RUNNING DJANGO ON AWS

Using Python 2.7 and Django >= 1.6 on AWS

This is a step-by-step guide to deploying a Django app to AWS. Note, I mention Python 2.7 specifically. This is because AWS has Python 2.6 as the default version for it's linux distribution (as of writing this guide) . Django version 1.6 is the last one to support Python 2.6 and with Django 1.7 already out in the wild, it seems prudent to set up our stack for Python 2.7 (if not 3+).

Things we'll set up:

a. Using Amazon's EC2 Management console to provision a server. For this, you need an AWS account with Amazon (aws.amazon.com).

b. Install Apache and mod_wsgi. Python 2.7 requires a few additional steps to setup and compile mod_wsgi because the mod_wsgi that EC2 currently comes with is compiled against Python 2.6.

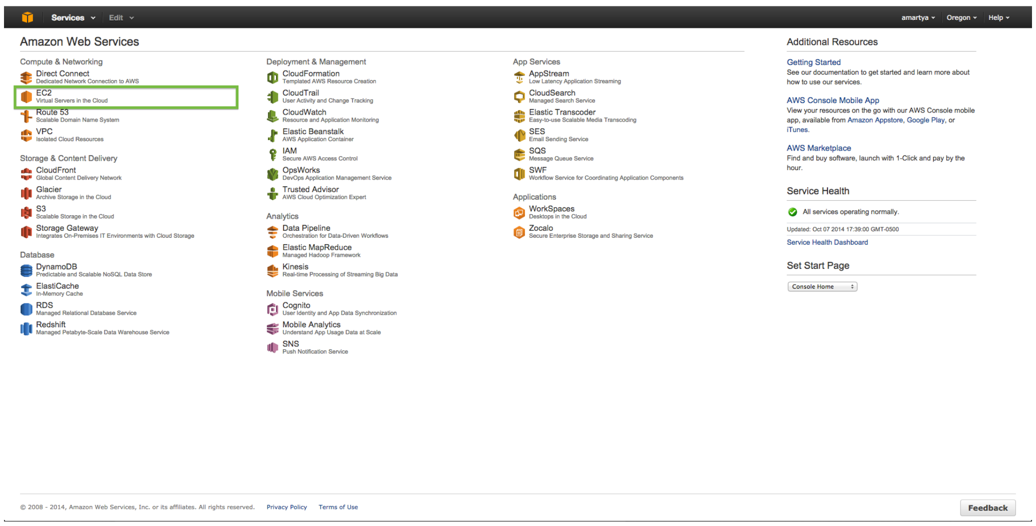
c. Set up a database. In this case, MySQL

d. Install Django

e. Create a Django app and configure it to be served via Apache

Starting an AWS instance

1.



Amazon Web Services

Compute & Networking
Direct Connect – Dedicated Network Connection to AWS
EC2 – Virtual Servers in the Cloud
Route 53 – Scalable Domain Name System
VPC – Isolated Cloud Resources

Storage & Content Delivery
CloudFront – Global Content Delivery Network
Glacier – Archive Storage in the Cloud
S3 – Scalable Storage in the Cloud
Storage Gateway – Integrates On-Premises IT Environments with Cloud Storage

Database
DynamoDB – Predictable and Scalable NoSQL Data Store
ElastiCache – In-Memory Cache
RDS – Managed Relational Database Service
Redshift – Managed Petabyte-Scale Data Warehouse Service

Deployment & Management
CloudFormation – Templated AWS Resource Creation
CloudTrail – User Activity and Change Tracking
CloudWatch – Resource and Application Monitoring
Elastic Beanstalk – AWS Application Container
IAM – Secure AWS Access Control
OpsWorks – DevOps Application Management Service
Trusted Advisor – AWS Cloud Optimization Expert

Analytics
Data Pipeline – Orchestration for Data-Driven Workflows
Elastic MapReduce – Managed Hadoop Framework
Kinesis – Real-time Processing of Streaming Big Data

Mobile Services
Cognito – User Identity and App Data Synchronization
Mobile Analytics – Understand App Usage Data at Scale
SNS – Push Notification Service

App Services
AppStream – Low Latency Application Streaming
CloudSearch – Managed Search Service
Elastic Transcoder – Easy-to-use Scalable Media Transcoding
SES – Email Sending Service
SQS – Message Queue Service
SWF – Workflow Service for Coordinating Application Components

Applications
WorkSpaces – Desktops in the Cloud
Zocalo – Secure Enterprise Storage and Sharing Service

Additional Resources

Getting Started
See our documentation to get started and learn more about how to use our services.

AWS Console Mobile App
View your resources on the go with our AWS Console mobile app, available from Amazon Appstore, Google Play, or iTunes.

AWS Marketplace
Find and buy software, launch with 1-Click and pay by the hour.

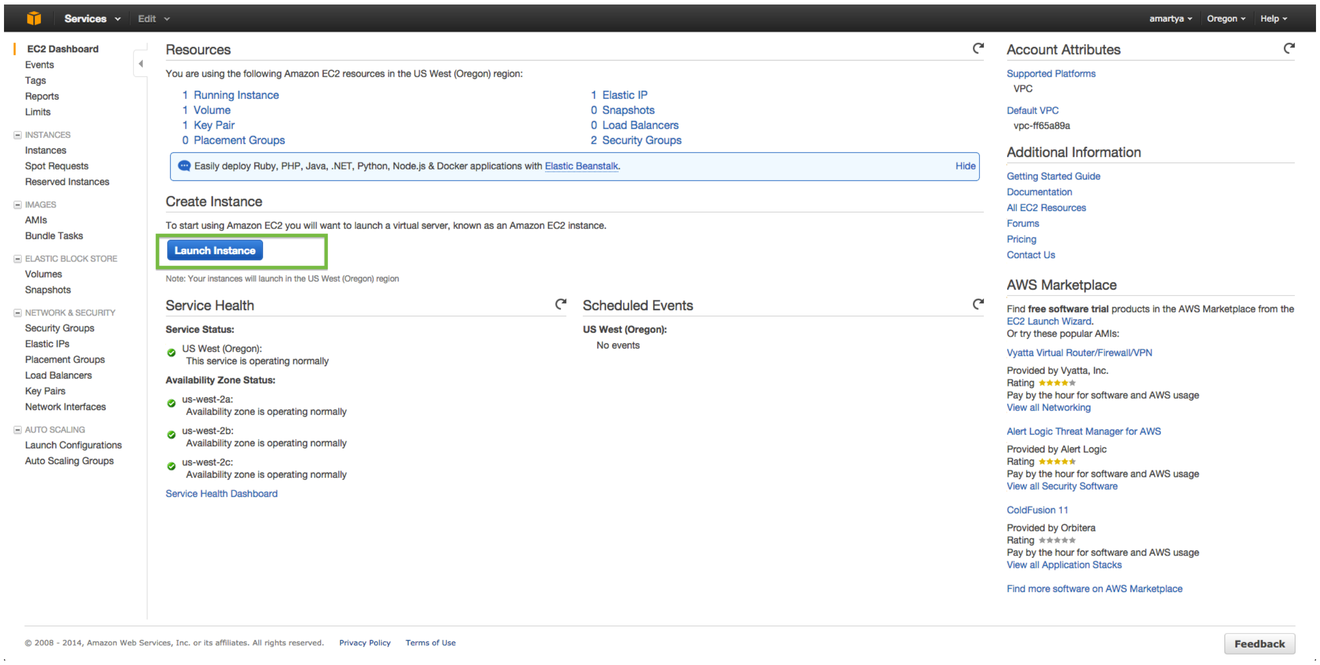
Service Health
All services operating normally.
Updated: Oct 07 2014 17:39:00 GMT-0500
Service Health Dashboard

Set Start Page
Console Home

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Feedback

After logging in to the AWS management console, select EC2

2. Follow the screenshots below to get an instance up and running.



EC2 Dashboard
Events
Tags
Reports
Limits

INSTANCES
Instances
Spot Requests
Reserved Instances

IMAGES
AMIs
Bundle Tasks

ELASTIC BLOCK STORE
Volumes
Snapshots

NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Load Balancers
Key Pairs
Network Interfaces

AUTO SCALING
Launch Configurations
Auto Scaling Groups

Resources

You are using the following Amazon EC2 resources in the US West (Oregon) region:

1 Running Instance	1 Elastic IP
1 Volume	0 Snapshots
1 Key Pair	0 Load Balancers
0 Placement Groups	2 Security Groups

Easily deploy Ruby, PHP, Java, .NET, Python, Node.js & Docker applications with Elastic Beanstalk.

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the US West (Oregon) region

Service Health

Service Status:
US West (Oregon):
This service is operating normally

Availability Zone Status:
us-west-2a:
Availability zone is operating normally
us-west-2b:
Availability zone is operating normally
us-west-2c:
Availability zone is operating normally

Service Health Dashboard

Scheduled Events

US West (Oregon):
No events

Account Attributes

Supported Platforms
VPC
Default VPC
vpc-ff65a89a

Additional Information

Getting Started Guide
Documentation
All EC2 Resources
Forums
Pricing
Contact Us

AWS Marketplace

Find **free software trial** products in the AWS Marketplace from the EC2 Launch Wizard.
Or try these popular AMIs:

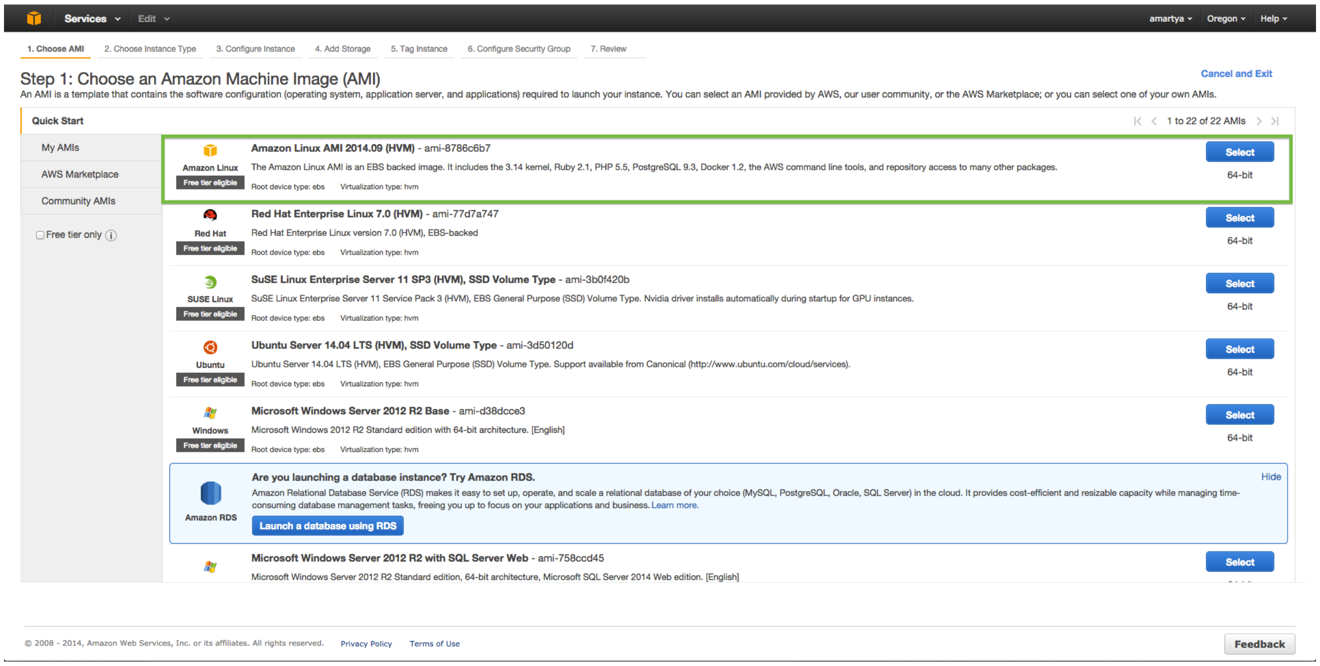
Vyatta Virtual Router/Firewall/VPN
Provided by Vyatta, Inc.
Rating ★★★★★
Pay by the hour for software and AWS usage
View all Networking

Alert Logic Threat Manager for AWS
Provided by Alert Logic
Rating ★★★★★
Pay by the hour for software and AWS usage
View all Security Software

ColdFusion 11
Provided by Orbitera
Rating ★★★★★
Pay by the hour for software and AWS usage
View all Application Stacks

Find more software on AWS Marketplace

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Amazon Linux — Free tier eligible
Amazon Linux AMI 2014.09 (HVM) - ami-8786c6b7
The Amazon Linux AMI is an EBS backed image. It includes the 3.14 kernel, Ruby 2.1, PHP 5.5, PostgreSQL 9.3, Docker 1.2, the AWS command line tools, and repository access to many other packages.
Root device type: ebs Virtualization type: hvm
64-bit

Red Hat — Free tier eligible
Red Hat Enterprise Linux 7.0 (HVM) - ami-77d7a747
Red Hat Enterprise Linux version 7.0 (HVM), EBS-backed
Root device type: ebs Virtualization type: hvm
64-bit

SUSE Linux — Free tier eligible
SuSE Linux Enterprise Server 11 SP3 (HVM), SSD Volume Type - ami-3b0f420b
SuSE Linux Enterprise Server 11 Service Pack 3 (HVM), EBS General Purpose (SSD) Volume Type. Nvidia driver installs automatically during startup for GPU instances.
Root device type: ebs Virtualization type: hvm
64-bit

Ubuntu — Free tier eligible
Ubuntu Server 14.04 LTS (HVM), SSD Volume Type - ami-3d50120d
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
64-bit

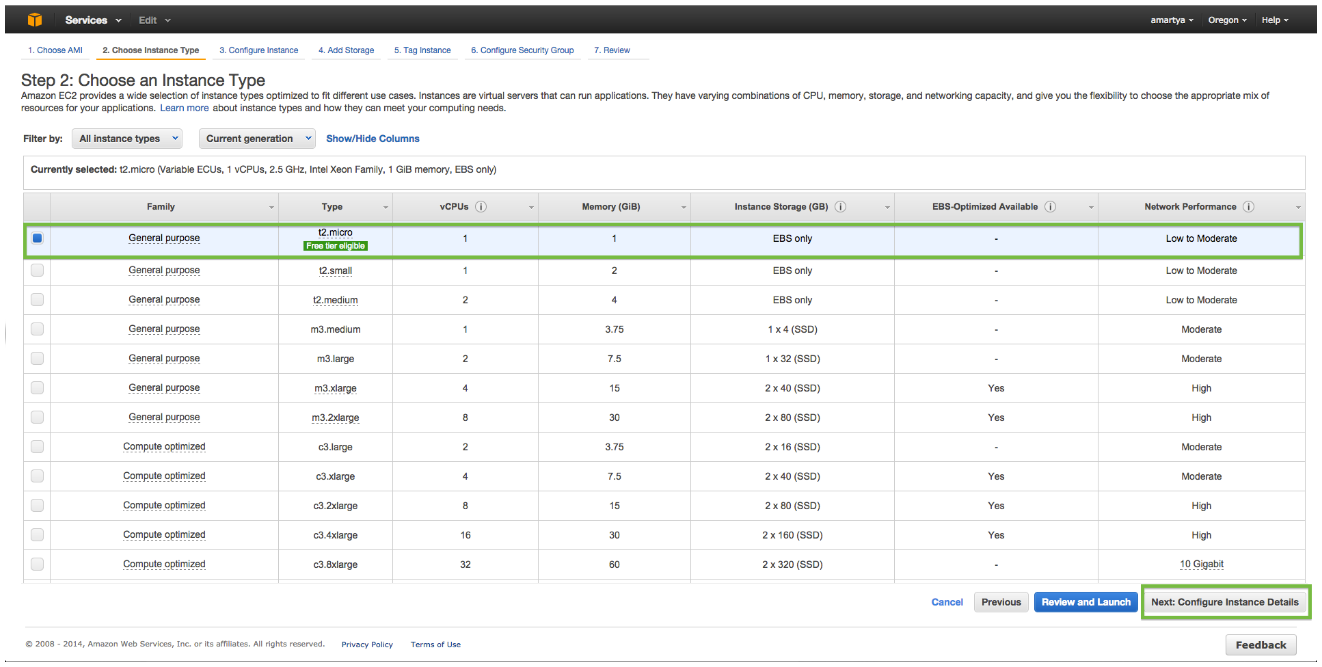
Windows — Free tier eligible
Microsoft Windows Server 2012 R2 Base - ami-d38dcce3
Microsoft Windows 2012 R2 Standard edition with 64-bit architecture. [English]
Root device type: ebs Virtualization type: hvm
64-bit

Amazon RDS
Are you launching a database instance? Try Amazon RDS.
Amazon Relational Database Service (RDS) makes it easy to set up, operate, and scale a relational database of your choice (MySQL, PostgreSQL, Oracle, SQL Server) in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database management tasks, freeing you up to focus on your applications and business. Learn more.

Microsoft Windows Server 2012 R2 with SQL Server Web - ami-758ccd45
Microsoft Windows Server 2012 R2 Standard edition, 64-bit architecture, Microsoft SQL Server 2014 Web edition. [English]

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

We'll go with the Amazon flavoured Linux distro because they include a lot of the stuff we need by default.



Step 2: Choose an Instance Type

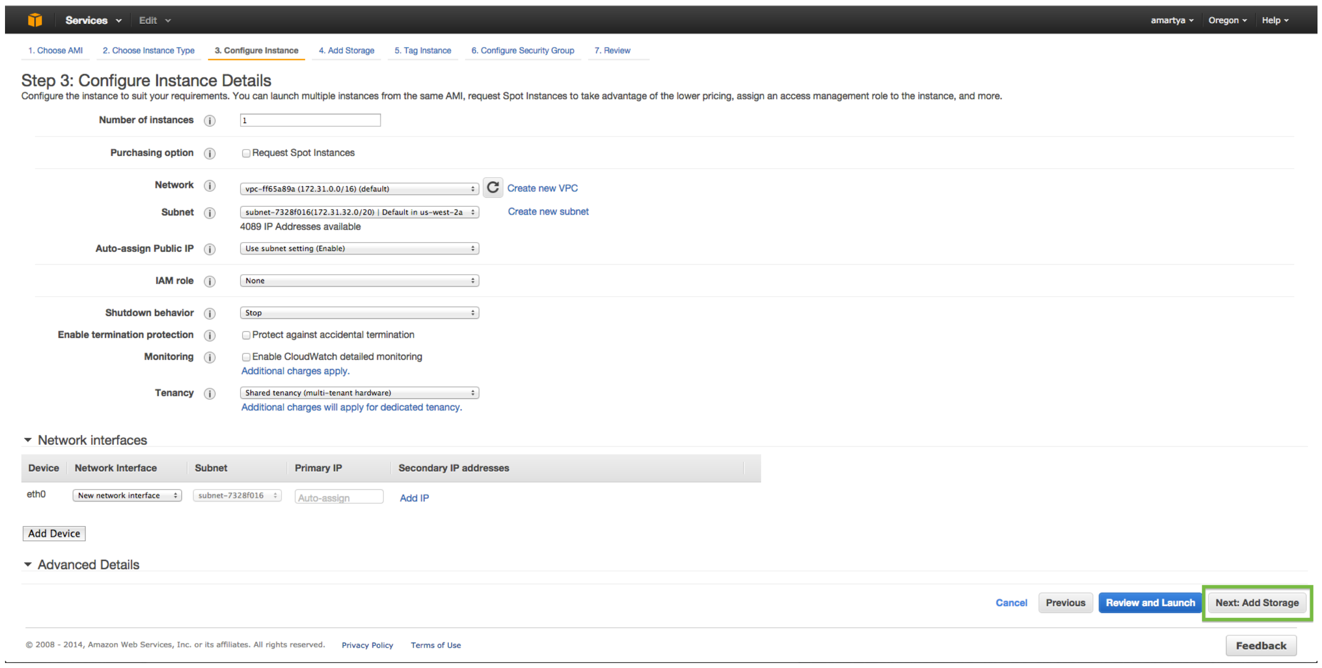
Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources 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)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
✓	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
	General purpose	m3.medium	1	3.75	1 x 4 (SSD)	-	Moderate
	General purpose	m3.large	2	7.5	1 x 32 (SSD)	-	Moderate
	General purpose	m3.xlarge	4	15	2 x 40 (SSD)	Yes	High
	General purpose	m3.2xlarge	8	30	2 x 80 (SSD)	Yes	High
	Compute optimized	c3.large	2	3.75	2 x 16 (SSD)	-	Moderate
	Compute optimized	c3.xlarge	4	7.5	2 x 40 (SSD)	Yes	Moderate
	Compute optimized	c3.2xlarge	8	15	2 x 80 (SSD)	Yes	High
	Compute optimized	c3.4xlarge	16	30	2 x 160 (SSD)	Yes	High
	Compute optimized	c3.8xlarge	32	60	2 x 320 (SSD)	-	10 Gigabit

Cancel
Previous
Review and Launch
Next: Configure Instance Details
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback



Step 3: Configure Instance Details

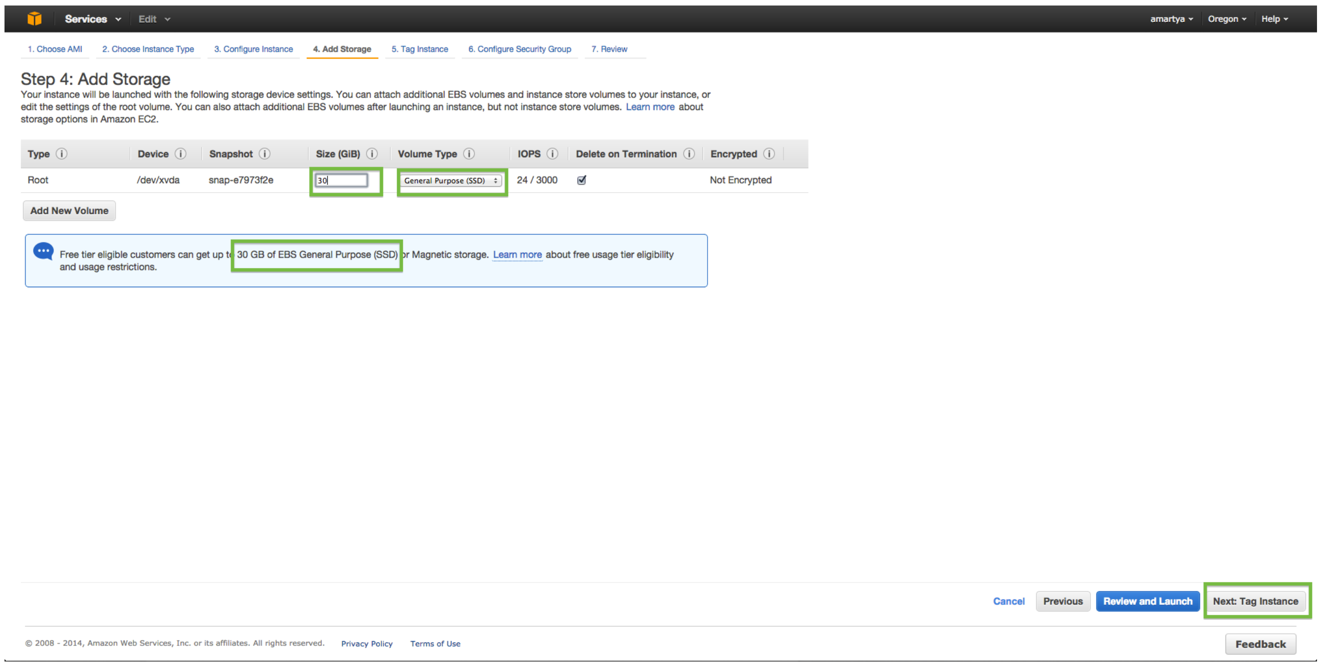
Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1
Purchasing option	☐ Request Spot Instances
Network	vpc-ff65a89a (172.31.0.0/16) (default) — Create new VPC
Subnet	subnet-7328f016(172.31.32.0/20) | Default in us-west-2a — Create new subnet 4089 IP Addresses available
Auto-assign Public IP	Use subnet setting (Enable)
IAM role	None
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-7328f016	Auto-assign	Add IP

Nothing to do on this particular screen, move on to the next screen (“Add Storage”).



Step 4: Add Storage

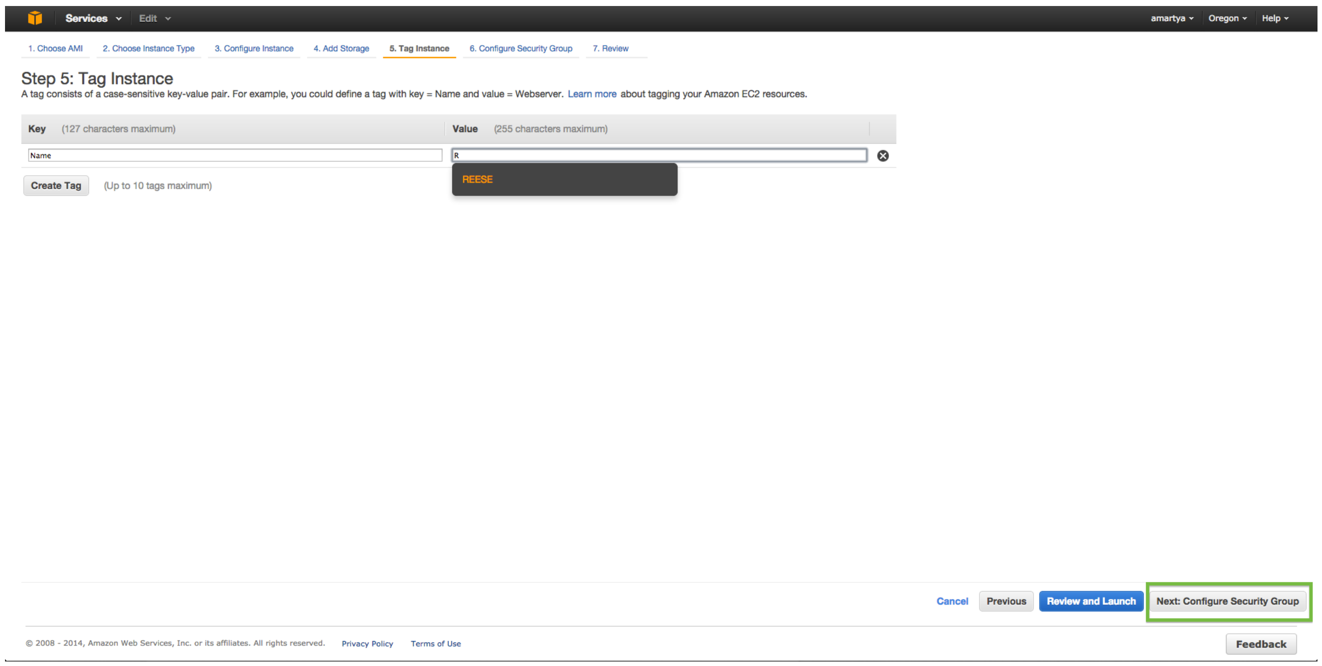
Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/xvda	snap-e7973f2e	30	General Purpose (SSD)	24 / 3000	✓	Not Encrypted

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.

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Maxing out the default storage for now. Amazon has something called RDS that allows us to attach resizable storage (you can see the option in the very first screen where we chose EC-2). However, it seems to be an overkill for our purposes and it seems prudent to have as much storage assigned to EC2 initially as possible.



Step 5: Tag Instance

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.

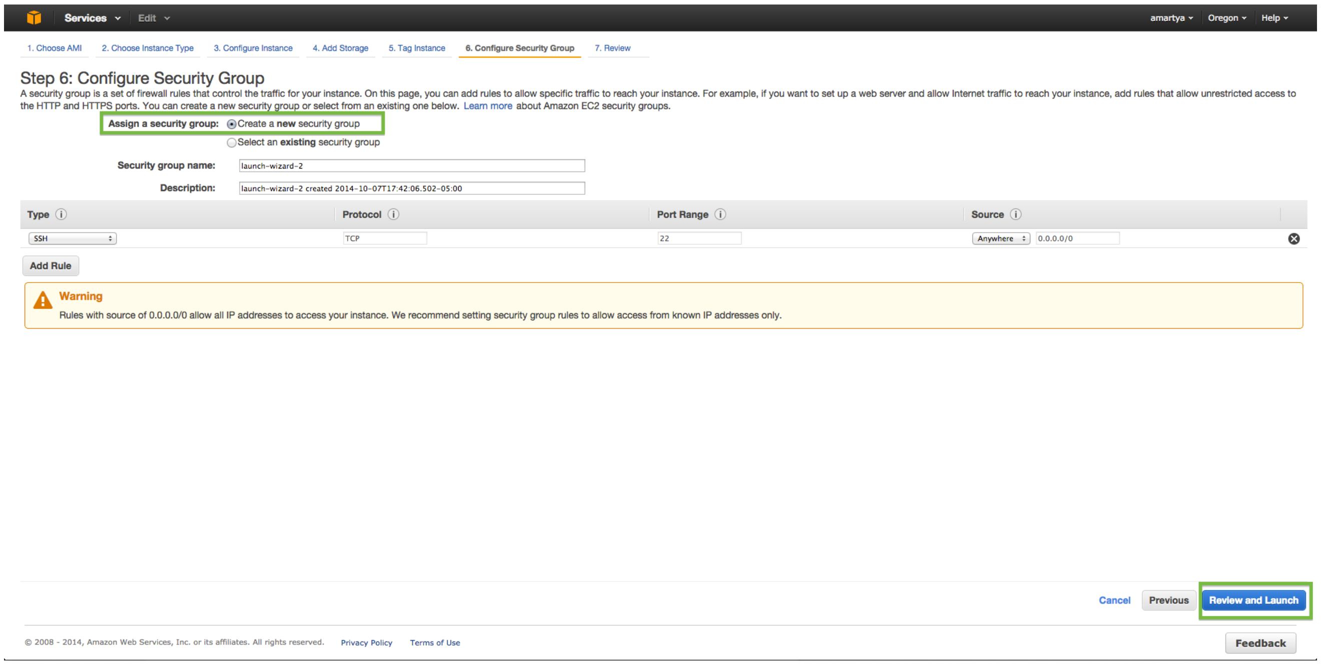
Key (127 characters maximum)	Value (255 characters maximum)
Name	R

REESE

(Up to 10 tags maximum)

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Give this instance a Name key to make it easier to track in the EC2 management console.



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: launch-wizard-2

Description: launch-wizard-2 created 2014-10-07T17:42:06.502-05:00

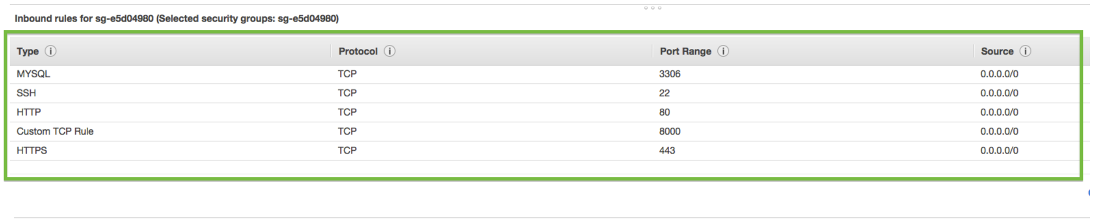
Type	Protocol	Port Range	Source
SSH	TCP	22	Anywhere 0.0.0.0/0

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.

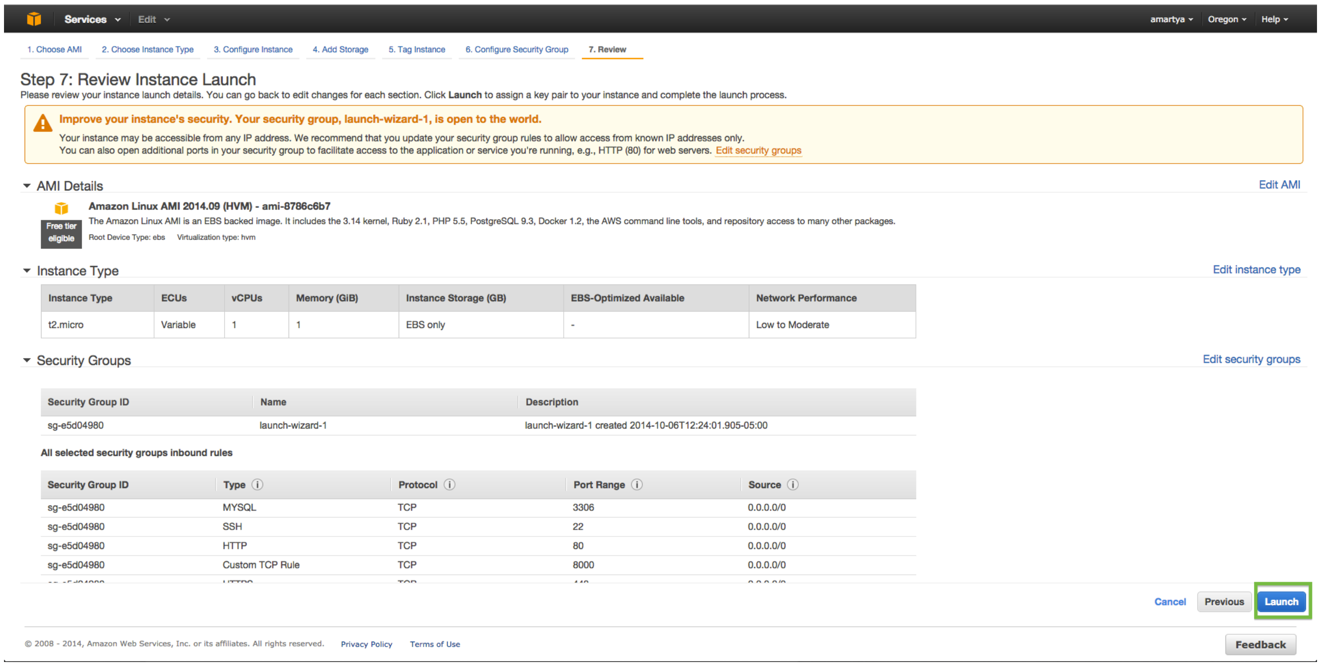
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

The security group setting is IMPORTANT. Copy the configuration shown below. This basically says what kind of traffic/ports should the server allow. If you already have a security group and wish to reuse it, just add all these rules.



Inbound rules for sg-e5d04980 (Selected security groups: sg-e5d04980)

Type	Protocol	Port Range	Source
MYSQL	TCP	3306	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0
Custom TCP Rule	TCP	8000	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0



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 instance's security. Your security group, launch-wizard-1, is open to the world.

Your instance 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

Amazon Linux AMI 2014.09 (HVM) - ami-8786c6b7

Free tier eligible

The Amazon Linux AMI is an EBS backed image. It includes the 3.14 kernel, Ruby 2.1, PHP 5.5, PostgreSQL 9.3, Docker 1.2, the AWS command line tools, and repository access to many other packages.

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

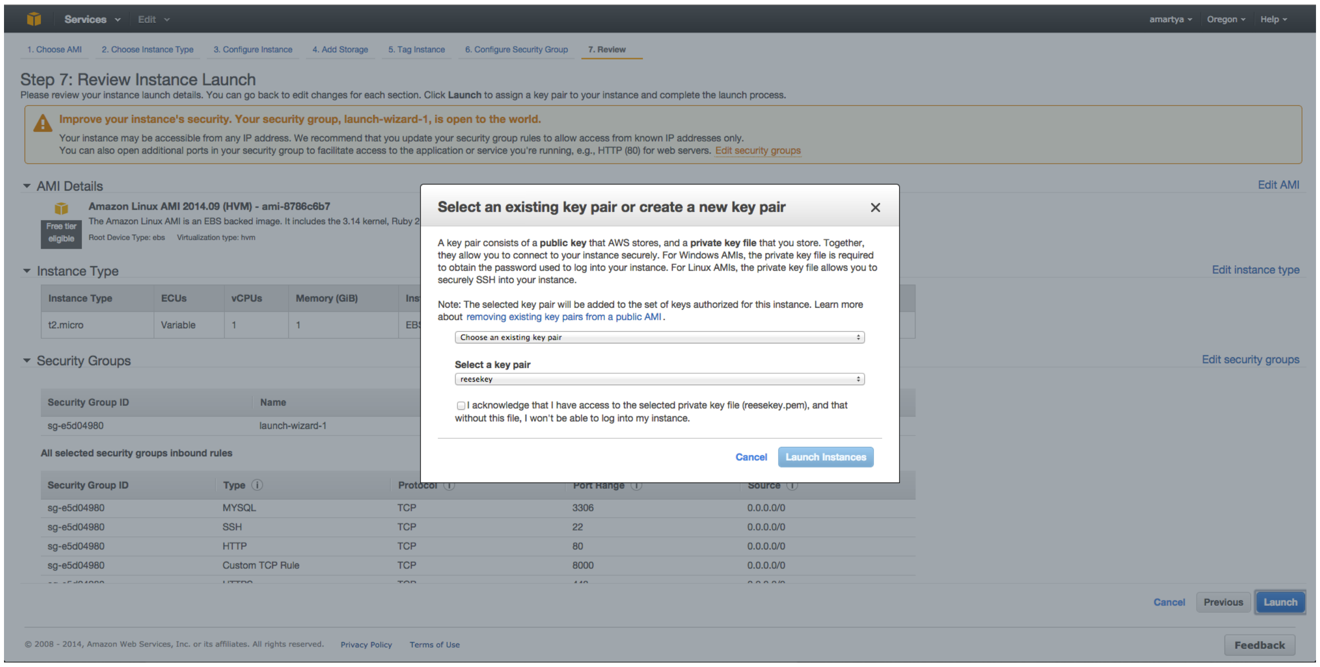
Security Groups — Edit security groups

Security Group ID	Name	Description
sg-e5d04980	launch-wizard-1	launch-wizard-1 created 2014-10-06T12:24:01.905-05:00

All selected security groups inbound rules

Security Group ID	Type	Protocol	Port Range	Source
sg-e5d04980	MYSQL	TCP	3306	0.0.0.0/0
sg-e5d04980	SSH	TCP	22	0.0.0.0/0
sg-e5d04980	HTTP	TCP	80	0.0.0.0/0
sg-e5d04980	Custom TCP Rule	TCP	8000	0.0.0.0/0

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use



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 instance's security. Your security group, launch-wizard-1, is open to the world.

Your instance 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

Amazon Linux AMI 2014.09 (HVM) - ami-8786c6b7
Free tier eligible
The Amazon Linux AMI is an EBS backed image. It includes the 3.14 kernel, Ruby 2
Root Device Type: ebs   Virtualization type: hvm

Instance Type — Edit instance type

Instance Type	ECUs	vCPUs	Memory (GiB)	Ins
t2.micro	Variable	1	1	EB

Security Groups — Edit security groups

Security Group ID	Name
sg-e5d04980	launch-wizard-1

All selected security groups inbound rules

Security Group ID	Type	Protocol	Port Range	Source
sg-e5d04980	MYSQL	TCP	3306	0.0.0.0/0
sg-e5d04980	SSH	TCP	22	0.0.0.0/0
sg-e5d04980	HTTP	TCP	80	0.0.0.0/0
sg-e5d04980	Custom TCP Rule	TCP	8000	0.0.0.0/0

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Choose an existing key pair

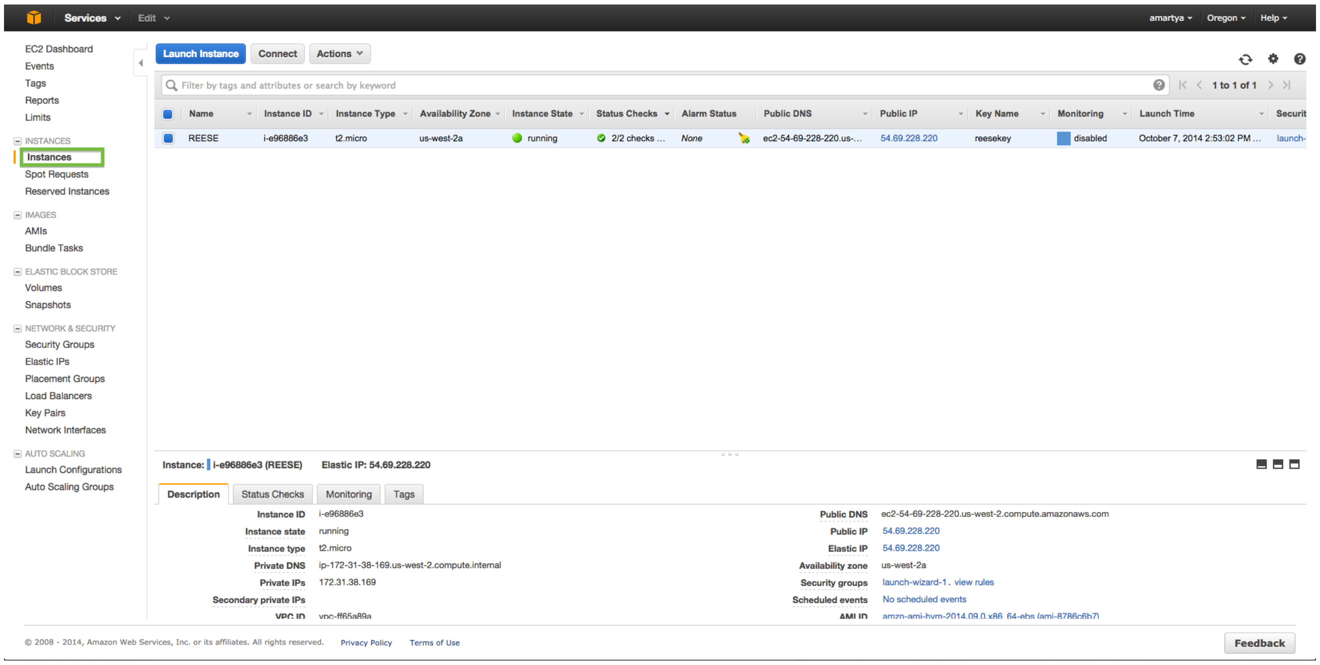
Select a key pair: reesekey

☐ I acknowledge that I have access to the selected private key file (reesekey.pem), and that without this file, I won't be able to log into my instance.

Cancel
Launch Instances
Cancel
Previous
Launch
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

I already had a key setup. However, you can create a new key by selecting "Create a new Key Pair" and giving it a name. Download the key (I have it in a folder on Dropbox). We'll have to edit the folder and permissions for this key.

We are almost done, just Launch Instance and navigate to the "Instances" section under the EC2 management console. You should see something similar to the screenshot below.



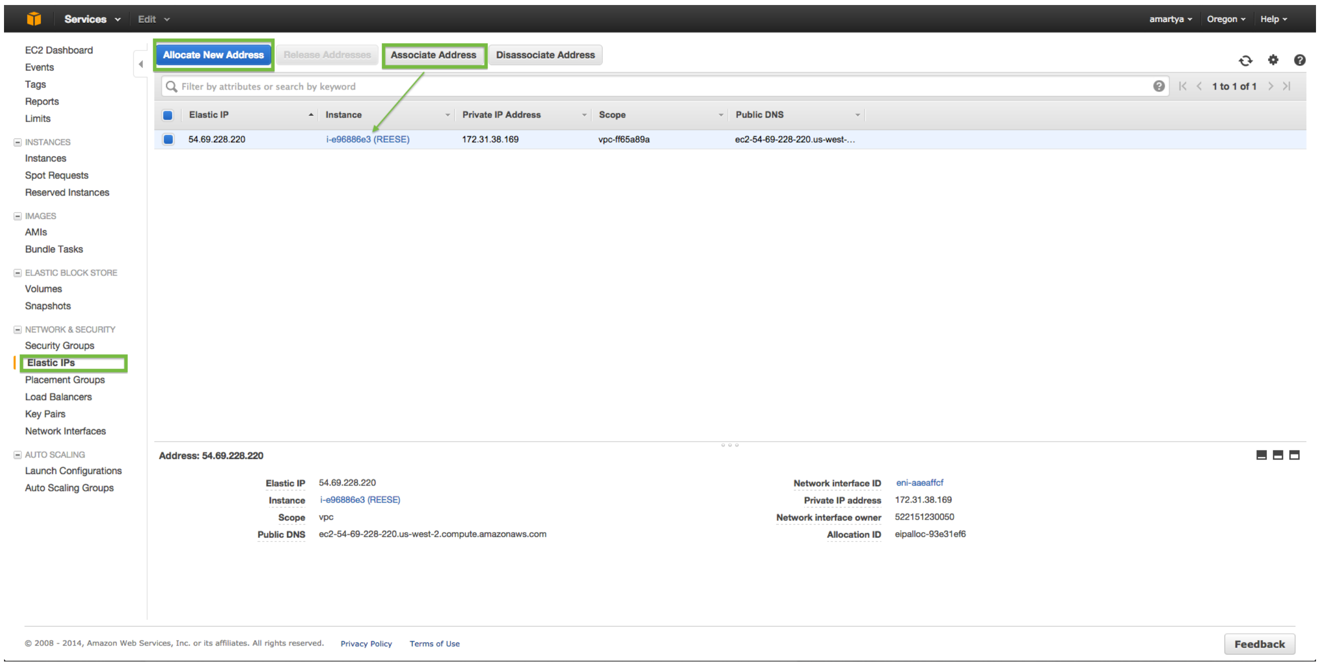
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS	Public IP	Key Name	Monitoring	Launch Time	Securit
REESE	i-e96886e3	t2.micro	us-west-2a	● running	✓ 2/2 checks ...	None	ec2-54-69-228-220.us-...	54.69.228.220	reesekey	disabled	October 7, 2014 2:53:02 PM ...	launch-

Instance: i-e96886e3 (REESE)   Elastic IP: 54.69.228.220

Instance ID	i-e96886e3	Public DNS	ec2-54-69-228-220.us-west-2.compute.amazonaws.com
Instance state	running	Public IP	54.69.228.220
Instance type	t2.micro	Elastic IP	54.69.228.220
Private DNS	ip-172-31-38-169.us-west-2.compute.internal	Availability zone	us-west-2a
Private IPs	172.31.38.169	Security groups	launch-wizard-1. view rules
Secondary private IPs		Scheduled events	No scheduled events
VPC ID	vpc-ff65a89a	AMI ID	amzn-ami-hvm-2014.09.0.x86_64-ebs (ami-8786c6b7)

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

AWS assigns us a public IP and a DNS. However, these are for a machine in the cloud that might be restarted. Which implies that this IP could randomly change. To fix that, we need to create an "Elastic IP" and "Associate" it with the EC2 instance that we just created.



Elastic IP	Instance	Private IP Address	Scope	Public DNS
54.69.228.220	i-e96886e3 (REESE)	172.31.38.169	vpc-ff65a89a	ec2-54-69-228-220.us-west-...

Address: 54.69.228.220

Elastic IP	54.69.228.220	Network interface ID	eni-aaeaffcf
Instance	i-e96886e3 (REESE)	Private IP address	172.31.38.169
Scope	vpc	Network interface owner	522151230050
Public DNS	ec2-54-69-228-220.us-west-2.compute.amazonaws.com	Allocation ID	eipalloc-93e31ef6

© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.

That's it! Once we have an elastic IP pointing to our instance, a server restart on Amazon's end would no longer result in our work being lost in the ether :).

Moving on to setting up the setup that includes apache, django etc. from the next page.

Installing Apache and Mod_Wsgi

1. First, we have to login to our EC2 instance that we set up. Fire up Terminal on the Mac and navigate to the folder containing the key you downloaded (the *.pem file). You can copy paste the commands from the text boxes that follow:

```
#Login
#Can use the domain name (ec2-aws....) instead of the I.P. too
chmod 400 reesekey.pem (change file permissions to be able to add the reesekey.pem as an ssh key)
ssh-add reesekey.pem
cd path_to_ #keypair location
ssh -i reesekey.pem ec2-user@54.69.228.220 #use elastic IP.

#if it throws a warning saying that the "remote host identification has changed", run "ssh-keygen -R
54.69.228.220"
#and then follow if up with the "ssh -i reesekey.pem ec2-user@54.69.228.220" command
```

2. Once we are logged in, try using the command “pwd” (present working directory). It should say something like “/home/ec2-user”. Now run “python -V”. Currently EC2 comes with Python 2.6, we’ll start by updating it to Python 2.7 for future Django versions. We also install Pip, a fantastic python package manager :).

```
# install build tools
sudo yum install make automake gcc gcc-c++ kernel-devel git-core -y

# install python 2.7 and change default python symlink
sudo yum install python27-devel -y
sudo rm /usr/bin/python
sudo ln -s /usr/bin/python2.7 /usr/bin/python

# yum still needs 2.6, so write it in and backup script
sudo cp /usr/bin/yum /usr/bin/_yum_before_27
sudo sed -i s/python/python2.6/g /usr/bin/yum
sudo sed -i s/python2.6/python2.6/g /usr/bin/yum

# should now display 2.7.5 or later instead of 2.6xx:
python -V

# now install pip for 2.7
#Just to keep things neat, create a downloads directory in ec2-user
# for all application packages (rpm, tar etc.)
cd /home/ec2-user
mkdir downloads
cd downloads
wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py

# should display current version of pip:
pip -V
```

3. Install apache

```
#Apache
sudo yum install httpd #installs apache as a service

sudo service httpd start
#starts apache, test by going to http://<your elastic IP> here it was http://54.69.228.220
sudo service httpd stop #stops apache
sudo service httpd restart #restarts apache
sudo chkconfig httpd on #starts the Apache service whenever the server reboots
```

You should see the Apache default welcome screen.

4. Just having Apache is not going to work when we try to serve a Django view. We need another interface called mod_wsgi that acts as a conduit. While mod_wsgi comes with EC2 as defaultt, we'll have to do some grunt work to get mod_wsgi to work with the updated Python 2.7 that we got going (the original version that came with the EC2 instance was compiled for Python 2.6).

```
#Mod-Wsgi
#wsgi = Web Server Gateway Interface, a python spec. for server to application communication
# install development apache tools
yum install httpd-devel.x86_64

#downloads and compiles mod_wsgi to be used for python2.7, I recommend using the downloads folder
# you created while installing pip (step-2) before you run the wget
wget http://modwsgi.googlecode.com/files/mod_wsgi-3.4.tar.gz
tar -zvxf mod_wsgi-3.4.tar.gz
cd mod_wsgi-3.4
./configure --with-python=/usr/bin/python2.7
make
sudo make install
```

Installing MySQL

Next, we'll install MySQL. After installing MySQL, we need to ensure that we install the python driver for it, create a database along with a user (root) and password. The commands are all listed below:

```
#MySQL
sudo yum install mysql mysql-server MySQL-python mysql-devel
sudo pip install MySQL-python #python driver for MySQL
service mysqld start #start MySql server
/usr/bin/mysqladmin -u root password 'R33535' # Set password for MySQL root user'
mysql -u root -p -e 'CREATE DATABASE reese' (if it asks asks for pwd: R33535)
#Enter python shell and "import MySQLdb" to test installation
#if something went wrong and you wish to reinstall mysql
#sudo yum -y remove mysql-server
```

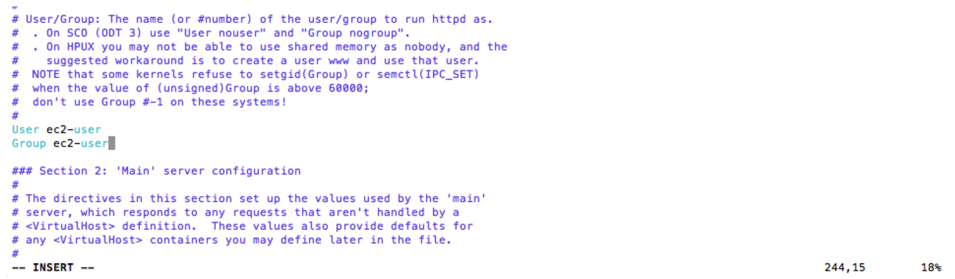
Installing and configuring Django App

In this part of the guide, we'll install Django, create a basic Django app and then serve it via the mod_wsgi and Apache layers we'd added in the previous step.
Once you've installed the app, we need to configure our server to serve this Django app. Go to the next page for the apache mod_wsgi settings.

```
#Django
sudo pip install django==1.6

#If this fails, get the latest package from the django servers and follow the steps below
wget https://www.djangoproject.com/download/1.7/tarball/ #download Django
mv index.html Django-1.7.tar.gz #renaming file manually, giving a file name didn't work in the prev.
step
tar xvf Django-1.7.tar.gz #extract tar archive, x= get, v= verbose, w= interactive(not used here),
f=file/hostname
cd Django-1.7
python setup.py install #installs django
#Enter python shell and "import django" to test installation

#create basic Django project in a directory of choice, I created it in my /home/ec2-user directory
django-admin.py startproject reese

#If you go inside the reese directory, you'd notice that Django has created another directory called
#reese inside 'this' reese directory. That's normal :). In the outer reese directory, you should see
#a file called manage.py at the same level as another "reese" directory
```

Here's a slightly tricky part. It's tricky only because we need to pay extra attention to the file paths that we lay out. Bear with me :), it isn't as bad as it might seem. To start off, we need to access/edit Apache's config file (httpd.conf).

```
#Configure apache to serve from Django app
#Edit the httpd.conf file in /etc/httpd/conf/
sudo vim /etc/httpd/conf/httpd.conf

# Add to the end of the opened file. tip: use ctrl+d to scroll down & ctrl+u to scroll up
WSGIPythonHome /usr
WSGIPythonPath /home/ec2-user/reese:/usr/lib64/python2.7/site-packages:/usr/lib64/python2.7
<VirtualHost *:80>
        DocumentRoot /home/ec2-user/reese
        ErrorLog /home/ec2-user/reese/logs/apache_error.log
        CustomLog /home/ec2-user/reese/logs/apache_access.log combined
        WSGIScriptAlias / /home/ec2-user/reese/apache/django.wsgi
        Alias /static/ /home/ec2-user/reese/static/

        <Directory /home/ec2-user/reese/media>
                Order deny,allow
                Allow from all
        </Directory>

        <Directory /home/ec2-user/reese/apache>
                Order deny,allow
                Allow from all
        </Directory>

        LogLevel warn
</VirtualHost>

#THEN find a couple of lines that say User and Group in the same file, mine were line numbers 243
#and 244 and change them to ec2-user (screenshot below)
```



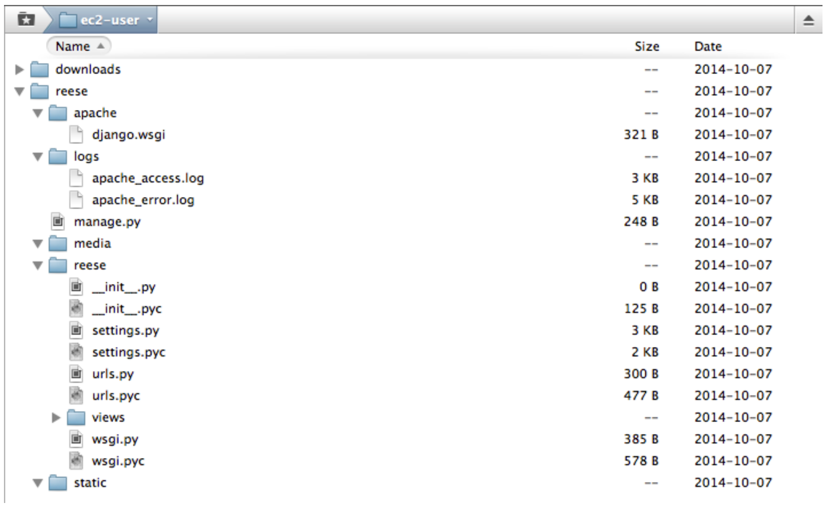
```
#
# User/Group: The name (or #number) of the user/group to run httpd as.
#  . On SCO (ODT 3) use "User nouser" and "Group nogroup".
#  . On HPUX you may not be able to use shared memory as nobody, and the
#    suggested workaround is to create a user www and use that user.
#  NOTE that some kernels refuse to setgid(Group) or semctl(IPC_SET)
#  when the value of (unsigned)Group is above 60000;
#  don't use Group #-1 on these systems!
#
User ec2-user
Group ec2-user

### Section 2: 'Main' server configuration
#
# The directives in this section set up the values used by the 'main'
# server, which responds to any requests that aren't handled by a
# <VirtualHost> definition.  These values also provide defaults for
# any <VirtualHost> containers you may define later in the file.
#
-- INSERT --                                                  244,15        18%
```

```
#save and exit this file (:wq is the Vim command)
#Run apachectl configtest  to ensure that the file is syntactically correct
#If running configtest results in an error like "Invalid command 'WSGIPythonHome'…", we need to
#import the mod_wsgi module for apache.
#Go to the httpd folder, inside it you'll find a directory called "conf.d". Inside conf.d, create a
#file called wsgi.conf (if it exists, edit it)
cd /etc/httpd/conf.d
sudo vim wsgi.conf

Add the line "LoadModule wsgi_module modules/mod_wsgi.so" to the wsgi.conf file and save it.
now running apachectl configtest should say "Syntax OK"


#We've done a few things here. We've told apache that it should redirect all HTTP traffic (port 80)
#to the /home/ec2-user/reese directory. You see a reference to a django.wsgi file, folders such as
#media, static, apache and logs. These directories don't exist right now, we'll create them :).

# Go to the parent reese folder (/home/ec2-user/reese), create 2 folders, logs and apache
# In the apache folder, create a django.wsgi file (note the WSGIScriptAlias variable in the httpd
conf. file above). We'll fill it in later :).

# Similarly, create blank log files called apache_error.log and apache_access.log in the logs folder
# Just ensure that the paths make sense w.r.t. the configuration shown above (httpd.conf)
```

Now, in the ***django.wsgi*** file that you create in the previous page, add the following lines of code

```
import os,sys

apache_configuration = os.path.dirname(__file__)
project = os.path.dirname(apache_configuration)
workspace = os.path.dirname(project)
sys.path.append(workspace)

os.environ['DJANGO_SETTINGS_MODULE'] = 'reese.settings'

#For DJANGO 1.6, use these two lines
import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()

#IF YOU ARE USING DJANGO 1.7, replace the last two lines with this
#from django.core.wsgi import get_wsgi_application
#application = get_wsgi_application()
```

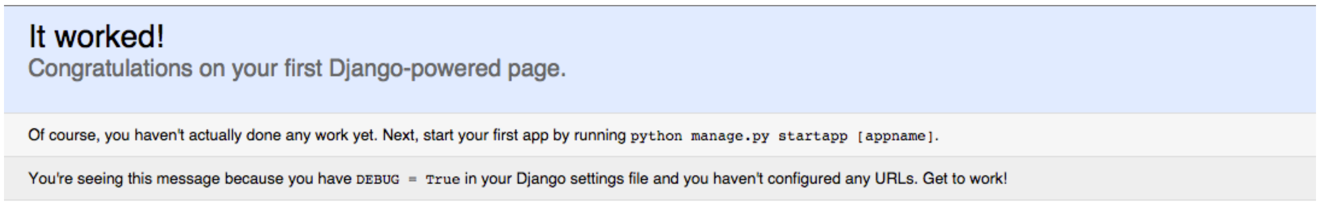
Basically, your directory structure should be similar to the screenshot below (screenshot from FTP client):



```
ec2-user
Name                                   Size    Date
▶ downloads                            --      2014-10-07
▼ reese                                --      2014-10-07
  ▼ apache                             --      2014-10-07
      django.wsgi                      321 B   2014-10-07
  ▼ logs                               --      2014-10-07
      apache_access.log                3 KB    2014-10-07
      apache_error.log                 5 KB    2014-10-07
    manage.py                          248 B   2014-10-07
  ▼ media                              --      2014-10-07
  ▼ reese                              --      2014-10-07
      __init__.py                      0 B     2014-10-07
      __init__.pyc                     125 B   2014-10-07
      settings.py                      3 KB    2014-10-07
      settings.pyc                     2 KB    2014-10-07
      urls.py                          300 B   2014-10-07
      urls.pyc                         477 B   2014-10-07
    ▶ views                            --      2014-10-07
      wsgi.py                          385 B   2014-10-07
      wsgi.pyc                         578 B   2014-10-07
  ▼ static                             --      2014-10-07
```

Don't worry if you don't have a directory called views in there, that's something I created later on. After the httpd.conf file has been edited and saved, restart the server (***sudo service httpd restart***).

With a all fingers crossed and a prayer, go to http:<your elastic I.P.> (in this case http://54.69.228.220/) and you'd should see this :)



It worked!

Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [appname]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

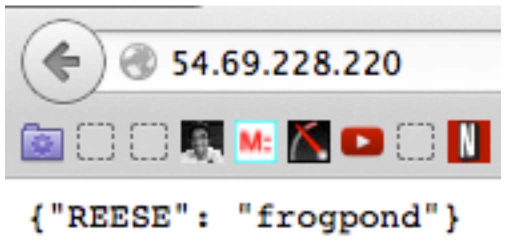
So right now Django is running in Debug mode. We need to turn off debug mode and setup Django to use the MySQL database we had set up. To do that, we need to edit the settings.py file

```
vim /home/ec2-user/reese/reese/settings.py

#find the DATABASE variable and change it from the default sqllite to MySQL

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', ‘mysql', or 'sqlite3'
        'NAME': 'reese',
        'USER': 'root',
        'PASSWORD': 'R33535',
        'HOST': '',
        'PORT': '', # Set to empty string for default.
        'OPTIONS': {
        "init_command": "SET foreign_key_checks = 0;",
        },
    }
}

#then find the ALLOWED_HOSTS variable and change it to:
ALLOWED_HOSTS = ['*']

#finally, find the variable called DEBUG and set it to false
DEBUG = False
```

To test if things are still working, create a view. I prefer having a folder that contains all my views. So I created a “views” folder in the same directory that contains the settings.py file. Make sure you create a file (empty file) called ***__init.py__*** inside this views folder, it tells django to look inside this directory for files. I created a view called “home” with a function called index that returns a json.

```
#This is the home.py file

from django.template import RequestContext,Context
from django.core.urlresolvers import reverse
from django.http import HttpResponse
from django.http import HttpResponseRedirect
from django.shortcuts import redirect
from django.shortcuts import render
from django.views.decorators.csrf import csrf_protect
from django.shortcuts import render_to_response
from django.conf import settings

import json

def index(request):
    return HttpResponse(json.dumps({'REESE' : 'frogpond'}), content_type="application/json")


#then edit the urls.py to something like

from django.conf.urls import patterns, include, url
from django.contrib import admin

admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^$', 'reese.views.home.index', name=‘home’),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
)
```

Now, going to http://<your elastic I.P>, i.e. http://54.69.228.220/ in this case, should result in the following page:



```
{"REESE": "frogpond"}
```

Success!

Next: We look at how to setup Django to enable us to post data without running into cross-origin resource issues (e.g.when posting data from a standalone app to our end-point/server)

Getting Django and Dart to play nice

We want to be able to POST data to our server while getting around cross-site security issues that most browsers check against. Broadly, we'll do the following:

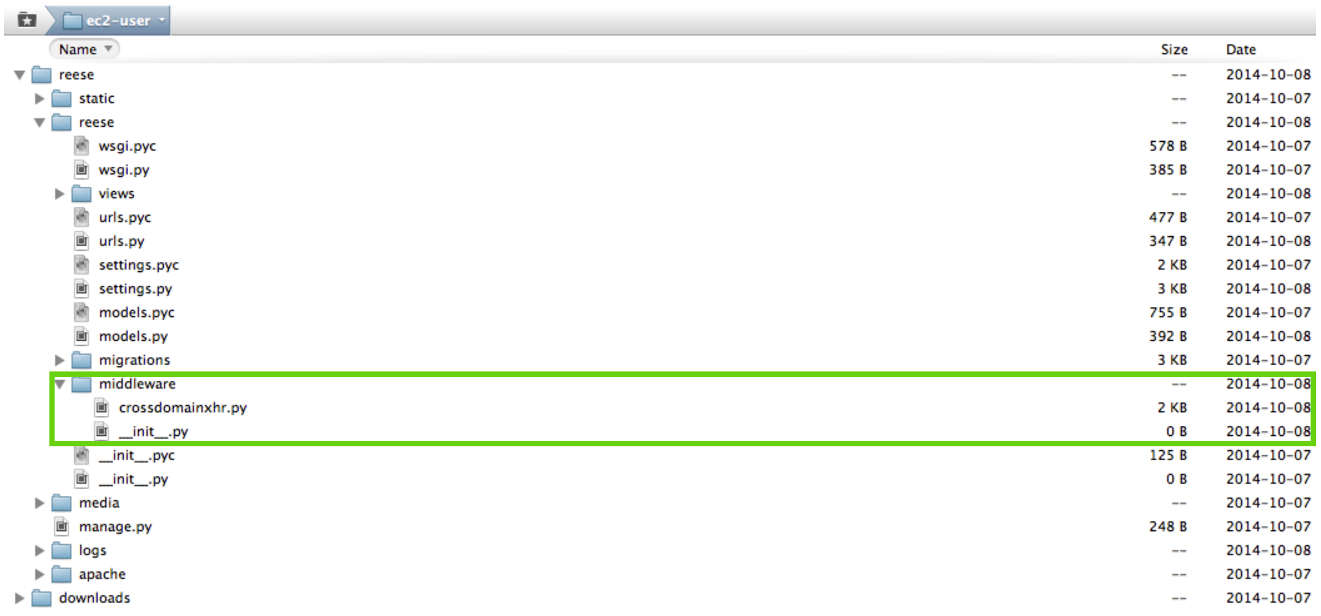
a. Setup Django according to the specs laid out in CORS (Cross-Origin Resource Sharing).
b. Create and POST a json request from Dart.
c. Look at Django's csrf (cross-site request forgery) security issue

Django Setup for CORS

While we could have manually set this up according to the CORS specs in each view, since Django supports plugins/middleware, it is neater to add CORS relevant code as a middleware.

In your Django app, create a folder called "middleware". Inside it create 2 files, __init.py__ (empty file that tells Django that this directory is a part of your project) and crossdomainxhr.py

This is what the directory structure should look like



```
ec2-user
Name                                   Size    Date
▼ reese                                 --     2014-10-08
  ▶ static                              --     2014-10-07
  ▼ reese                               --     2014-10-08
      wsgi.pyc                         578 B   2014-10-07
      wsgi.py                          385 B   2014-10-07
    ▶ views                             --     2014-10-08
      urls.pyc                         477 B   2014-10-07
      urls.py                          347 B   2014-10-08
      settings.pyc                      2 KB   2014-10-07
      settings.py                       3 KB   2014-10-08
      models.pyc                       755 B   2014-10-07
      models.py                        392 B   2014-10-08
    ▶ migrations                        3 KB   2014-10-07
    ▼ middleware                        --     2014-10-08
        crossdomainxhr.py               2 KB   2014-10-08
        __init__.py                     0 B    2014-10-08
      __init__.pyc                     125 B   2014-10-07
      __init__.py                       0 B    2014-10-07
  ▶ media                               --     2014-10-07
    manage.py                          248 B   2014-10-07
  ▶ logs                                --     2014-10-08
  ▶ apache                              --     2014-10-07
▶ downloads                             --     2014-10-07
```

Inside the crossdomainxhr.py file, paste the following code

```
from django import http

try:
    from django.conf import settings
    XS_SHARING_ALLOWED_ORIGINS = settings.XS_SHARING_ALLOWED_ORIGINS
    XS_SHARING_ALLOWED_METHODS = settings.XS_SHARING_ALLOWED_METHODS
    XS_SHARING_ALLOWED_HEADERS = settings.XS_SHARING_ALLOWED_HEADERS
    XS_SHARING_ALLOWED_CREDENTIALS = settings.XS_SHARING_ALLOWED_CREDENTIALS
except AttributeError:
    XS_SHARING_ALLOWED_ORIGINS = '*'
    XS_SHARING_ALLOWED_METHODS = ['POST', 'GET', 'OPTIONS', 'PUT', 'DELETE']
    XS_SHARING_ALLOWED_HEADERS = ['Content-Type', '*']
    XS_SHARING_ALLOWED_CREDENTIALS = 'true'


class XsSharing(object):
    """
    This middleware allows cross-domain XHR using the html5 postMessage API.

    Access-Control-Allow-Origin: http://foo.example
    Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE

    Based off https://gist.github.com/426829
    """
    def process_request(self, request):
        if 'HTTP_ACCESS_CONTROL_REQUEST_METHOD' in request.META:
            response = http.HttpResponse()
            response['Access-Control-Allow-Origin']  = XS_SHARING_ALLOWED_ORIGINS
            response['Access-Control-Allow-Methods'] = ",".join( XS_SHARING_ALLOWED_METHODS )
            response['Access-Control-Allow-Headers'] = ",".join( XS_SHARING_ALLOWED_HEADERS )
            response['Access-Control-Allow-Credentials'] = XS_SHARING_ALLOWED_CREDENTIALS
            return response

        return None

    def process_response(self, request, response):
        response['Access-Control-Allow-Origin']  = XS_SHARING_ALLOWED_ORIGINS
        response['Access-Control-Allow-Methods'] = ",".join( XS_SHARING_ALLOWED_METHODS )
        response['Access-Control-Allow-Headers'] = ",".join( XS_SHARING_ALLOWED_HEADERS )
        response['Access-Control-Allow-Credentials'] = XS_SHARING_ALLOWED_CREDENTIALS

        return response
```

Now, we need to add this to our settings file (settings.py) and assign a couple of variables:

```
#Cross-site sharing related vars
XS_SHARING_ALLOWED_ORIGINS = '*'
XS_SHARING_ALLOWED_METHODS = ['POST','GET','OPTIONS']
XS_SHARING_ALLOWED_HEADERS = ['Content-Type', '*']

.
.
.
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    ‘reese.middleware.crossdomainxhr.XsSharing', # JUST ADD THIS to MIDDLEWARE_CLASSES
)
```

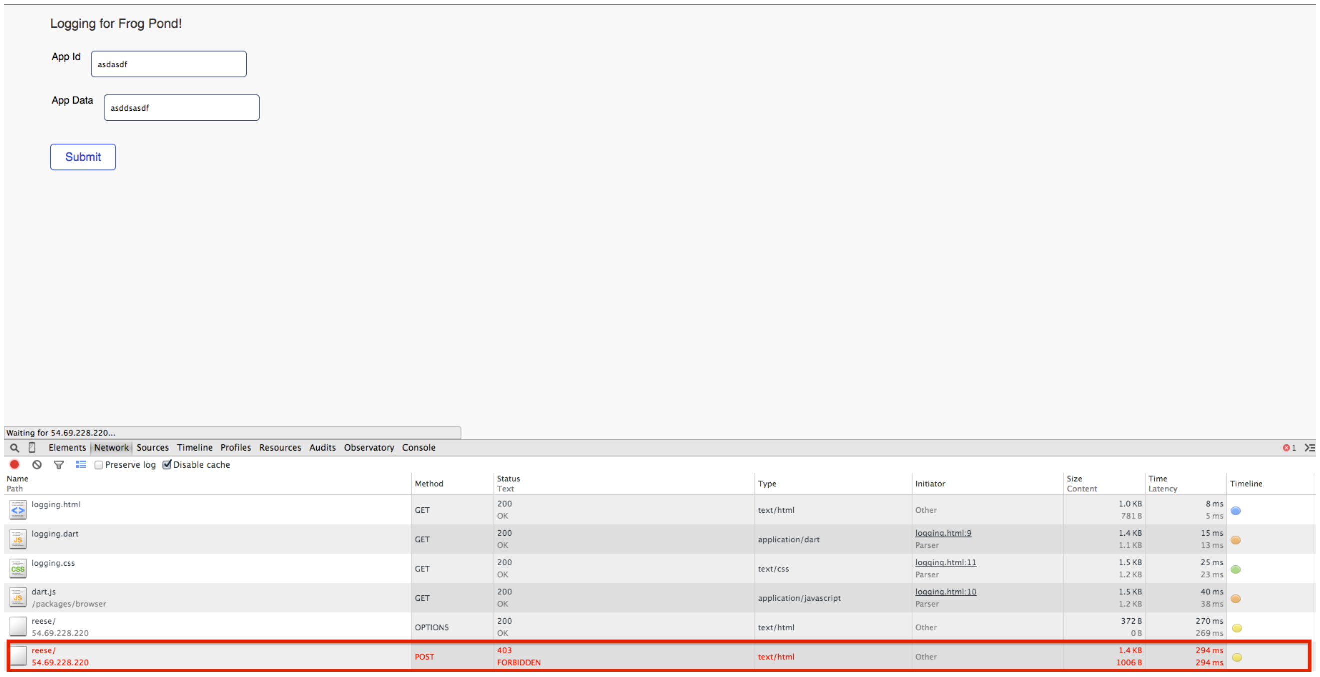
Basically, we just told our Django app to accept cross-origin requests of type POST/GET/OPTIONS/ etc. Restart you server for good measure even though it's not mandatory ("sudo service https restart")

Dart Code

We can post from any machine. Notice the url we are posting to, it has to be a valid url defined in urls.py in Django.

```
import 'dart:html';
import 'dart:convert';

void main() {
  querySelector("#submit-btn")
      ..text = "Submit"
      ..onClick.listen(sendData);
}

void sendData(MouseEvent event) {
  InputElement appId = document.querySelector("#app-id");
  InputElement appData = document.querySelector("#app-data");

  HttpRequest request = new HttpRequest(); // create a new XHR

  // add an event handler that is called when the request finishes
  request.onReadyStateChange.listen((_) {
    if (request.readyState == HttpRequest.DONE &&
       (request.status == 200 || request.status == 0)) {
      // data saved OK.
      print(request.responseText); // output the response from the server
    }
  });

  //Store form data
  Map data = new Map();
  data['app_id'] = appId.value;
  data['app_data'] = appData.value;


  //setup to POST data to the server
  //request.open("POST","http://54.69.228.220/reese/", async: false);
  request.open("POST","http://54.69.228.220/reese/", async: true);

  //we can set the content type to "application/x-www-form-urlencoded; charset=UTF-8"
  //otherwise the Django side gets an empty POST Query dictionary,
  //If we set the content-type to application/json, we can use the request.body
  //variable in Django
  //request.setRequestHeader("Content-type","application/x-www-form-urlencoded;
charset=UTF-8");

  request.setRequestHeader("Content-type","application/json");

  request.send(JSON.encode(data)); // perform the async POST
}
```

One would expect that this would be enough, however, if you try to post something right now, you'd get a 403 response similar to the screenshot below



Logging for Frog Pond!

App Id: asdasdf

App Data: asddsasdf

Submit
Waiting for 54.69.228.220...
Elements Network Sources Timeline Profiles Resources Audits Observatory Console
Preserve log
Disable cache

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline
logging.html	GET	200 OK	text/html	Other	1.0 KB 781 B	8 ms 5 ms	●
logging.dart	GET	200 OK	application/dart	logging.html:9 Parser	1.4 KB 1.1 KB	15 ms 13 ms	●
logging.css	GET	200 OK	text/css	logging.html:11 Parser	1.5 KB 1.2 KB	25 ms 23 ms	●
dart.js /packages/browser	GET	200 OK	application/javascript	logging.html:10 Parser	1.5 KB 1.2 KB	40 ms 38 ms	●
reese/ 54.69.228.220	OPTIONS	200 OK	text/html	Other	372 B 0 B	270 ms 269 ms	●
reese/ 54.69.228.220	POST	403 FORBIDDEN	text/html	Other	1.4 KB 1006 B	294 ms 294 ms	●

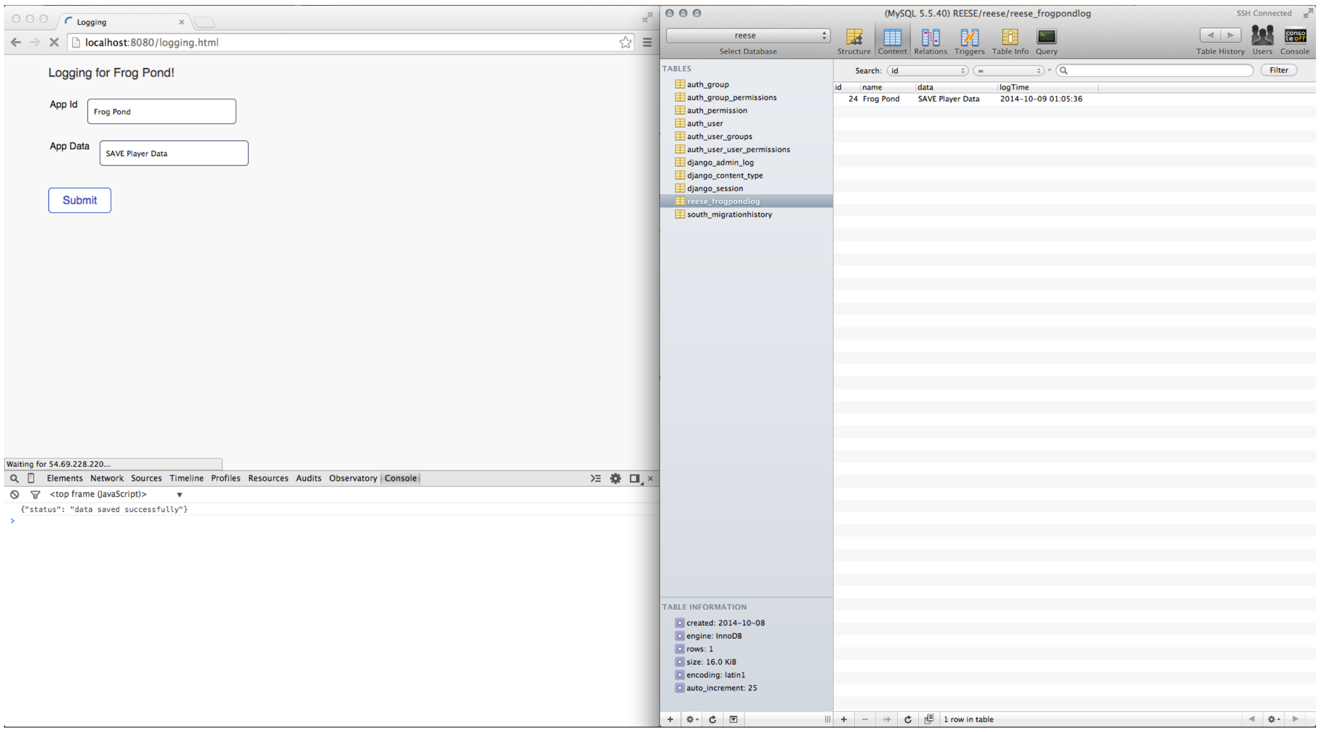
We need to do one last thing :).

Setting the Django csrf flag

It's also called a decorator in Django parlance. Adding a decorator "@csrf_exempt" just above your Django view would do it

```
from django.views.decorators.csrf import csrf_exempt

#include the decorator “@csrf_exempt” for all views that receive data without csrf token
@csrf_exempt
def reeselog(request):
    if request.method == "POST":
        try:
            #read request data as json from request body
            request_data = json.loads(request.body)

            #store the app_id and app_data payload
            app_id = request_data['app_id']
            app_data= request_data['app_data']
        except:
            app_id = "appID error"
            app_data = "appData error"
            return HttpResponse(json.dumps({'status': 'failed to saved data'}),
content_type="application/json")

        #create a log object and save it in the database
        log_data = FrogPondLog(name=app_id, data=app_data)
        log_data.save()

        return HttpResponse(json.dumps({'status': 'data saved successfully'}),
content_type="application/json")
```

Now, if you try to POST data from the remote/cross-site, it should work :)



Logging
localhost:8080/logging.html
Submit
Waiting for 54.69.228.220...
Elements Network Sources Timeline Profiles Resources Audits Observatory Console
<top frame (JavaScript)>
(MySQL 5.5.40) REESE/reese/reese_frogpondlog
SSH Connected
reese
Select Database
Structure Content Relations Triggers Table Info Query
Table History Users Console
Search: id =
Filter

Logging for Frog Pond!

App Id: Frog Pond

App Data: SAVE Player Data

```
{"status": "data saved successfully"}
```

TABLES
- auth_group
- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_user_permissions
- django_admin_log
- django_content_type
- django_session
- reese_frogpondlog
- south_migrationhistory

id	name	data	logTime
24	Frog Pond	SAVE Player Data	2014-10-09 01:05:36

TABLE INFORMATION
- created: 2014-10-08
- engine: InnoDB
- rows: 1
- size: 16.0 KiB
- encoding: latin1
- auto_increment: 25

1 row in table

Addendum

More of a reminder rather than a step, ensure that you have your django App listed under Installed_Apps in settings.py and have run syncdb to create the tables listed under models.py

```
#Ensure that your Django app is listed under INSTALLED_APPS
…
# Application definition
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'south',
    'reese',
)
…

#Make sure you've run syncdb
python manage.py syncdb
```