgrade
```

Create a user named `deploy` for deploying the application code:

```
sudo useradd -d /home/deploy -m deploy
```

This will create the user `deploy` along with its home directory. The application will be deployed into this directory. Set the password for `deploy` user:

```
sudo passwd deploy
```

Enter password and confirm it. This password will be required by RVM for the Ruby installation. Also, add the `deploy` user to sudoers as well. Run `sudo visudo` and paste the following into the file:

```
deploy ALL=(ALL:ALL) ALL
```

Save the file and exit.

As we will be using GitHub to host our Git repository, the `deploy` user will need access to the repository for deployment. As such, we will generate a key pair for that user now:

```
su - deploy  
ssh-keygen
```

Do not set a passphrase for the key as it will be used as deploy key.

```
cat .ssh/id_rsa.pub
```

Copy the output and [set as your deploy key on GitHub](#).

Capistrano will connect to the server via ssh for deployment as the `deploy` account. Since AWS allows public key authentication only, copy the public key from your local machine to the `deploy` user account on the EC2 instance. The public key is your default `~/.ssh/id_rsa.pub` key, in most cases. On the server:

```
nano ~/.ssh/authorized_keys
```

Paste your local public key into the file. Save and exit.

Git is required for automated deployments via Capistrano, so install Git on the server:

```
sudo apt-get install git
```

If you are using JRuby, install a Java Virtual Machine (JVM):

```
sudo apt-get install openjdk-7-jdk
```

Installing Nginx

First, install Nginx which is our reverse proxy:

```
sudo apt-get install nginx
```

Now, configure the default site as our requirement. Open the site config file:

```
sudo nano /etc/nginx/sites-available/default
```

Comment out the existing content and paste the following into the file.

```
upstream app {
    # Path to Puma SOCK file, as defined previously
    server unix:/home/deploy/contactbook/shared/tmp/sockets/1
}

server {
    listen 80;
    server_name localhost;

    root /home/deploy/contactbook/public;

    try_files $uri/index.html $uri @app;

    location / {
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
        proxy_redirect off;
        proxy_http_version 1.1;
        proxy_set_header Connection '';
        proxy_pass http://app;
    }

    location ~ ^/(assets|fonts|system)/|favicon.ico|robots.txt {
        gzip_static on;
        expires max;
        add_header Cache-Control public;
    }
}
```

```
error_page 500 502 503 504 /500.html;  
client_max_body_size 4G;  
keepalive_timeout 10;  
}
```

Save the file and exit. We have configured nginx as a reverse proxy to redirect HTTP requests to the Puma application server through a UNIX socket. We will not restart nginx just yet, as the application is ready. Let's install PostgreSQL now.

Installing PostgreSQL

```
sudo apt-get install postgresql postgresql-contrib libpq-dev
```

After postgresQL is installed, create a production database and its user:

```
sudo -u postgres createuser -s contactbook
```

Set the user's password from `psql` console:

```
sudo -u postgres psql
```

After logging into the console, change the password:

```
postgres=# \password contactbook
```

Enter your new password and confirm it. Exit the console with `\q`. It's time to create a database for our application:

```
sudo -u postgres createdb -O contactbook contactbook_production
```

Installing RVM & Ruby

We will use RVM to install our desired Ruby version:

```
su - deploy
gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796  
\curl -sSL https://get.rvm.io | bash -s stable
```

This will install RVM into the `deploy` user's home directory. Logout and login again to load RVM into the `deploy` user's shell. Logout with Ctrl+D and login again with `su - deploy`.

Now, install Ruby:

For using MRI Ruby – `rvm install ruby`

For JRuby – `rvm install jruby`

After Ruby is installed, switch to installed version:

```
rvm use jruby
```

OR

```
rvm use ruby
```


Install bundler:

```
gem install bundler --no-ri --no-rdoc
```

Create the directories and files required by Capistrano. We will create the **database.yml** and **application.yml** files to store the database settings and other environment specific data:

```
mkdir contactbook  
mkdir -p contactbook/shared/config  
nano contactbook/shared/config/database.yml
```

Paste the following in **database.yml**:

```
production:  
  adapter: postgresql  
  encoding: unicode  
  database: contactbook_production  
  username: contactbook  
  password: contactbook  
  host: localhost  
  port: 5432
```

After that, create **application.yml**

```
nano contactbook/shared/config/application.yml
```

and add the following:

```
SECRET_KEY_BASE: "8a2ff74119cb2b8f14a85dd6e213fa24d8540:
```

Change the secret to a new secret using the `rake secret` command.

Okay, we're almost done with the server. Go back to your local machine to start deployment with Capistrano. Edit the **config/deploy/production.rb** to set the server IP. Open the file and paste the following into the file. Change the IP address to match with your server's IP:

```
server '52.2.139.74', user: 'deploy', roles: %w{web app db
```

Now let's start the deployment using Capistrano:

```
cap production deploy
```

Since this is the first deployment, Capistrano will create all the necessary directories and files on the server, which may take some time. Capistrano will deploy the application, migrate the database, and start the Puma application server. Now, login to the server and restart nginx so that our new configuration will be reloaded:

```
sudo service nginx restart
```

Open up the browser and point it to `/contacts`. The application should be working properly.

Listing Contacts

Name	Address	City	Phone	Email
Devdatta Kane	Shivaji Nagar	Pune	13133323	a@a.com

[Show](#) [Edit](#) [Destroy](#)

[New Contact](#)

Wrap Up

Today, we learned how to deploy Rails application on AWS with Capistrano. Our application, being simple, does not use additional services such as background jobs, so I did not cover that today. But installation and configuration of such services may be required for complex applications. But that's for another day.

Your comments and views are always welcome.



Devdatta Kane



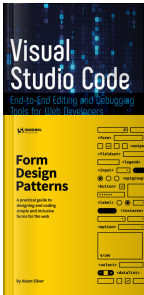
Devdatta Kane is a software developer and designer based in Pune, India. He works with [Radinik Technologies](#) building traceability solutions for a variety of industries. He is also the lead developer of [refers2](#), a CRM for small businesses. He works in Ruby on Rails, but likes to dabble with various new technologies as well. An aspiring photographer and passionate traveler, he loves traveling on his motorcycle, capturing experiences through camera.

Popular Books



Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers

[Privacy](#) - [Terms](#)



Form Design Patterns



Jump Start Git, 2nd Edition



Stuff we do

- Premium
- Forums
- Corporate memberships
- Become an affiliate
- Remote Jobs

About

- Our story

Contact

- Contact us
- FAQ
- Publish your book with us
- Write an article for us
- Advertise

Legals

- Terms of use
- Privacy policy

Connect



© 2000 – 2020 SitePoint Pty. Ltd.