Deploy in EC2

This chapter explains how to use Ansible to deploy the sample application we just created into Amazon EC2. The steps describe both how to deploy the application and how to configure the AWS components required to run the application, like RDS or DynamoDB, .

The following sections show how to deploy a single instance to create a test environment. The same steps can be applied to create other deployment sets, like a development or a production environment, backed by Amazon AWS services.

The scripts are available in Github. They can be downloaded and modified as required to help you deploy the sample application in EC2.

Using Ansible for deployment

Ansible is a tool for DevOps that allows you to remotely execute commands in your servers. It provides a modular system in which you can define individual tasks to run according to the server role. Ansible complements the scripts with a thorough API that helps with common operations and a set of plugins for popular components (Amazon AWS, PostgreSQL, etc.). As a note, the Amazon API that Ansible provides, which you can use to interact with AWS components, is not used in this article.

Ansible has minimal requirements; the only piece of software necessary on your machine to launch Ansible is Python 2.6 (or greater) and a few Python modules. This facilitates its deployment in any development or system administration computer. The connection with the servers is established via SSH, a common protocol included by default on all Linux machines..

Ansible uses a series of scripts to define the instructions to run on a given server. The scripts are called Playbooks and they follow the YAML format. Ansible allows you to execute the scripts on several servers at once, in parallel. This means that, in Ansible, you execute a Playbook once and all your servers will process that set of commands, receiving the same configuration.

The aim of the Ansible scripts in this article is to create a custom AMI that can be used by Amazon AWS to enable auto-scaling of our application. This means that the scripts are run against one instance only, but they can be easily modified to run targeting many instances.

Structure of Ansible Folder

The Github repository contains the Ansible scripts we use to deploy the sample application. If you explore the folder you may notice that there is a certain structure in it. The folders and files are organized following Best Practices for Ansible.

Using best practices is not just a recommendation but almost a mandatory requirement. Many Ansible commands (on version 1.2 and onwards) assume some files will be located in the locations recommended by the best practices by default. Following these recommendations will save you time when trying to use these commands, and will keep the Playbooks nicely organized.

If you open the folder with the scripts, you will notice the following contents:

- **groups_vars** (folder): contains configuration specific to some server groups
- **host_vars** (folder): contains configuration specific to a particular host
- **roles** (folder): contains the role Playbooks, modules with reusable functionality
- **dbservers.yml:** defines the roles to execute when dealing with database servers
- **developments.hosts:** contains a list of hosts and groups of hosts that belong to a

development environment

- **production.hosts:** contains a list of hosts and groups of hosts that belong to a production environment
- **Readme.md:** some notes on the contents of the folder
- **site.yml:** defines the roles to execute when deploying our application
- **testing.hosts:** contains a list of hosts and groups of hosts that belong to a testing environment
- **webservers.yml:** defines the roles to execute when dealing with web servers

From this set of files and folders, there are some that demand a more detailed explanation. For example, there are three configuration files which contain IP address of servers (*development.hosts*, *testing.hosts* and *production.hosts*). If you edit one, you will see something like:

```
1 # Contains the addresses of all EC2 instances used in testing in each region
2
3 # Webservers in Ireland region
4 [ireland-webservers]
5 ec2-54-216-124-229.eu-west-1.compute.amazonaws.com
```

Line 4 defines a group. A group contains a list of IP that map to existing hosts (in line 5, an EC2 instance) or other groups.

The purpose of these files is to define the servers on which Ansible executes the Playbooks. By defining some groups, we can refer to a set of servers via the group name in the Playbooks, ensuring that we run the proper commands in each host. Groups also allow us to expand or reduce the number of servers managed by Ansible by adding or removing IP from the list.

By having three files with the same group names in them (but different IP in the groups), we are defining different execution environments that can benefit from the same scripts. When running Ansible we can select which environment we want to target.

In the root folder there are three main Playbooks available. Each one can be executed to set up either all our system (*site.yml*) or a specific subset of machines (*webservers.yml* or *dbservers.yml*). The *site.yml* Playbook contains just references to the other two Playbooks. In turn, these Playbooks contain a list of *roles* to run, in order of execution. The *roles* are the ones doing the real work, while these main Playbooks just list which ones we trigger for each environment. This division makes it very easy to change the commands being executed on each host.

In this example, the database Playbook is provided for completeness purposes. We use Amazon RDS, which makes this script unnecessary as we don't manage the database servers. But if you decide to deploy your own servers with your own database versions, you can use the Playbook to trigger the proper configuration steps.

In the folder's list you may notice two folders that contain specific configuration for the Playbooks, depending on the group or host we are targeting during execution. The *group_vars* folder has a list of files named as the groups defined in the hosts files (*webservers, databases, ireland-weberservers*) and a file named *all*. Files that correspond to a group name apply only to servers in that group while *all* applies to all the servers receiving Ansible commands.

Each file contains a list of pairs (key: value) which define variables and values. These variables are then used in the Playbooks, replacing each key by its corresponding value. This allows us, for example, to define the version of Play Framework we want to install in each group and to change it later on by editing only one location.

Finally, the *roles* folder contains the different '*roles*'. A role is a set of Playbooks, templates and other files that do a specific task in a server. For example, the '*common*' role has a list of commands to secure a Linux server, installing packages like *SELinux*. This division on roles facilitates reusing tasks across servers, and we can enable or disable specific tasks for a given group at any moment with minimal effort.

Ansible Scripts

A full description of an Ansible script is outside the scope of this document, please refer to the documentation for this. This section provides a quick overview of the basics of a script.

An Ansible script is a YAML file, which is plain text and human readable, that contains a set of commands to execute in the target servers. An example:

```
 1 # Secures an Ubuntu instance.
 2 #
 3 ---
 4 - name: Update APT package cache
 5   action: apt update_cache=yes
 6
 7 - name: Run apt-get upgrade
 8   action: apt upgrade=yes
 9
10 - name: Install fail2ban
11   action: apt pkg=fail2ban state=installed
```

In the sample above line 3 indicates the start of the script. As you can see, we have three actions defined by a *name* and the *action* itself. The *name* is displayed during the execution of the Playbook, allowing us to know which step is being run. The *action* corresponds to either an Ansible command, using the modules and API provided by the tool, or a raw command executed via *SSH*.

The steps are run in order. If none of them fail, the script finishes with a status of success. If a step fails and we have not opted to ignore errors in that particular step, then the script will abort and we will be notified of the error. Having errors in one server may not mean that the execution will stop in other servers.

When writing Playbooks, be aware that Ansible is not atomic. That is, if the execution of the Playbook fails in a server Ansible doesn't rollback any step that has been executed in that server. This means that it is recommended to add steps that can be run multiple times or with proper safeguards. The script may fail in the middle of execution and you may need to execute it again on the same server, which results in some steps being run twice.

Amazon AWS components

This section shows how to configure the different Amazon AWS components we need for the application. We explain how to do this through the UI provided by the Amazon Console.

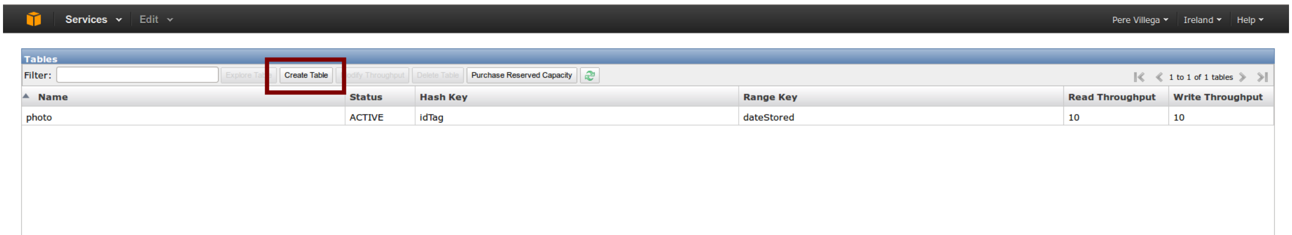
All the steps assume you have an AWS account, you have access to the Console and you are allowed to create components with your account. Management of account security and roles is out of scope for this article and therefore it is not explained.

Be sure to select the region where you want to deploy your components before proceeding, as to avoid issues caused by having components deployed across different regions.

Configure DynamoDB

As mentioned in the previous chapter, we use DynamoDB to store metadata of images and tags retrieved from Flickr. DynamoDB is very useful for this purpose, as metadata is something that may change as the application grows. In DynamoDB we can define a set of tables with no fixed schema and add more "columns" as needed, without having to change any settings in the table.

Using the DynamoDB console we can define the tables that the sample application uses. To create a table, press the *Create Table* button at the top:



Name	Status	Hash Key	Range Key	Read Throughput	Write Throughput
photo	ACTIVE	idTag	dateStored	10	10

This opens a popup to create a table. In it you can write the name of the table and select the type of key used. DynamoDB uses two kinds of keys, *Hash* and *Hash and Range*.

The *Hash* type is used for elements which can be considered unique. For example, in the sample application a *tag* is a unique entity, with no duplicate and no dependency to any other element, so a *hash* key is appropriate.

For the images, which are related to a *tag*, we use a *hash and range* key where the *hash* matches the *hash* of an existing tag and the *range* part, a date stored as a string, allows us to differentiate each individual image belonging to a given tag.

First we want to create a table to store the tags, so we type *tag* as name and select a *hash key* with name *id* and type *string* as the key of the table:

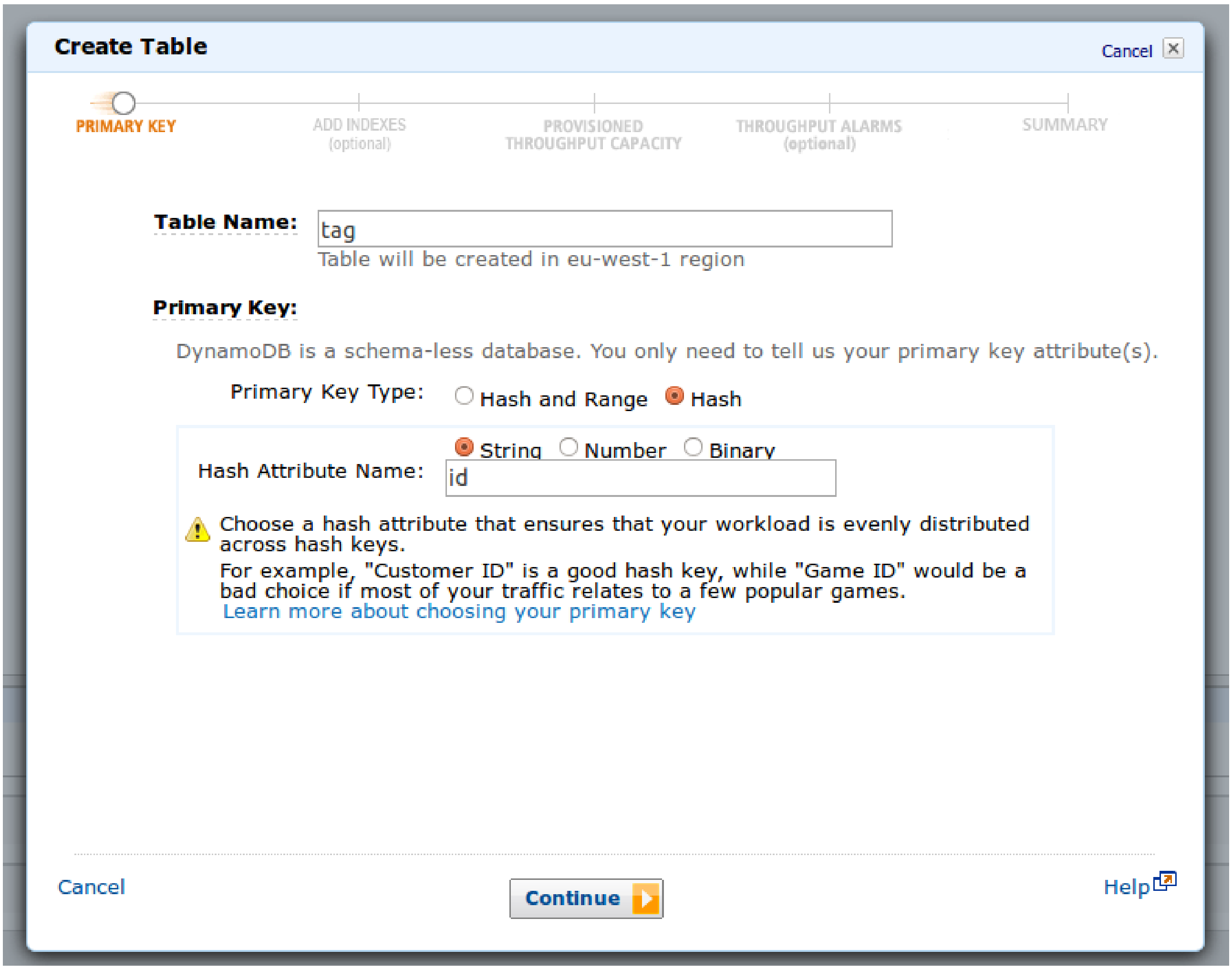


Table Name: tag

Table will be created in eu-west-1 region

Primary Key:

DynamoDB is a schema-less database. You only need to tell us your primary key attribute(s).

Primary Key Type: ○ Hash and Range ● Hash

● String ○ Number ○ Binary

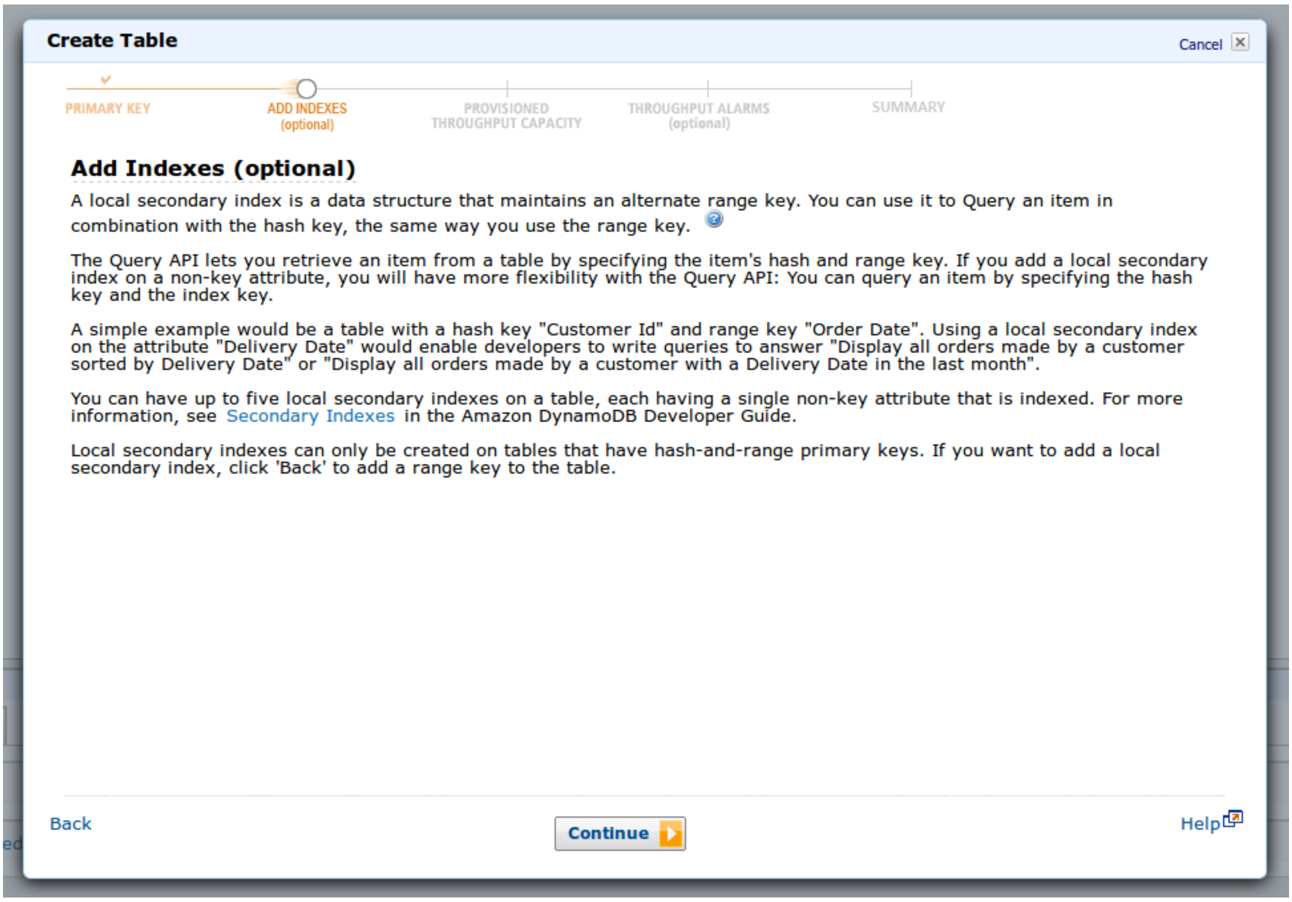
Hash Attribute Name: id

Choose a hash attribute that ensures that your workload is evenly distributed across hash keys.

For example, "Customer ID" is a good hash key, while "Game ID" would be a bad choice if most of your traffic relates to a few popular games.

Learn more about choosing your primary key

Pressing *Continue* brings us to the next screen, where you can add additional indexes:



Add Indexes (optional)

A local secondary index is a data structure that maintains an alternate range key. You can use it to Query an item in combination with the hash key, the same way you use the range key.

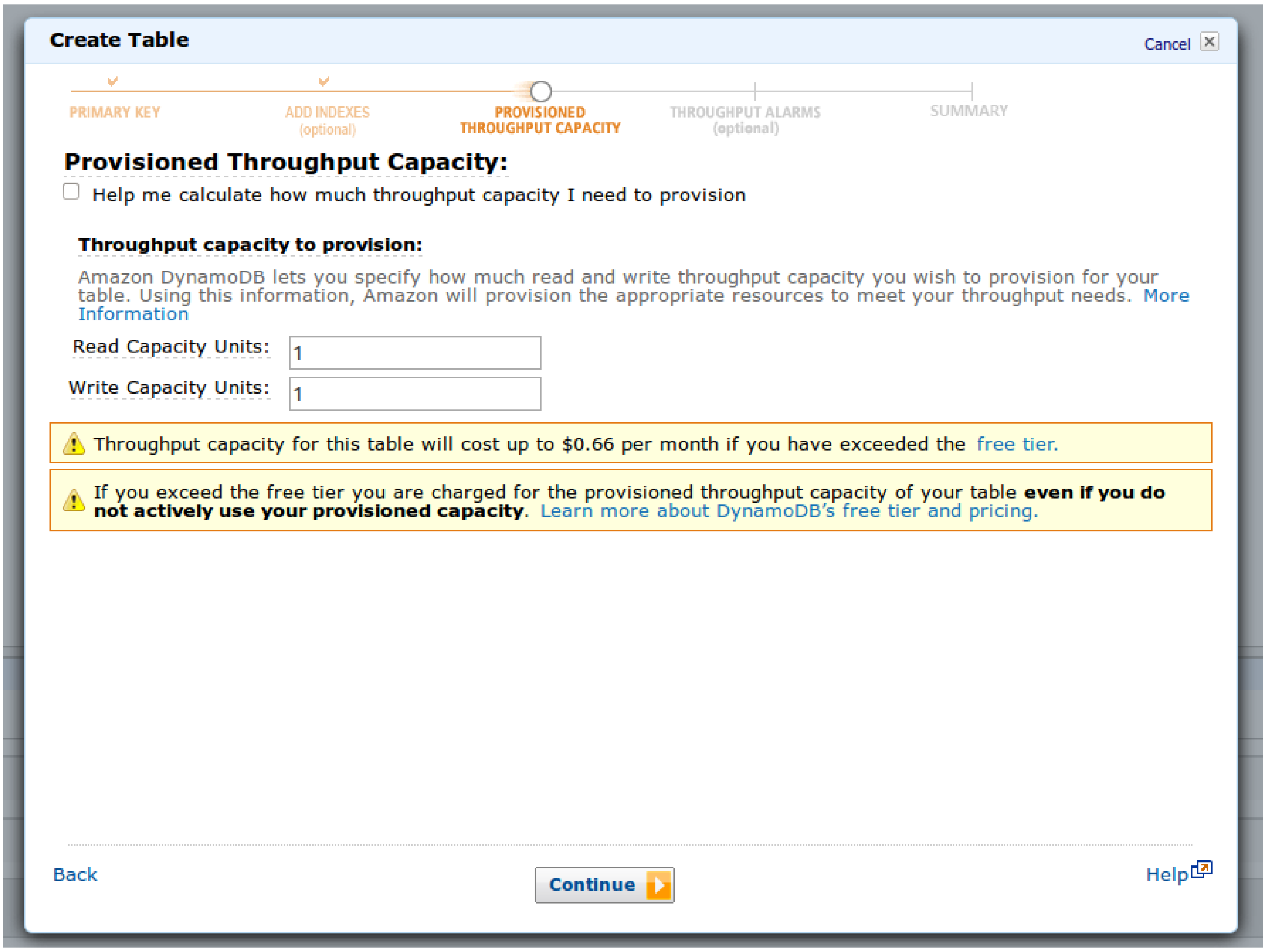
The Query API lets you retrieve an item from a table by specifying the item's hash and range key. If you add a local secondary index on a non-key attribute, you will have more flexibility with the Query API: You can query an item by specifying the hash key and the index key.

A simple example would be a table with a hash key "Customer Id" and range key "Order Date". Using a local secondary index on the attribute "Delivery Date" would enable developers to write queries to answer "Display all orders made by a customer sorted by Delivery Date" or "Display all orders made by a customer with a Delivery Date in the last month".

You can have up to five local secondary indexes on a table, each having a single non-key attribute that is indexed. For more information, see Secondary Indexes in the Amazon DynamoDB Developer Guide.

Local secondary indexes can only be created on tables that have hash-and-range primary keys. If you want to add a local secondary index, click 'Back' to add a range key to the table.

We don't need any, so we press *Continue* again. The following screen lets us select the P*rovisioned Capacity*. DynamoDB reserves a certain capacity for read and write operations on each table you create:



Provisioned Throughput Capacity:

□ Help me calculate how much throughput capacity I need to provision

Throughput capacity to provision:

Amazon DynamoDB lets you specify how much read and write throughput capacity you wish to provision for your table. Using this information, Amazon will provision the appropriate resources to meet your throughput needs. More Information

Read Capacity Units: 1

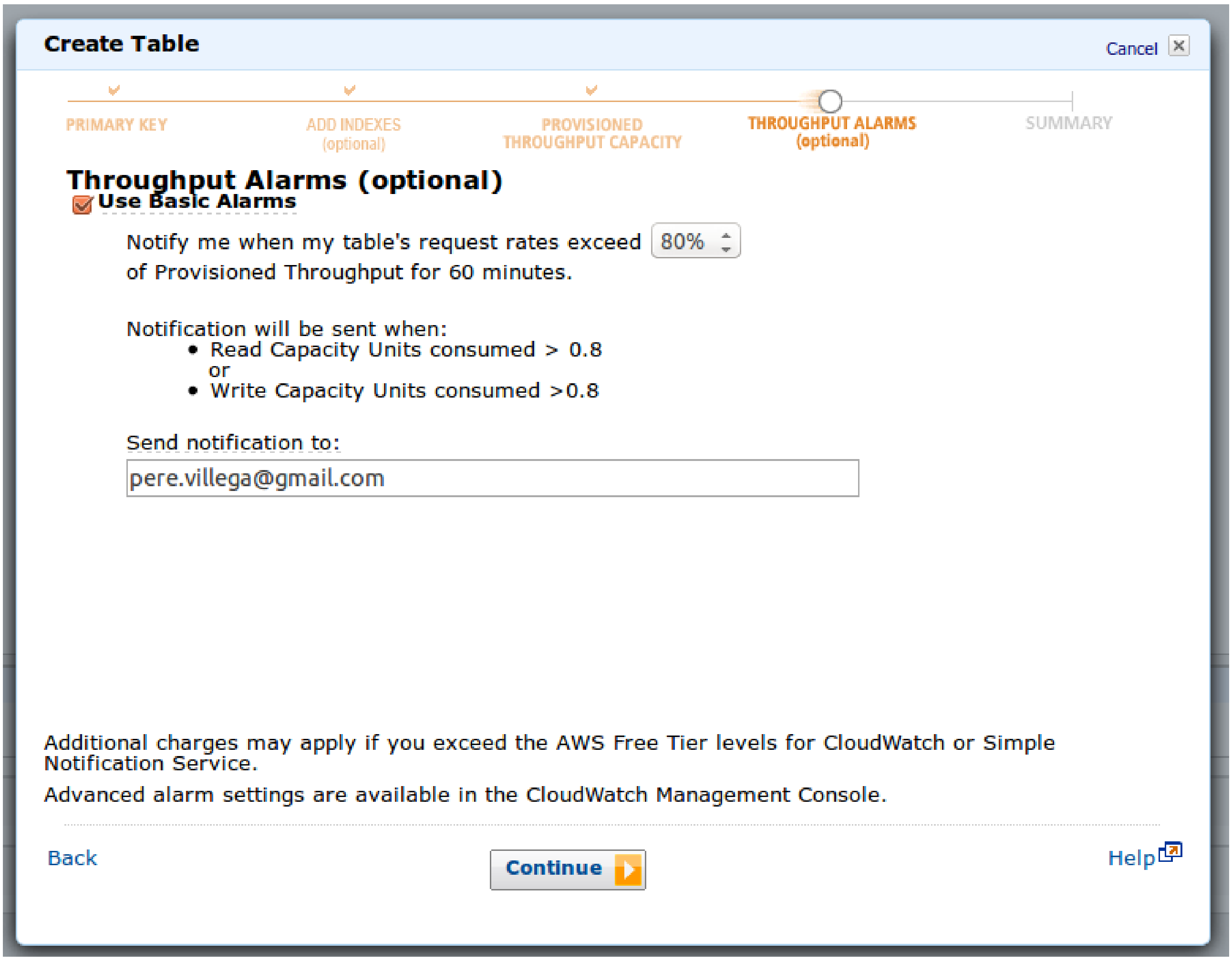
Write Capacity Units: 1

⚠ Throughput capacity for this table will cost up to $0.66 per month if you have exceeded the free tier.

⚠ If you exceed the free tier you are charged for the provisioned throughput capacity of your table **even if you do not actively use your provisioned capacity**. Learn more about DynamoDB's free tier and pricing.

The popup contains links that provide a more detailed description on how does capacity work, but a rough approximation is that each capacity unit is one read or write of a random item per second. Be aware that Amazon bills per capacity reserved, even if it is not used, so try to be conservative in your choice as you can modify it later on if required.

In the next screen we can configure alarms so we are notified if the read or write capacity is over a certain limit, that way the we can act accordingly by increasing the resources allocated for the table:



Throughput Alarms (optional)

☑ Use Basic Alarms

Notify me when my table's request rates exceed 80% of Provisioned Throughput for 60 minutes.

Notification will be sent when:

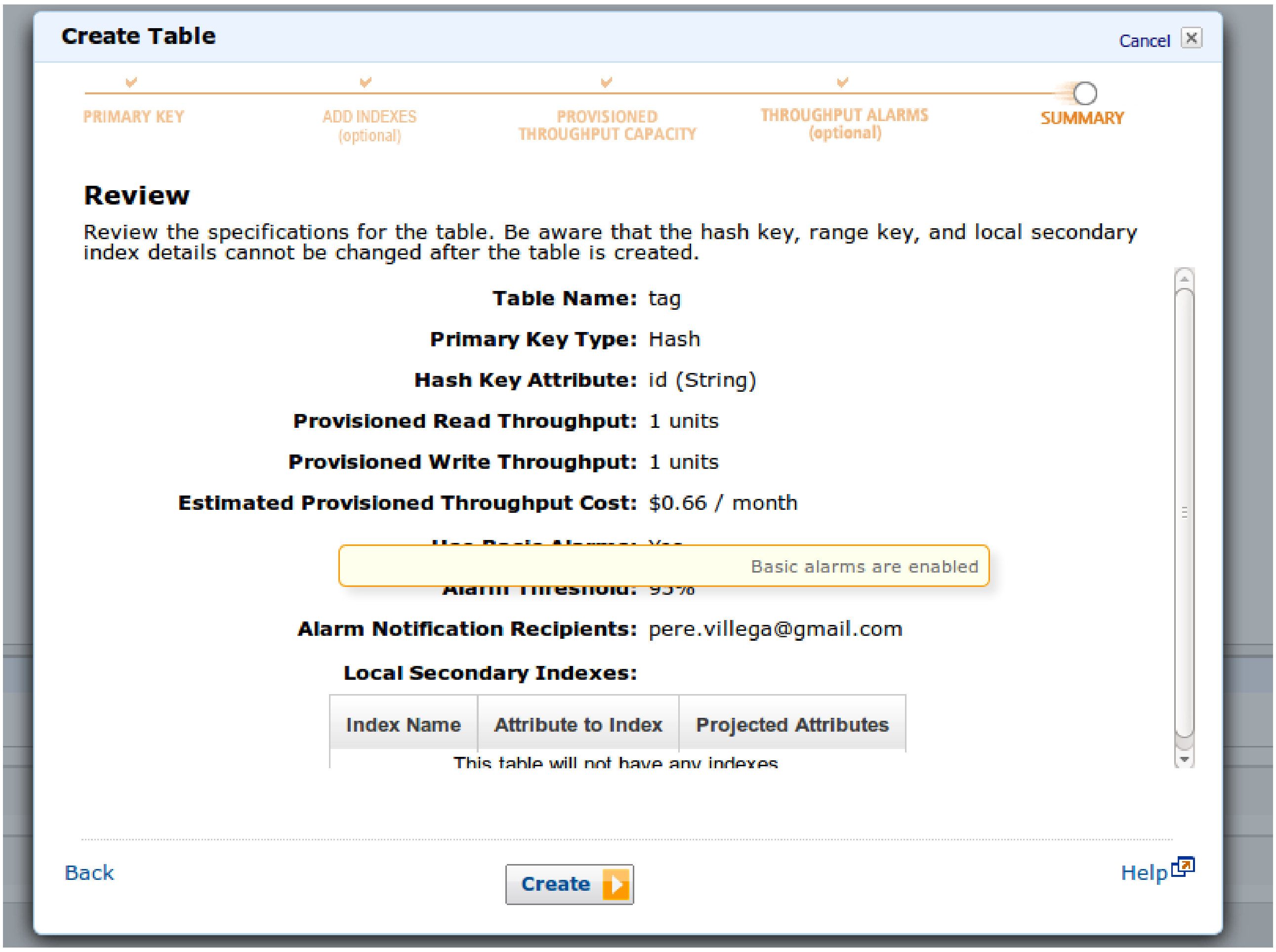
- Read Capacity Units consumed > 0.8
  or
- Write Capacity Units consumed >0.8

Send notification to: pere.villega@gmail.com

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service.

Advanced alarm settings are available in the CloudWatch Management Console.

The last screen is a summary of the table configuration, including an estimate of the monthly cost of keeping the table active:



Review

Review the specifications for the table. Be aware that the hash key, range key, and local secondary index details cannot be changed after the table is created.

Table Name: tag

Primary Key Type: Hash

Hash Key Attribute: id (String)

Provisioned Read Throughput: 1 units

Provisioned Write Throughput: 1 units

Estimated Provisioned Throughput Cost: $0.66 / month

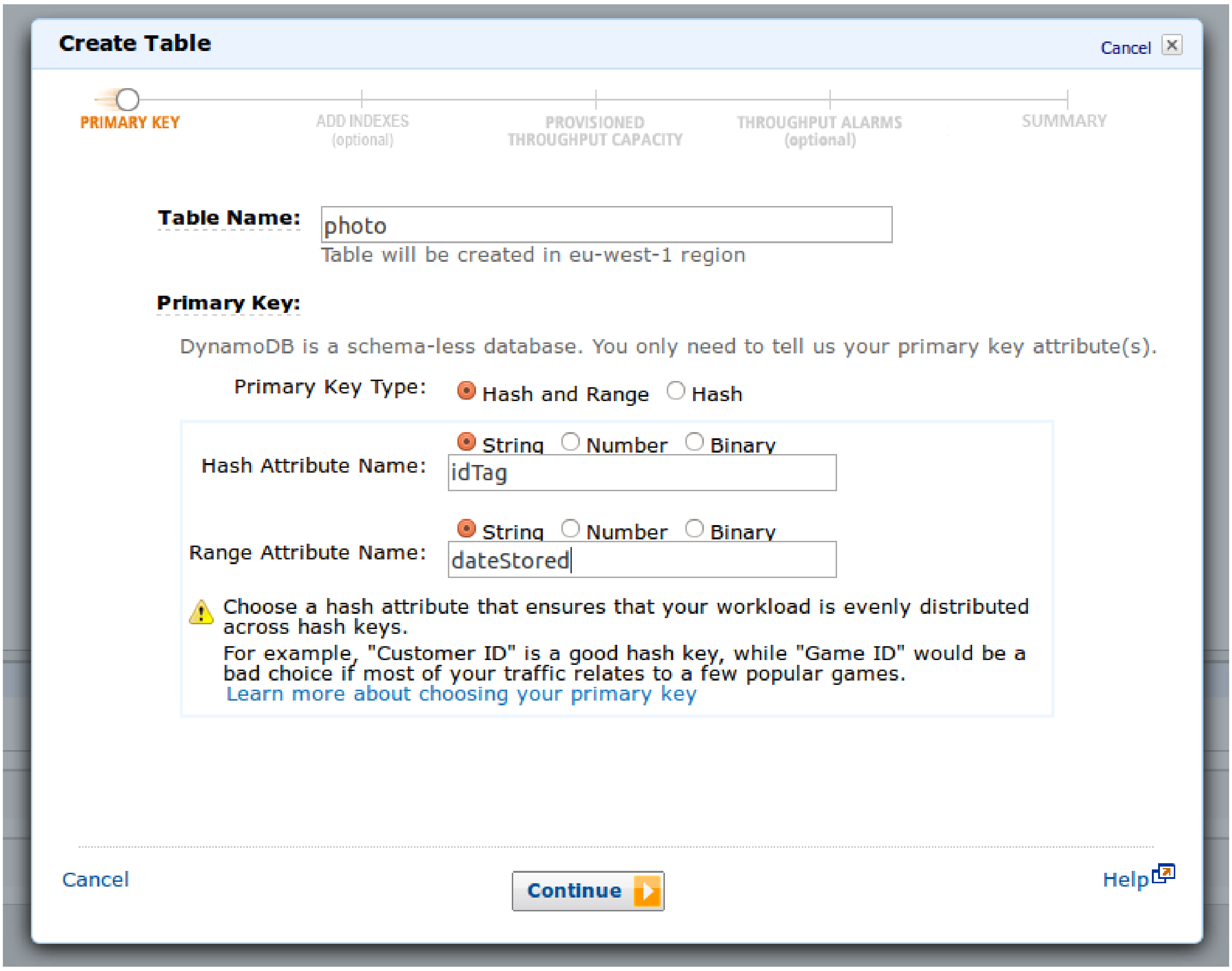
Basic alarms are enabled

Alarm Notification Recipients: pere.villega@gmail.com

Local Secondary Indexes:

Index Name	Attribute to Index	Projected Attributes
This table will not have any indexes		

Once the *tag* table is created, we can add the *photos* table to DynamoDB. Set a *hash and range* key in the first screen, as follows:



Create Table

Table Name: photo

Table will be created in eu-west-1 region

Primary Key:

DynamoDB is a schema-less database. You only need to tell us your primary key attribute(s).

Primary Key Type: ● Hash and Range ○ Hash

Hash Attribute Name: ● String ○ Number ○ Binary idTag

Range Attribute Name: ● String ○ Number ○ Binary dateStored

Choose a hash attribute that ensures that your workload is evenly distributed across hash keys.

For example, "Customer ID" is a good hash key, while "Game ID" would be a bad choice if most of your traffic relates to a few popular games.

Learn more about choosing your primary key

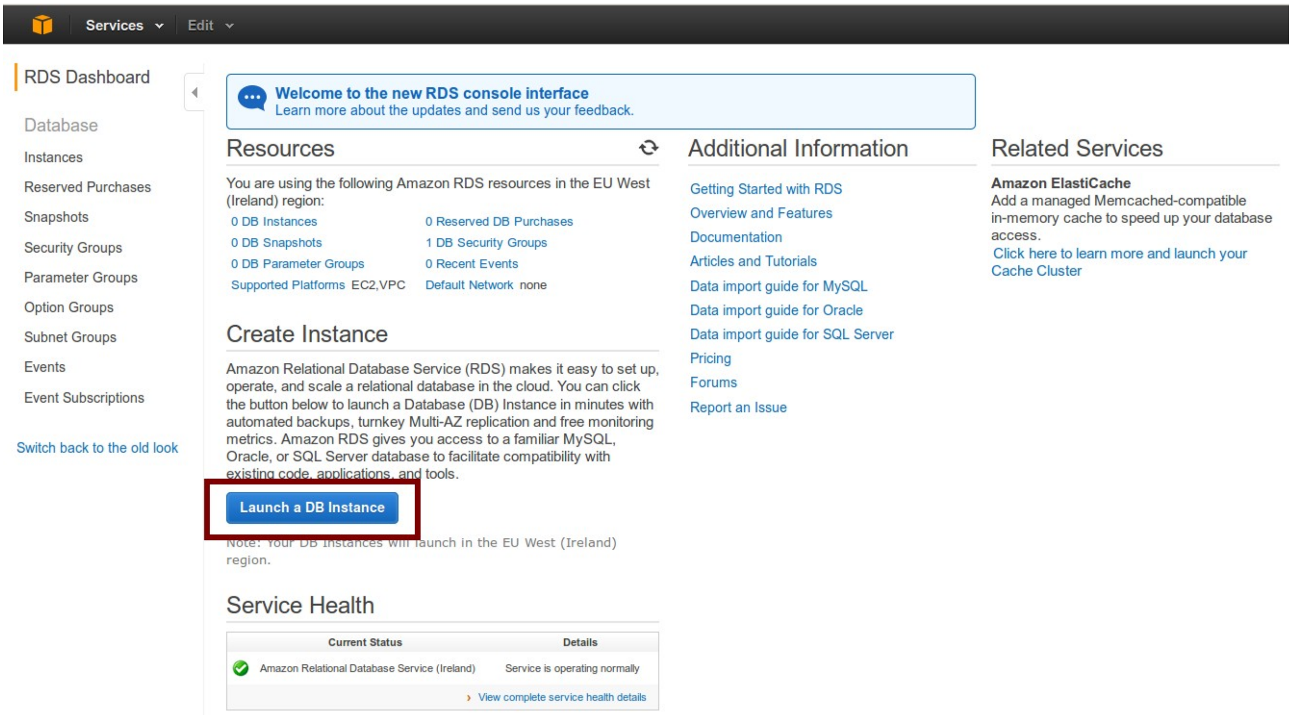
The other screens of the process match the ones seen when creating the *tag* table.

At this point we have created the tables we need for our application in DynamoDB. You can test them by configuring the Amazon AWS connection details in the sample application, in your development environment, and running the application. After the metadata is loaded from Flickr, you can come back to the console and explore the tables to see the stored data.

Configure Amazon RDS (MySQL)

As mentioned when creating the sample application, we use MySQL to store all the user details generated by Play Authenticate. To deploy our application in Amazon AWS, we need to create a MySQL database in RDS to store the data.

The RDS console allows you to create a MySQL instance that is completely managed by Amazon AWS, including the backups of your data. To start, press the *Launch a DB instance* button:



Welcome to the new RDS console interface
Learn more about the updates and send us your feedback.

Resources

You are using the following Amazon RDS resources in the EU West (Ireland) region:

0 DB Instances	0 Reserved DB Purchases
0 DB Snapshots	1 DB Security Groups
0 DB Parameter Groups	0 Recent Events
Supported Platforms EC2,VPC	Default Network none

Create Instance

Amazon Relational Database Service (RDS) makes it easy to set up, operate, and scale a relational database in the cloud. You can click the button below to launch a Database (DB) Instance in minutes with automated backups, turnkey Multi-AZ replication and free monitoring metrics. Amazon RDS gives you access to a familiar MySQL, Oracle, or SQL Server database to facilitate compatibility with existing code, applications, and tools.

Note: Your DB Instances will launch in the EU West (Ireland) region.

Service Health

Current Status	Details
✓ Amazon Relational Database Service (Ireland)	Service is operating normally

› View complete service health details

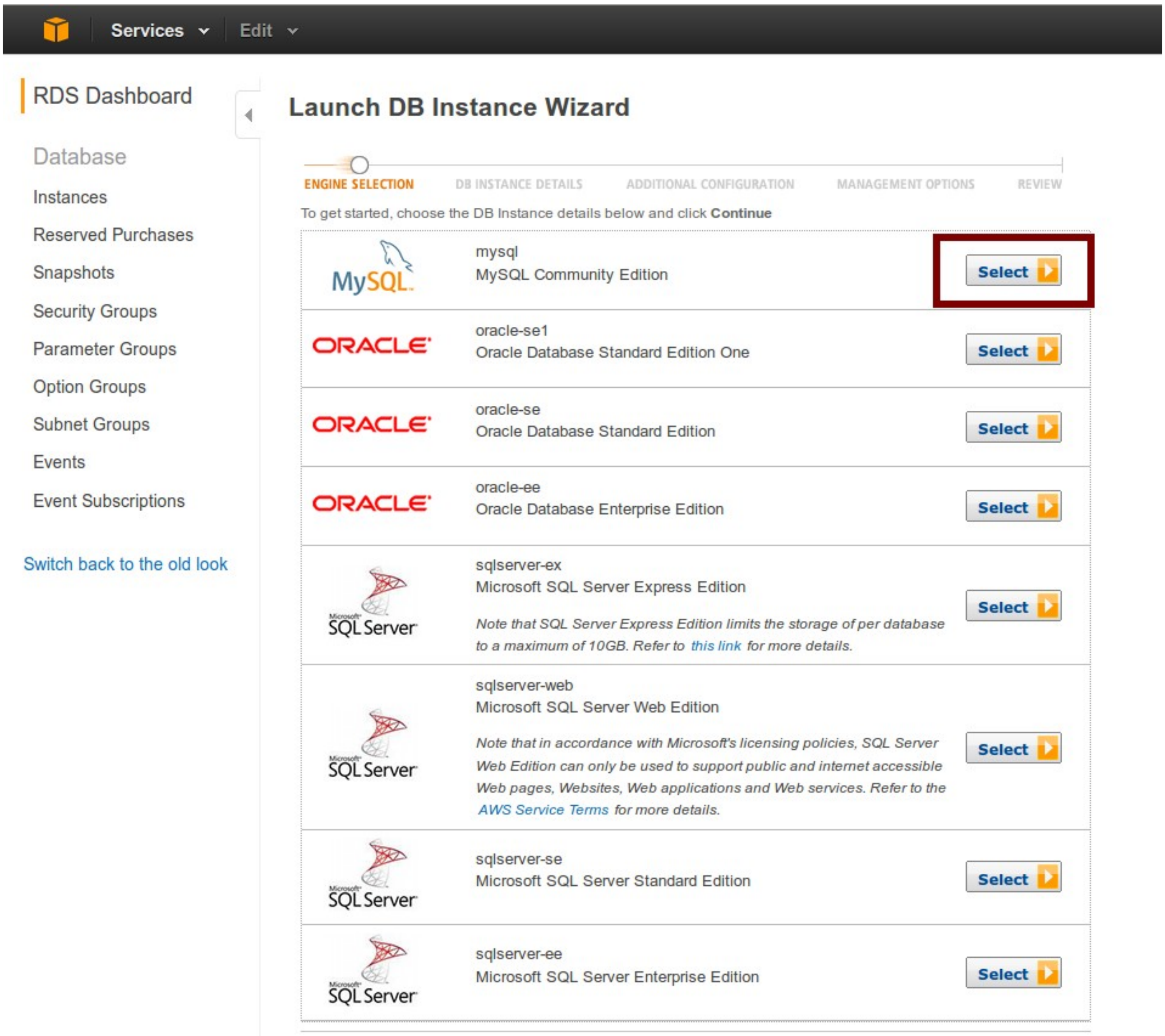
Additional Information

- Getting Started with RDS
- Overview and Features
- Documentation
- Articles and Tutorials
- Data import guide for MySQL
- Data import guide for Oracle
- Data import guide for SQL Server
- Pricing
- Forums
- Report an Issue

Related Services

Amazon ElastiCache
Add a managed Memcached-compatible in-memory cache to speed up your database access.
Click here to learn more and launch your Cache Cluster

After pressing the button, the wizard to create the database is started. At the first screen, select the database type needed for the application, in this case *MySQL*:

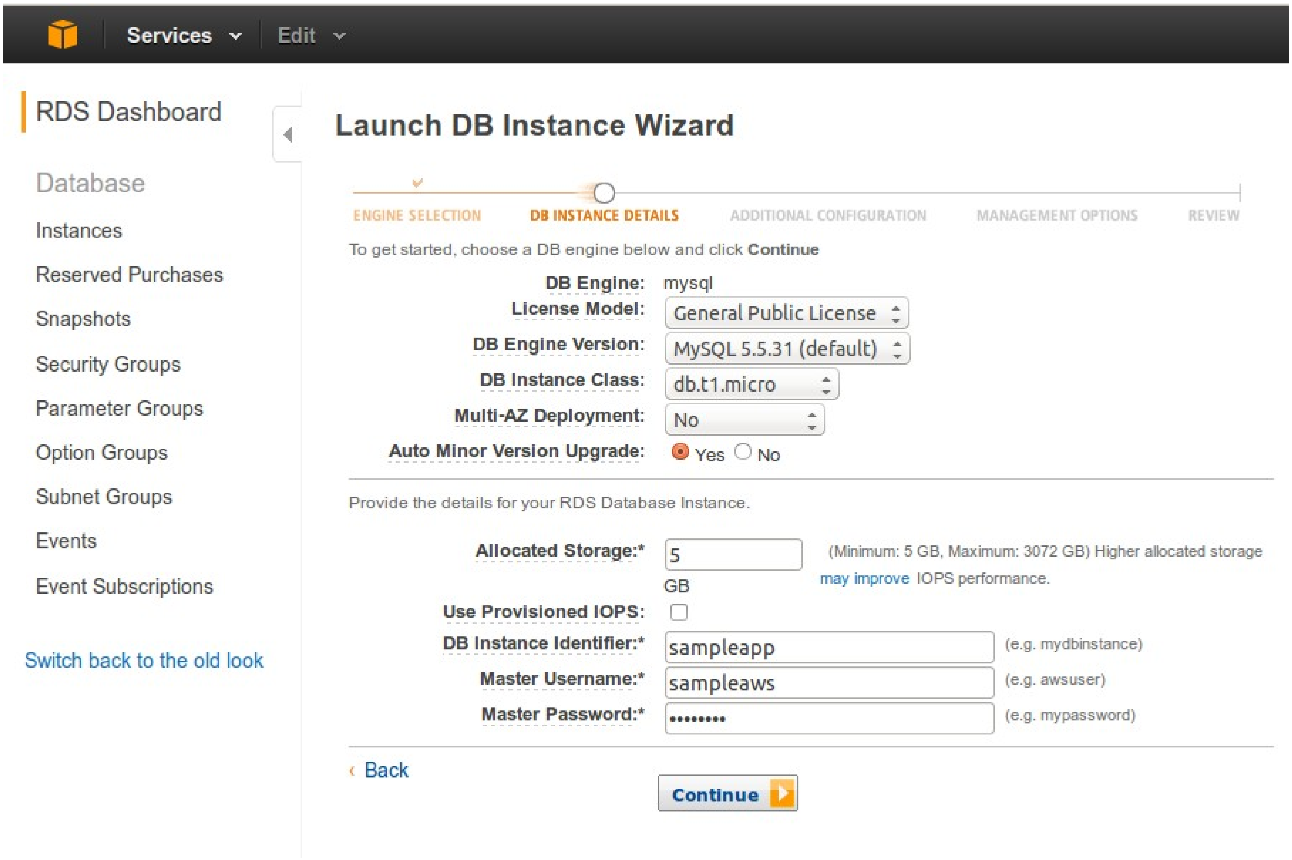


Launch DB Instance Wizard

To get started, choose the DB Instance details below and click **Continue**

MySQL	mysql MySQL Community Edition	Select
ORACLE	oracle-se1 Oracle Database Standard Edition One	Select
ORACLE	oracle-se Oracle Database Standard Edition	Select
ORACLE	oracle-ee Oracle Database Enterprise Edition	Select
Microsoft SQL Server	sqlserver-ex Microsoft SQL Server Express Edition *Note that SQL Server Express Edition limits the storage of per database to a maximum of 10GB. Refer to this link for more details.*	Select
Microsoft SQL Server	sqlserver-web Microsoft SQL Server Web Edition *Note that in accordance with Microsoft's licensing policies, SQL Server Web Edition can only be used to support public and internet accessible Web pages, Websites, Web applications and Web services. Refer to the AWS Service Terms for more details.*	Select
Microsoft SQL Server	sqlserver-se Microsoft SQL Server Standard Edition	Select
Microsoft SQL Server	sqlserver-ee Microsoft SQL Server Enterprise Edition	Select

In the next screen we can configure the database:



Launch DB Instance Wizard

To get started, choose a DB engine below and click **Continue**

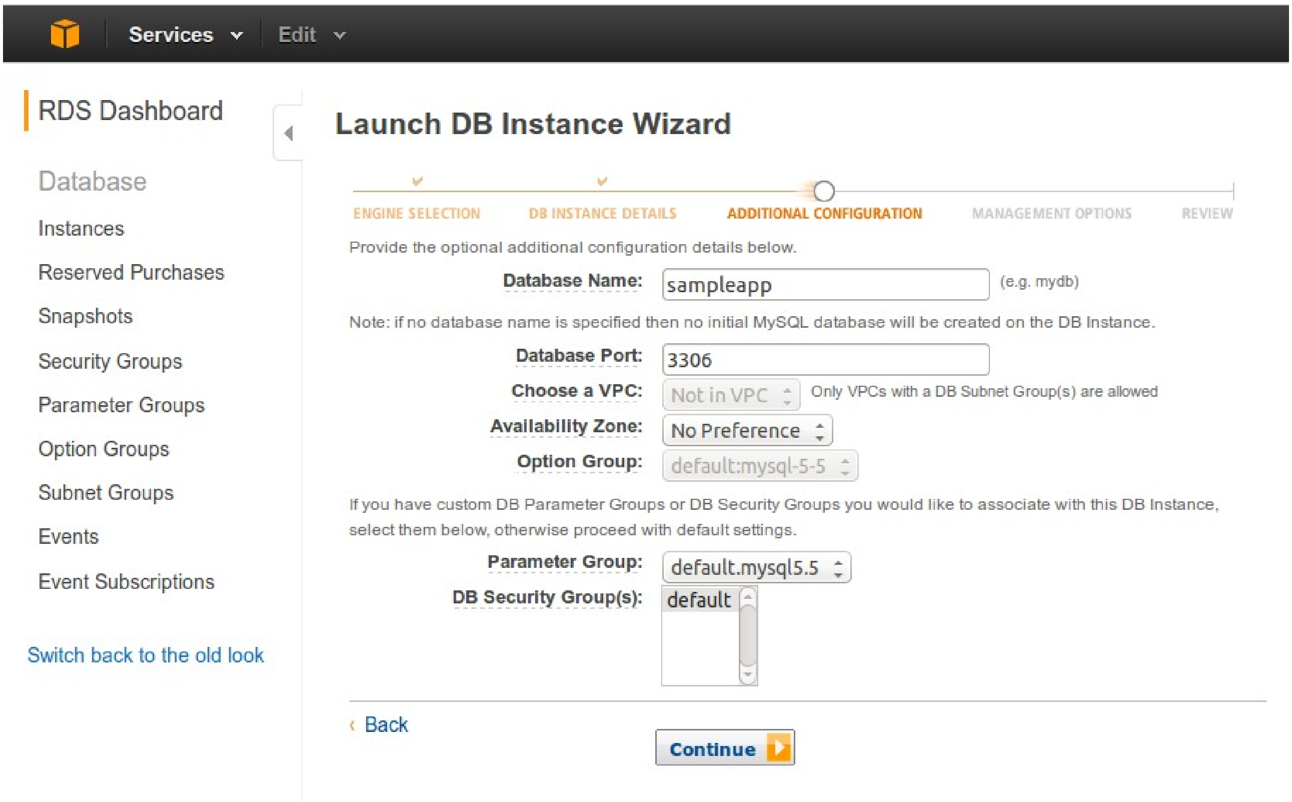
DB Engine:	mysql
License Model:	General Public License
DB Engine Version:	MySQL 5.5.31 (default)
DB Instance Class:	db.t1.micro
Multi-AZ Deployment:	No
Auto Minor Version Upgrade:	● Yes ○ No

Provide the details for your RDS Database Instance.

Allocated Storage:*	5 GB	(Minimum: 5 GB, Maximum: 3072 GB) Higher allocated storage may improve IOPS performance.
Use Provisioned IOPS:	□	
DB Instance Identifier:*	sampleapp	(e.g. mydbinstance)
Master Username:*	sampleaws	(e.g. awsuser)
Master Password:*	••••••••	(e.g. mypassword)

Among the different options available, you can select the version of MySQL to use, the size of the instance and its allocated storage. Remember to write down the identifier, username and password as they are required by the sample application to connect to the instance.

After pressing *Continue*, the next screen allows us to configure additional details, like the port used to connect to the database:



Launch DB Instance Wizard

Provide the optional additional configuration details below.

Database Name:	sampleapp	(e.g. mydb)

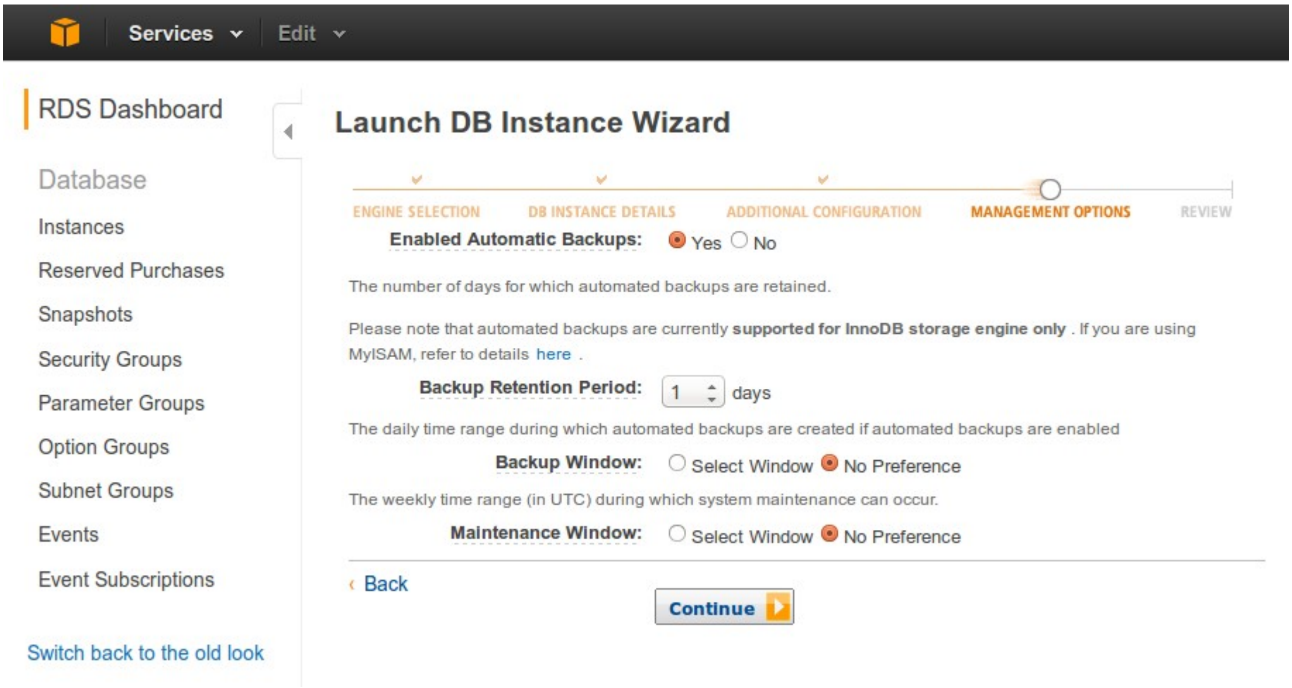
Note: if no database name is specified then no initial MySQL database will be created on the DB Instance.

Database Port:	3306	
Choose a VPC:	Not in VPC	Only VPCs with a DB Subnet Group(s) are allowed
Availability Zone:	No Preference	
Option Group:	default:mysql-5-5	

If you have custom DB Parameter Groups or DB Security Groups you would like to associate with this DB Instance, select them below, otherwise proceed with default settings.

Parameter Group:	default.mysql5.5
DB Security Group(s):	default

The following screen provides backup configuration, including the time at which we want to create the backups:



Enabled Automatic Backups: ● Yes ○ No

The number of days for which automated backups are retained.

Please note that automated backups are currently **supported for InnoDB storage engine only** . If you are using MyISAM, refer to details here .

Backup Retention Period: 1 days

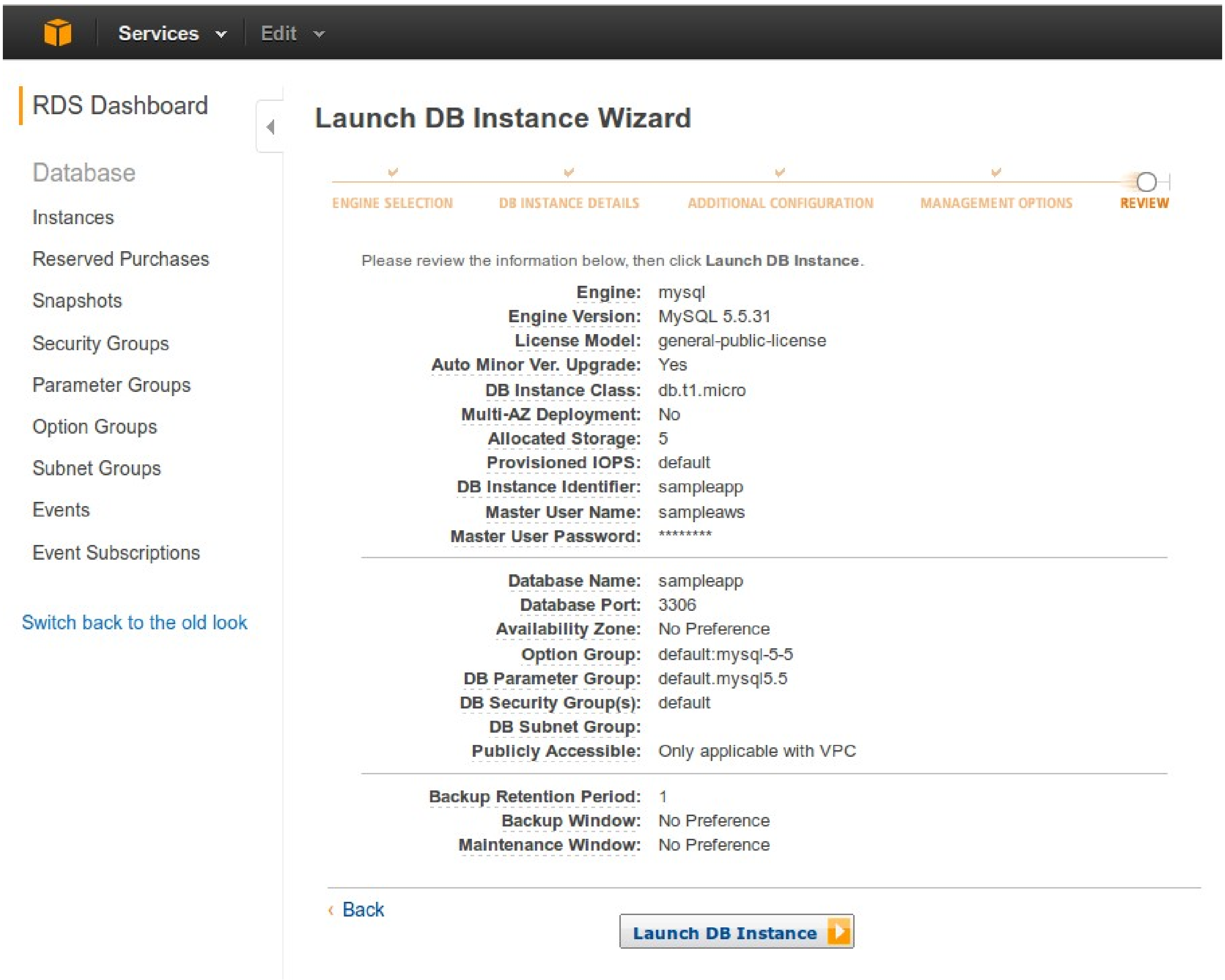
The daily time range during which automated backups are created if automated backups are enabled

Backup Window: ○ Select Window ● No Preference

The weekly time range (in UTC) during which system maintenance can occur.

Maintenance Window: ○ Select Window ● No Preference

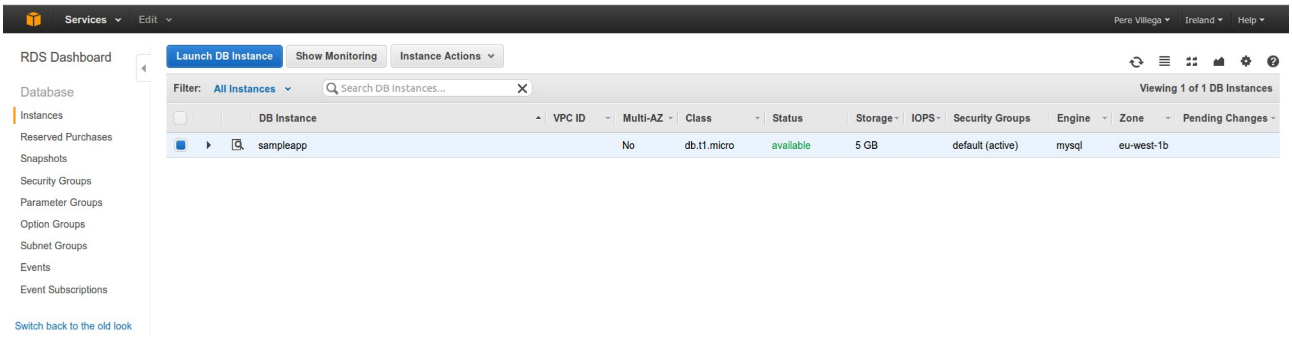
The last screen shows a summary of the configuration for this database instance:



Please review the information below, then click **Launch DB Instance**.

Engine:	mysql
Engine Version:	MySQL 5.5.31
License Model:	general-public-license
Auto Minor Ver. Upgrade:	Yes
DB Instance Class:	db.t1.micro
Multi-AZ Deployment:	No
Allocated Storage:	5
Provisioned IOPS:	default
DB Instance Identifier:	sampleapp
Master User Name:	sampleaws
Master User Password:	********
Database Name:	sampleapp
Database Port:	3306
Availability Zone:	No Preference
Option Group:	default:mysql-5-5
DB Parameter Group:	default.mysql5.5
DB Security Group(s):	default
DB Subnet Group:	
Publicly Accessible:	Only applicable with VPC
Backup Retention Period:	1
Backup Window:	No Preference
Maintenance Window:	No Preference

After pressing *Launch DB Instance* we can see the database being created in the dashboard:



DB Instance	VPC ID	Multi-AZ	Class	Status	Storage	IOPS	Security Groups	Engine	Zone	Pending Changes
sampleapp		No	db.t1.micro	available	5 GB		default (active)	mysql	eu-west-1b	

At this stage we have a fully functional MySQL database that we can use to store user details in the sample application.

If you need to scale your RDS instance, Amazon gives you two alternatives. You can resize the instance, using a more powerful box to run the database. Or you can create read-only replicas, which increase the throughput of the database when reading data. Both actions are managed by Amazon, you just need to select to command via the UI and the database settings are modified as requested.

Configure an EC2 instance

This section describes how to create your Amazon EC2 instance. After the instance is running, Ansible is used to configure it (deploying the sample application code) and a custom AMI is created.

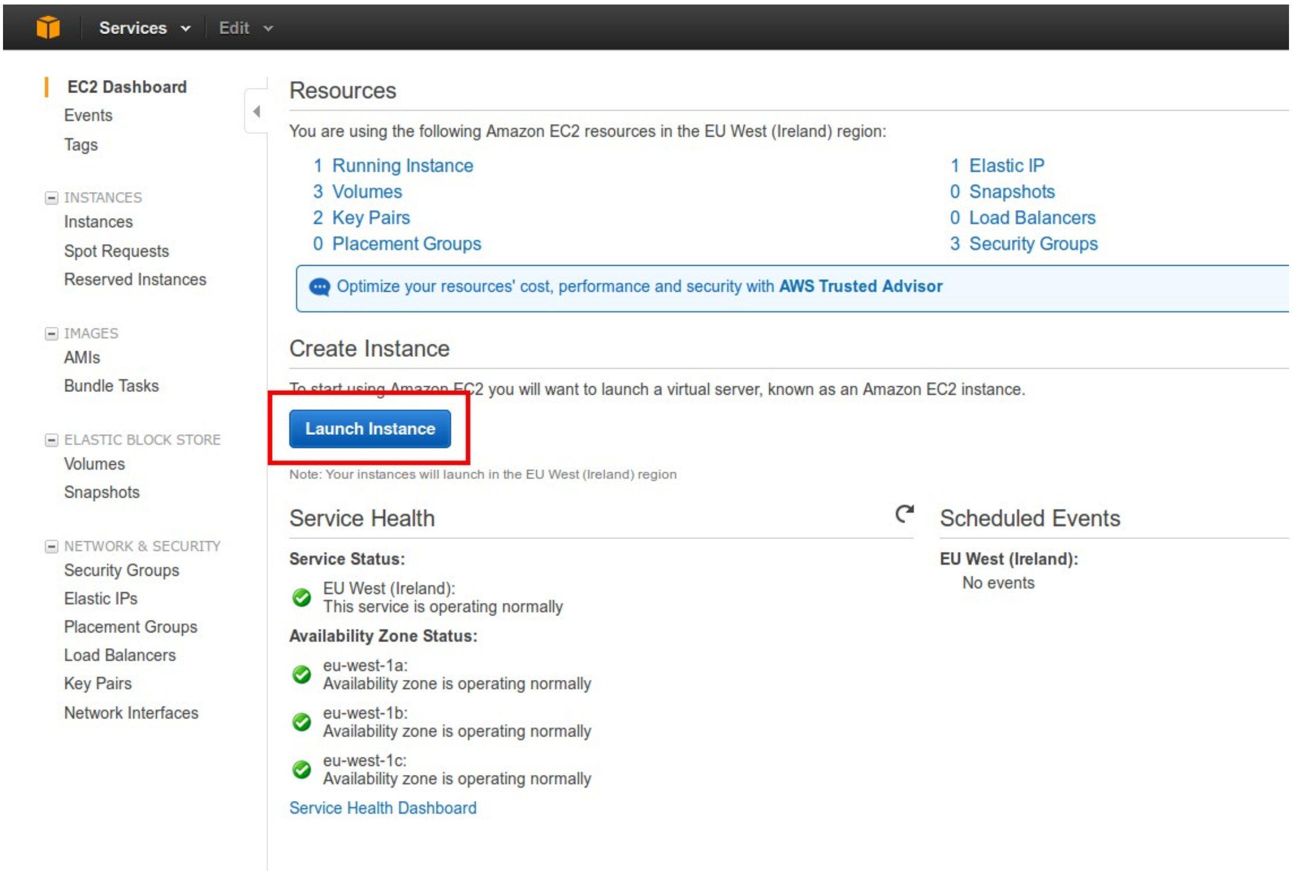
Be aware that this section only describes how to create a single instance that is used to generate a custom AMI. These steps can be replicated to create instances for other environments, like a development environment or a production deployment.

Alternatively, once the custom AMI is created you can use it to deploy additional EC2 instances as you need them. Simply select the custom AMI instead of the default Ubuntu one when creating the EC2 instance, and skip the remaining steps.

Create the instance

The Amazon EC2 console allows you to manage your EC2 instances. An EC2 instance is a virtual server managed by Amazon and billed by the hour. Amazon provides several types of instances, with different amounts of RAM and CPU power available for each one.

To create an EC2 instance press the *Launch Instance* button in the console:



Resources

You are using the following Amazon EC2 resources in the EU West (Ireland) region:

1 Running Instance	1 Elastic IP
3 Volumes	0 Snapshots
2 Key Pairs	0 Load Balancers
0 Placement Groups	3 Security Groups

Optimize your resources' cost, performance and security with **AWS Trusted Advisor**

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 EU West (Ireland) region

Service Health

Service Status:

EU West (Ireland):
This service is operating normally

Availability Zone Status:

eu-west-1a:
Availability zone is operating normally

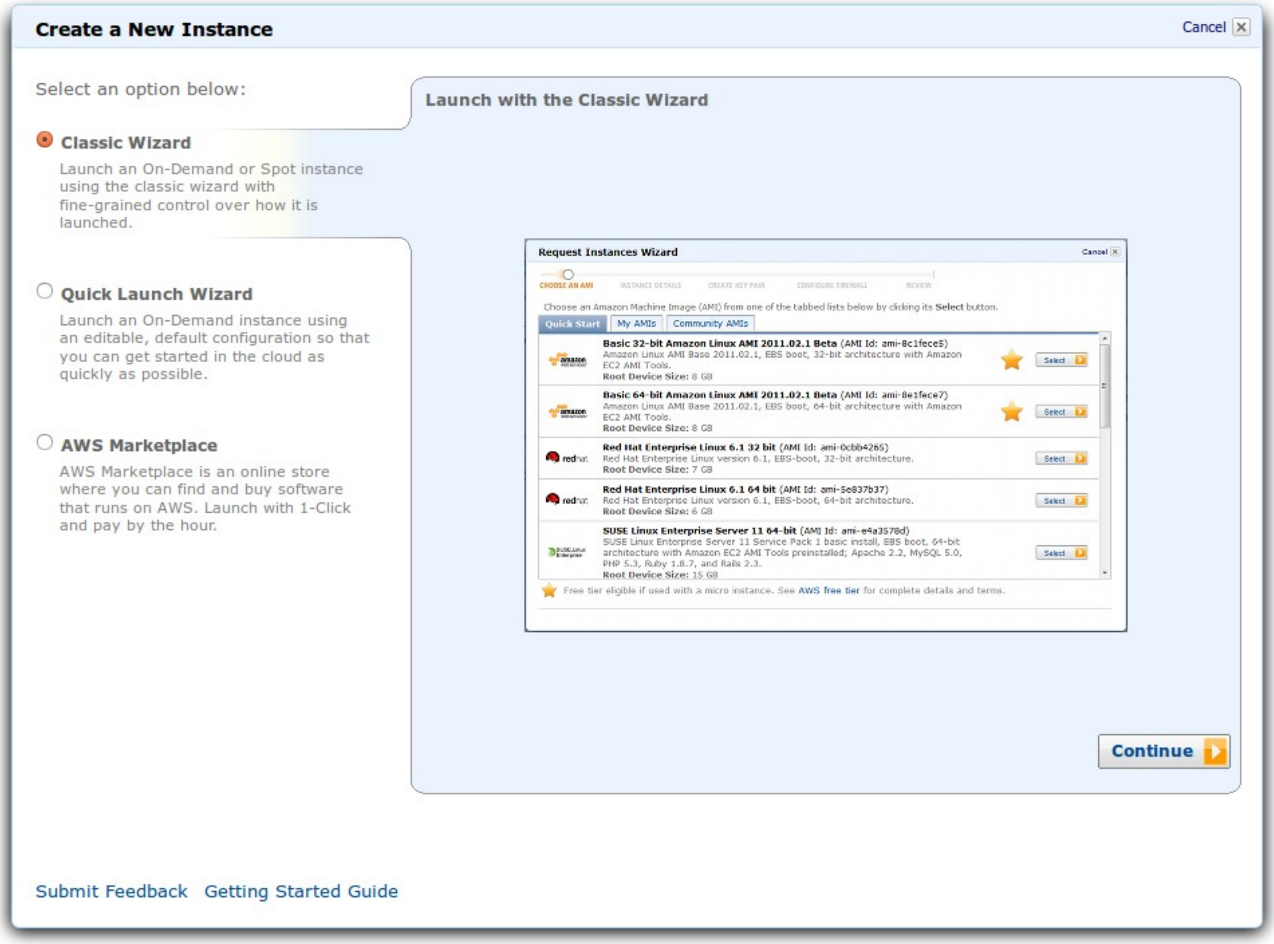
eu-west-1b:
Availability zone is operating normally

eu-west-1c:
Availability zone is operating normally

Scheduled Events

EU West (Ireland):
No events

This opens a window in which you can select how to create the instance:



Create a New Instance

Select an option below:

● **Classic Wizard**
Launch an On-Demand or Spot instance using the classic wizard with fine-grained control over how it is launched.

○ **Quick Launch Wizard**
Launch an On-Demand instance using an editable, default configuration so that you can get started in the cloud as quickly as possible.

○ **AWS Marketplace**
AWS Marketplace is an online store where you can find and buy software that runs on AWS. Launch with 1-Click and pay by the hour.

Launch with the Classic Wizard

Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

Basic 32-bit Amazon Linux AMI 2011.02.1 Beta (AMI Id: ami-8c1fece5)
Amazon Linux AMI Base 2011.02.1, EBS boot, 32-bit architecture with Amazon EC2 AMI Tools.
Root Device Size: 8 GB

Basic 64-bit Amazon Linux AMI 2011.02.1 Beta (AMI Id: ami-8e1fece7)
Amazon Linux AMI Base 2011.02.1, EBS boot, 64-bit architecture with Amazon EC2 AMI Tools.
Root Device Size: 8 GB

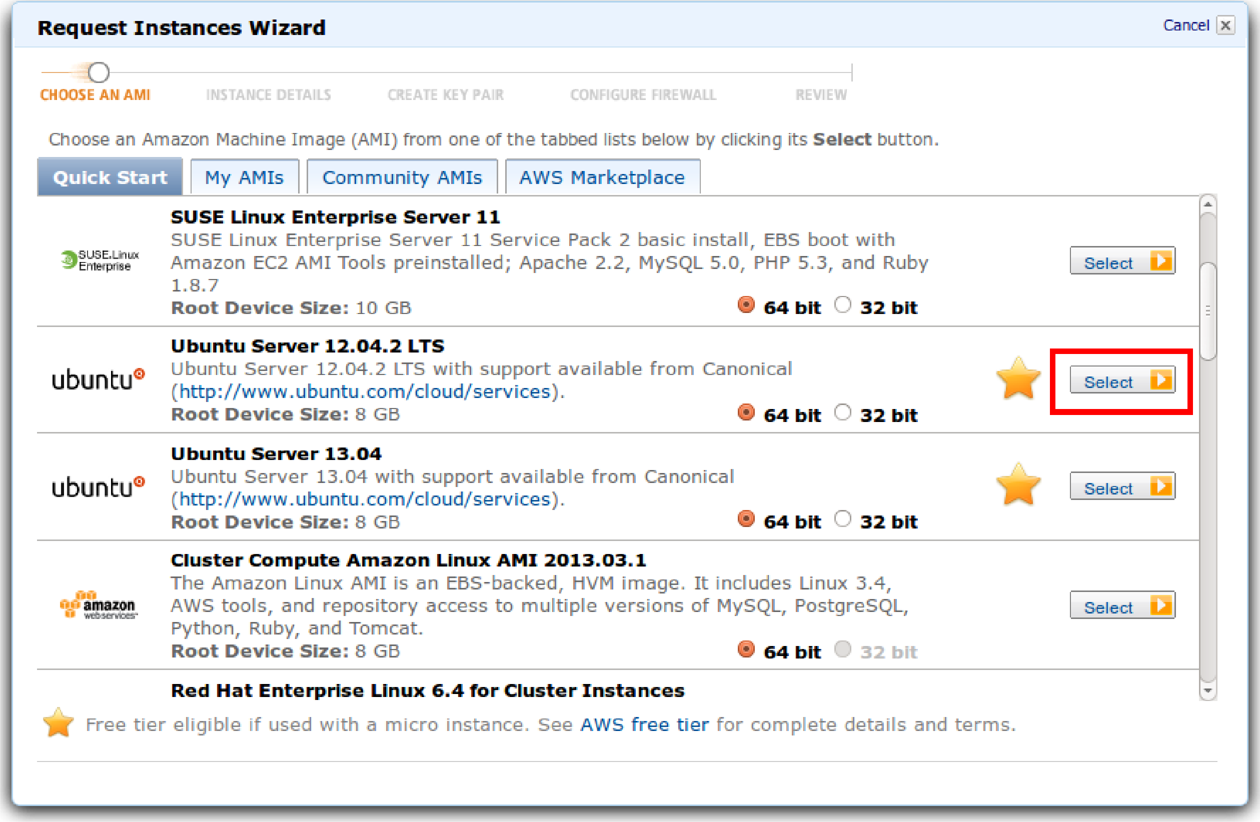
Red Hat Enterprise Linux 6.1 32 bit (AMI Id: ami-0cbb4265)
Red Hat Enterprise Linux version 6.1, EBS-boot, 32-bit architecture.
Root Device Size: 7 GB

Red Hat Enterprise Linux 6.1 64 bit (AMI Id: ami-5e837b37)
Red Hat Enterprise Linux version 6.1, EBS-boot, 64-bit architecture.
Root Device Size: 6 GB

SUSE Linux Enterprise Server 11 64-bit (AMI Id: ami-e4a3578d)
SUSE Linux Enterprise Server 11 Service Pack 1 basic install, EBS boot, 64-bit architecture with Amazon EC2 AMI Tools preinstalled; Apache 2.2, MySQL 5.0, PHP 5.3, Ruby 1.8.7, and Rails 2.3.
Root Device Size: 15 GB

Free tier eligible if used with a micro instance. See AWS free tier for complete details and terms.

Select *Classic Wizard* and press *Continue*. The next screen shows a list of AMI to load as operating system in the server. The default AMI are standalone operating systems, with no additional software installed. We want one of these so we can configure the server as our application needs. In the *Quick start* tab, select *Ubuntu Server 12.04.2 LTS*:



Request Instances Wizard

Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

SUSE Linux Enterprise Server 11
SUSE Linux Enterprise Server 11 Service Pack 2 basic install, EBS boot with Amazon EC2 AMI Tools preinstalled; Apache 2.2, MySQL 5.0, PHP 5.3, and Ruby 1.8.7
Root Device Size: 10 GB ● **64 bit** ○ **32 bit**

Ubuntu Server 12.04.2 LTS
Ubuntu Server 12.04.2 LTS with support available from Canonical (http://www.ubuntu.com/cloud/services).
Root Device Size: 8 GB ● **64 bit** ○ **32 bit**

Ubuntu Server 13.04
Ubuntu Server 13.04 with support available from Canonical (http://www.ubuntu.com/cloud/services).
Root Device Size: 8 GB ● **64 bit** ○ **32 bit**

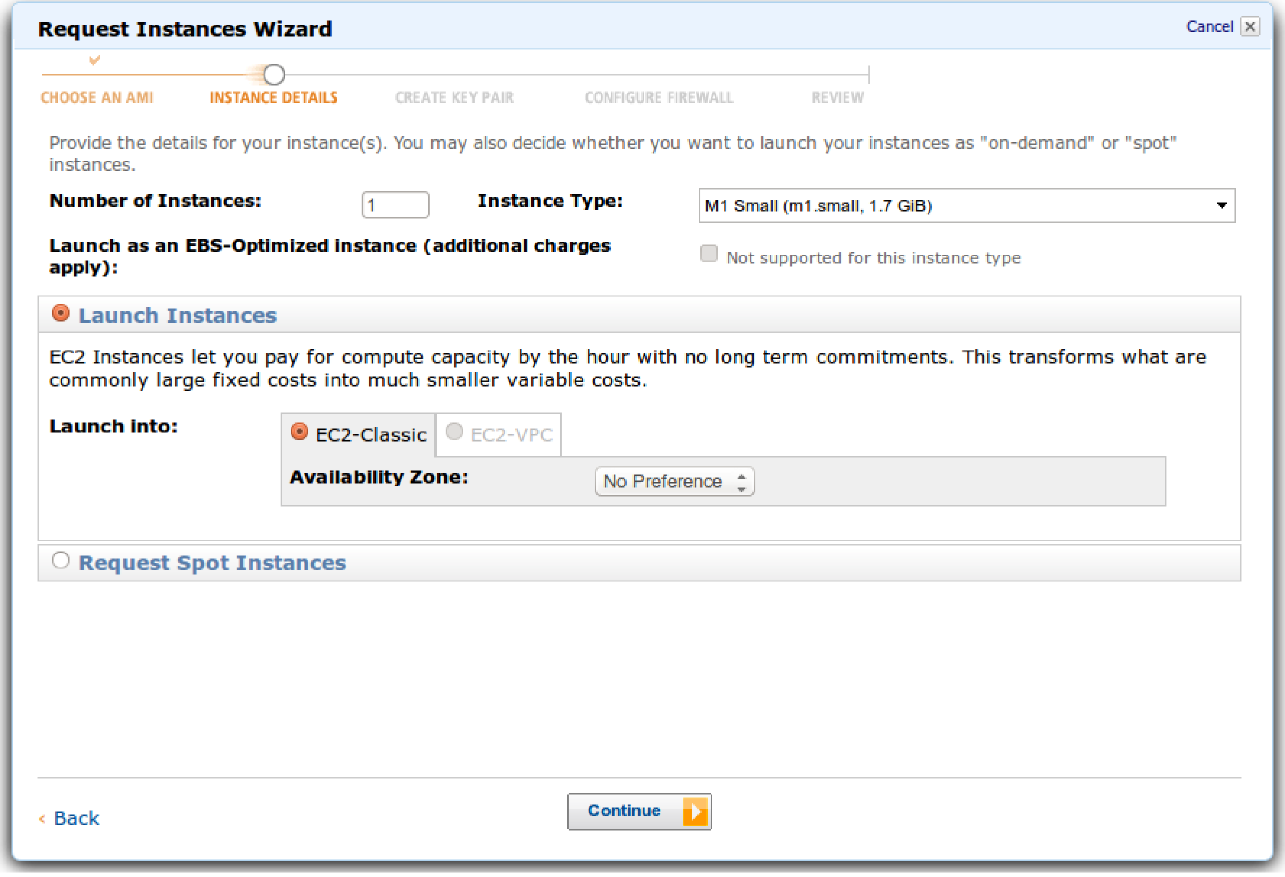
Cluster Compute Amazon Linux AMI 2013.03.1
The Amazon Linux AMI is an EBS-backed, HVM image. It includes Linux 3.4, AWS tools, and repository access to multiple versions of MySQL, PostgreSQL, Python, Ruby, and Tomcat.
Root Device Size: 8 GB ● **64 bit** ○ 32 bit

Red Hat Enterprise Linux 6.4 for Cluster Instances

Free tier eligible if used with a micro instance. See AWS free tier for complete details and terms.

This brings us to a screen where we can select how many instances we want to create and the type of instance. Currently we only want 1 instance, that we will use to generate our custom AMI. We select a *M1 small* instance type which is big enough to configure the environment.

Be aware that *t1 micro* instances, although much cheaper, are not suitable for setting up this sample application. The lack of RAM and computational power in these instances means that some operating systems, like Ubuntu, kill heavy computation tasks after a certain period of time. As a result you can't build a Play Framework application in a *micro* instance, as the *dist* process is terminated by the operating system before finishing:



Provide the details for your instance(s). You may also decide whether you want to launch your instances as "on-demand" or "spot" instances.

Number of Instances: 1 **Instance Type:** M1 Small (m1.small, 1.7 GiB)

Launch as an EBS-Optimized instance (additional charges apply): □ Not supported for this instance type

● **Launch Instances**

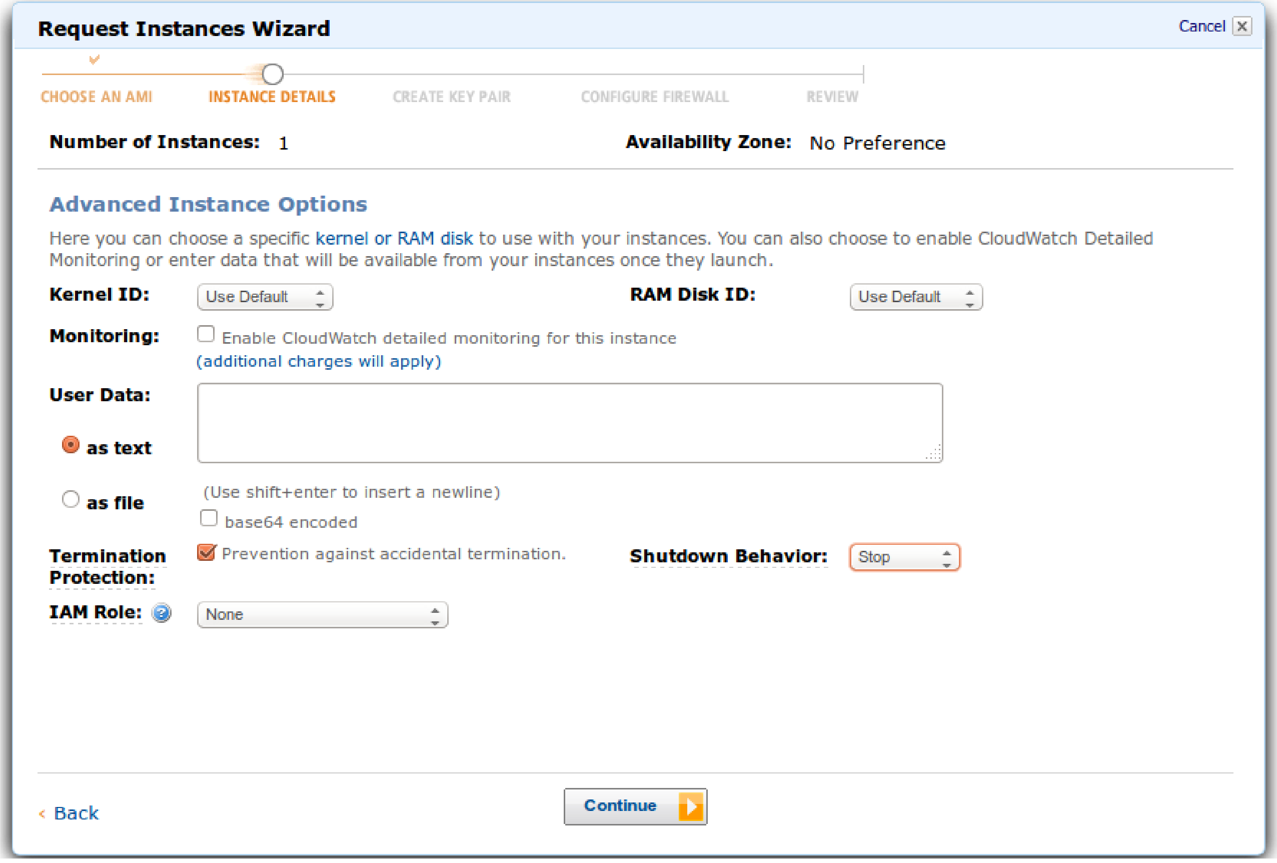
EC2 Instances let you pay for compute capacity by the hour with no long term commitments. This transforms what are commonly large fixed costs into much smaller variable costs.

Launch into: ● EC2-Classic ○ EC2-VPC

Availability Zone: No Preference

○ **Request Spot Instances**

The following screen provides some more instance details. We activate *Termination Protection* as once an instance is terminated, all its contents are lost. This avoids costly mistakes. We leave the default options for the other fields in this example, but you may want to read about *IAM Role* in your deployment to better safeguard access to the instance:



Request Instances Wizard Cancel ☒

Number of Instances: 1 **Availability Zone:** No Preference

Advanced Instance Options

Here you can choose a specific kernel or RAM disk to use with your instances. You can also choose to enable CloudWatch Detailed Monitoring or enter data that will be available from your instances once they launch.

Kernel ID: Use Default **RAM Disk ID:** Use Default

Monitoring: □ Enable CloudWatch detailed monitoring for this instance (additional charges will apply)

User Data: ● as text ○ as file

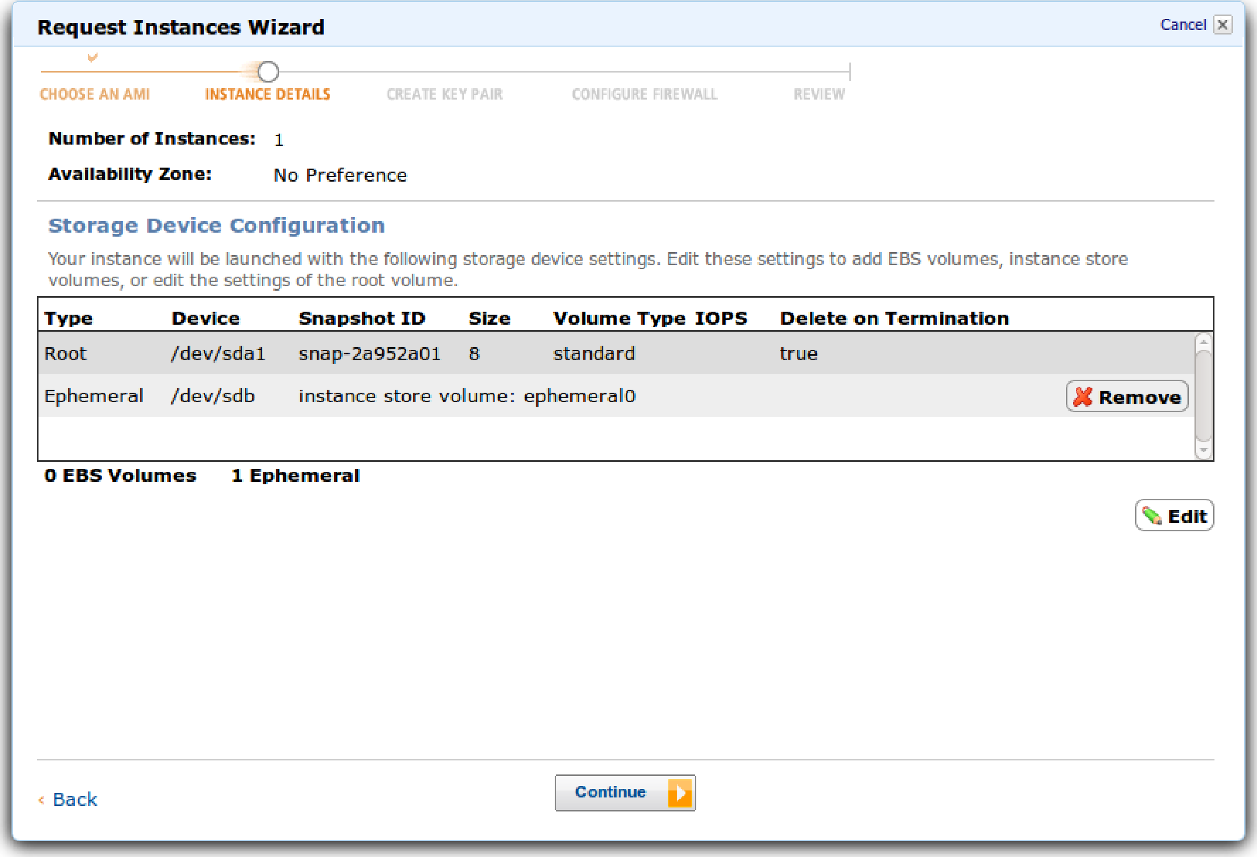
(Use shift+enter to insert a newline)

□ base64 encoded

Termination Protection: ☑ Prevention against accidental termination. **Shutdown Behavior:** Stop

IAM Role: None

The next screen contains more instance details of which we accept the defaults:



Request Instances Wizard Cancel ☒

Number of Instances: 1

Availability Zone: No Preference

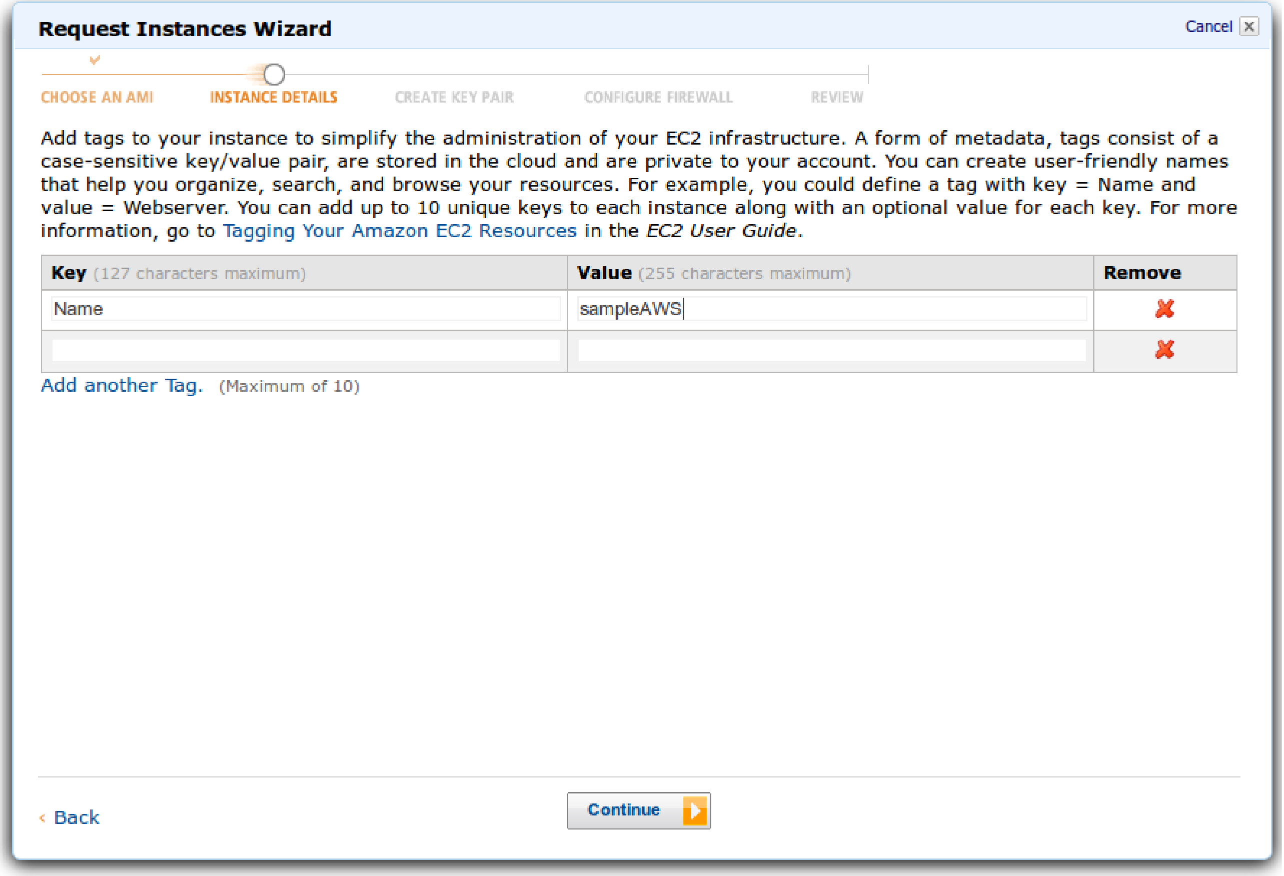
Storage Device Configuration

Your instance will be launched with the following storage device settings. Edit these settings to add EBS volumes, instance store volumes, or edit the settings of the root volume.

Type	Device	Snapshot ID	Size	Volume Type	IOPS	Delete on Termination	
Root	/dev/sda1	snap-2a952a01	8	standard		true	
Ephemeral	/dev/sdb	instance store volume: ephemeral0					✗ Remove

0 EBS Volumes **1 Ephemeral**

And in the following window we add a name to identify the instance:

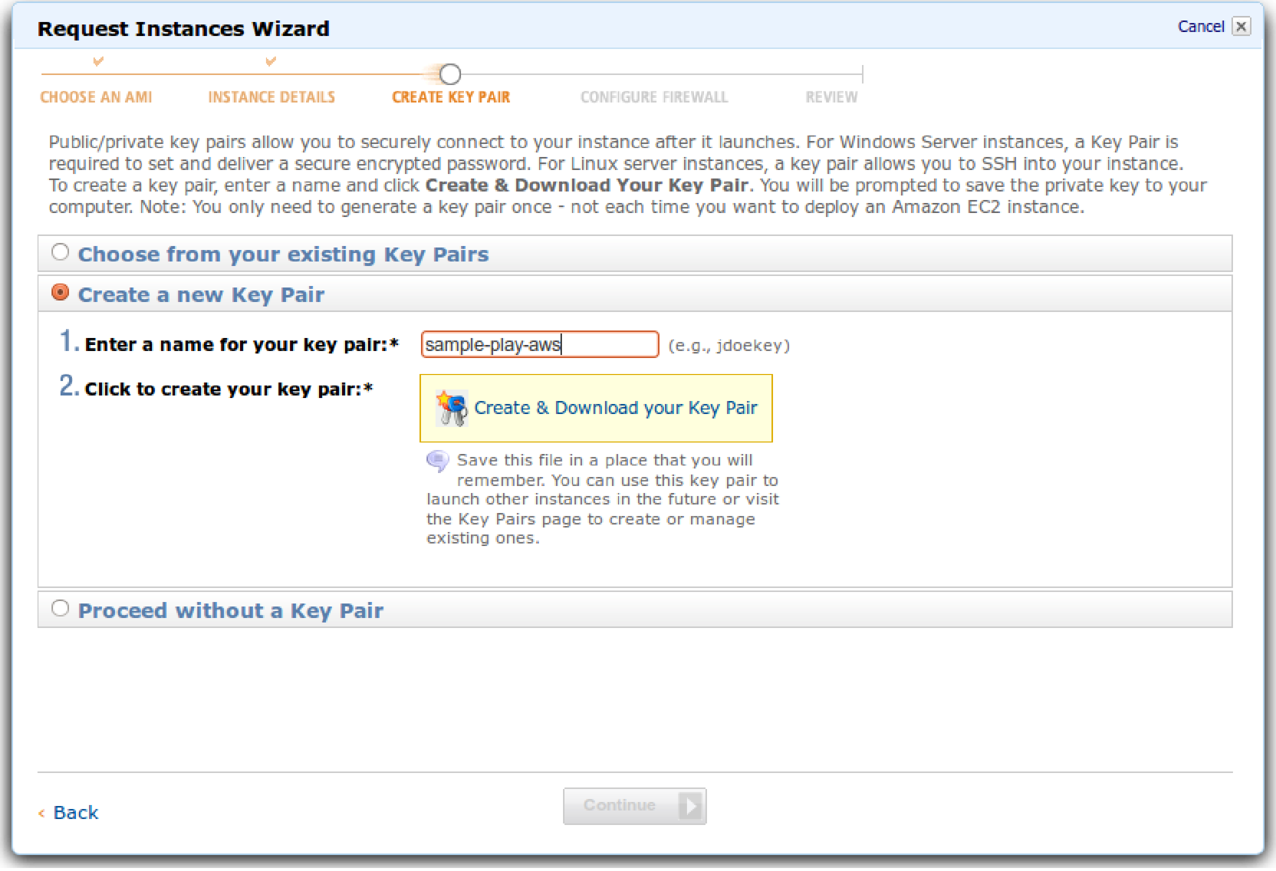


Add tags to your instance to simplify the administration of your EC2 infrastructure. A form of metadata, tags consist of a case-sensitive key/value pair, are stored in the cloud and are private to your account. You can create user-friendly names that help you organize, search, and browse your resources. For example, you could define a tag with key = Name and value = Webserver. You can add up to 10 unique keys to each instance along with an optional value for each key. For more information, go to Tagging Your Amazon EC2 Resources in the *EC2 User Guide*.

Key (127 characters maximum)	**Value** (255 characters maximum)	**Remove**
Name	sampleAWS	✗
		✗

Add another Tag. (Maximum of 10)

This brings us to a very important step, the creation of they key pair values. In this step we create the key pair that is used by *SSH* to connect to this instance. Without it, we are not able to log into it nor to execute the Ansible scripts to configure the server. If you have another Amazon key pair, you can select it as the one to use. Otherwise, as in the current example, you can create a new key pair by giving it a name and pressing *Create and Download*:



Public/private key pairs allow you to securely connect to your instance after it launches. For Windows Server instances, a Key Pair is required to set and deliver a secure encrypted password. For Linux server instances, a key pair allows you to SSH into your instance. To create a key pair, enter a name and click **Create & Download Your Key Pair**. You will be prompted to save the private key to your computer. Note: You only need to generate a key pair once - not each time you want to deploy an Amazon EC2 instance.

○ Choose from your existing Key Pairs

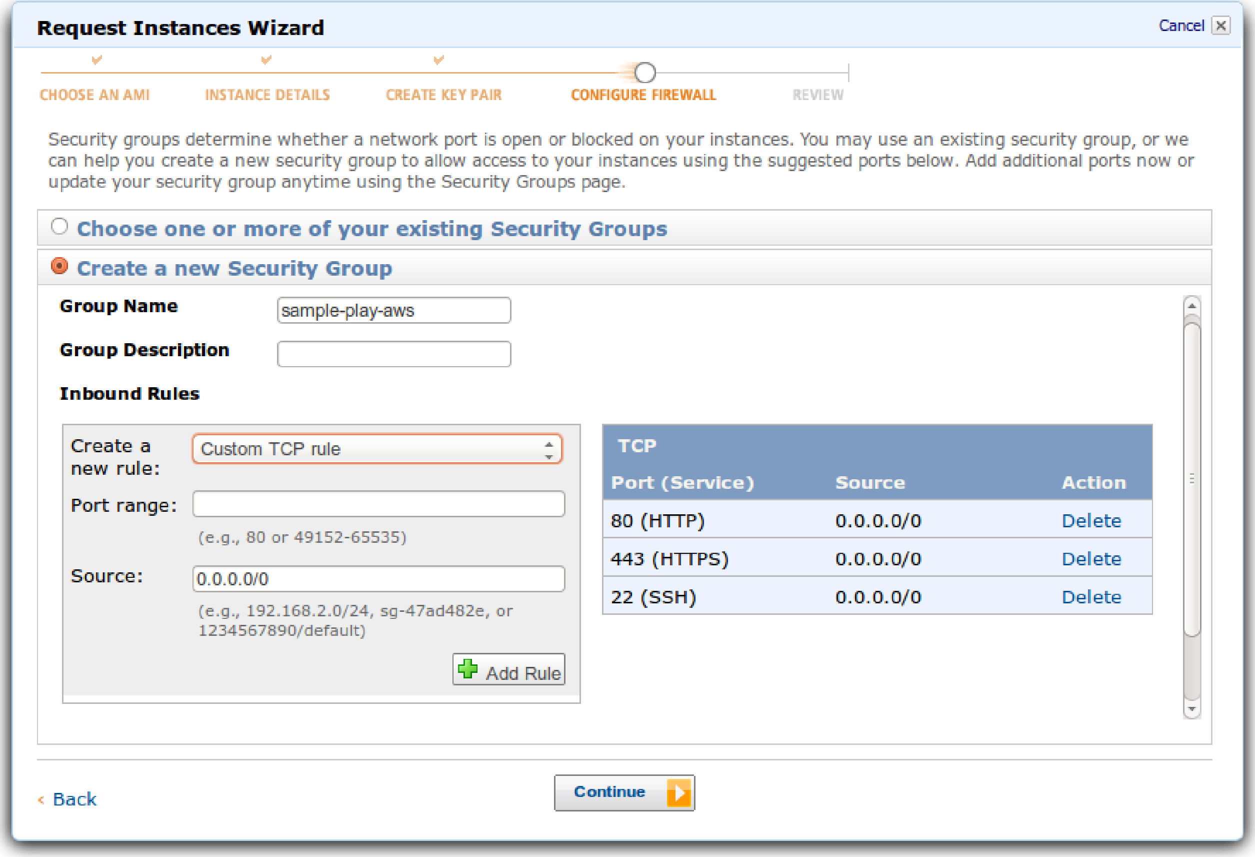
● Create a new Key Pair

1. Enter a name for your key pair:* sample-play-aws (e.g., jdoekey)
2. Click to create your key pair:* Create & Download your Key Pair

Save this file in a place that you will remember. You can use this key pair to launch other instances in the future or visit the Key Pairs page to create or manage existing ones.

○ Proceed without a Key Pair

Once generated and downloaded, the wizard displays another very important screen, the firewall settings. In this screen you can select an existing security group or, as in this example, create your own. We create a new group called *sample-play-aws* which allows any SSH, HTTP and HTTPS connections into the machine. This way we can run Ansible scripts and connect to the server once the application is running:



Request Instances Wizard

Security groups determine whether a network port is open or blocked on your instances. You may use an existing security group, or we can help you create a new security group to allow access to your instances using the suggested ports below. Add additional ports now or update your security group anytime using the Security Groups page.

○ Choose one or more of your existing Security Groups

● Create a new Security Group

Group Name sample-play-aws

Group Description

Inbound Rules

Create a new rule: Custom TCP rule

Port range:

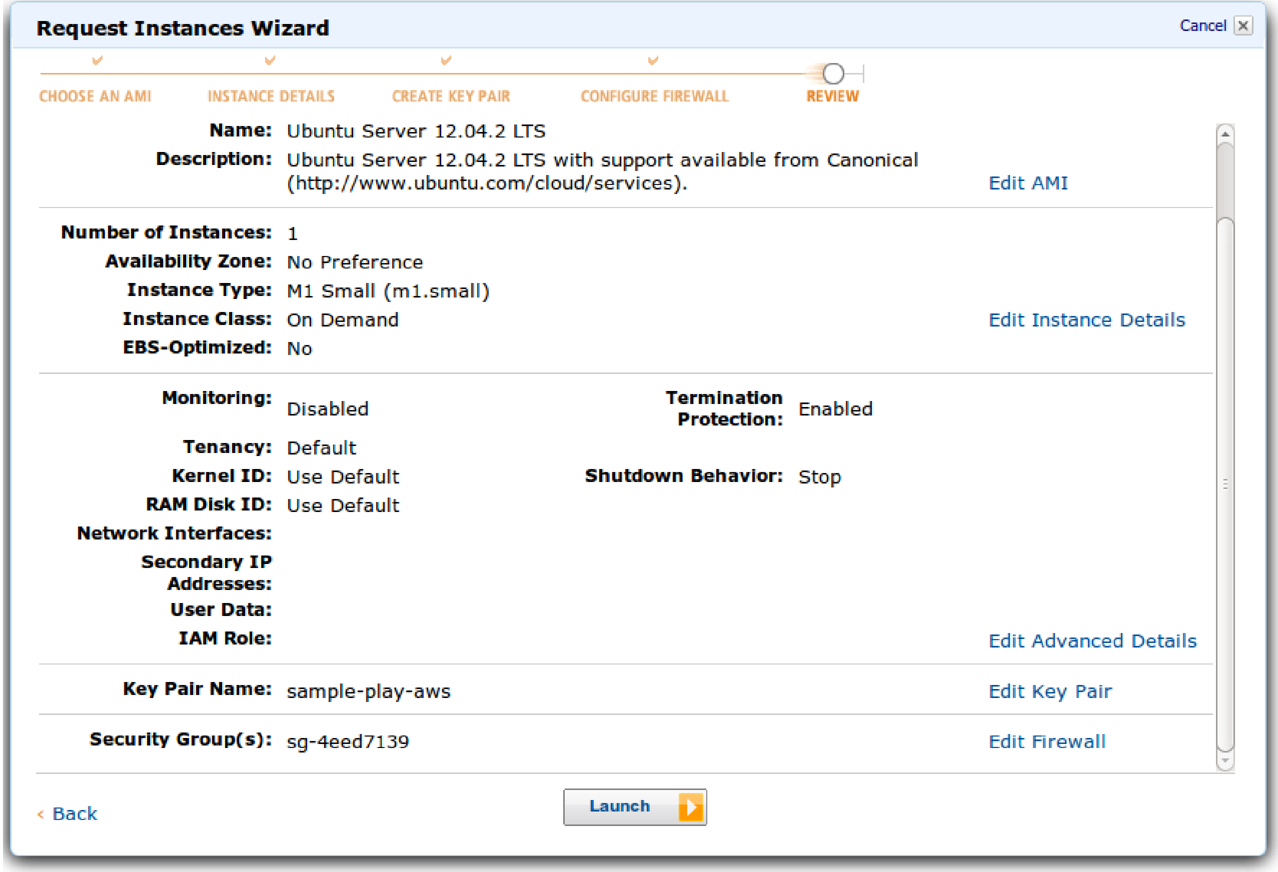
(e.g., 80 or 49152-65535)

Source: 0.0.0.0/0

(e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

TCP		
Port (Service)	Source	Action
80 (HTTP)	0.0.0.0/0	Delete
443 (HTTPS)	0.0.0.0/0	Delete
22 (SSH)	0.0.0.0/0	Delete

The last screen of the wizard shows a summary of the options selected and allows us to modify them before launching the instance itself:



Request Instances Wizard

Name: Ubuntu Server 12.04.2 LTS
Description: Ubuntu Server 12.04.2 LTS with support available from Canonical (http://www.ubuntu.com/cloud/services).

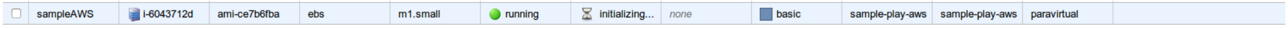
Number of Instances: 1
Availability Zone: No Preference
Instance Type: M1 Small (m1.small)
Instance Class: On Demand
EBS-Optimized: No

Monitoring: Disabled
Termination Protection: Enabled
Tenancy: Default
Kernel ID: Use Default
Shutdown Behavior: Stop
RAM Disk ID: Use Default
Network Interfaces:
Secondary IP Addresses:
User Data:
IAM Role:

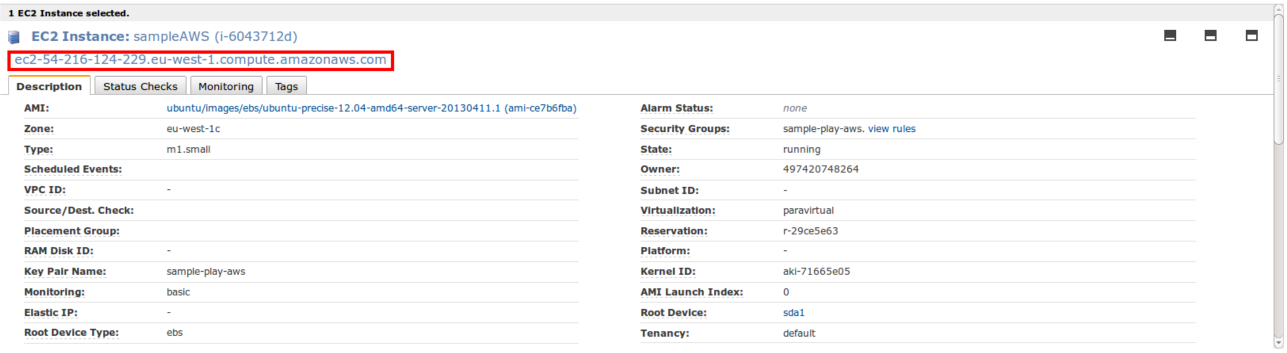
Key Pair Name: sample-play-aws

Security Group(s): sg-4eed7139

Once the instance is running you can see an entry in the console with its name, status *Running* and the address of the instance as shown in the following image:



□	sampleAWS	i-6043712d	ami-ce7b6fba	ebs	m1.small	● running	initializing...	*none*	basic	sample-play-aws	sample-play-aws	paravirtual



1 EC2 Instance selected.

EC2 Instance: sampleAWS (i-6043712d)

ec2-54-216-124-229.eu-west-1.compute.amazonaws.com

AMI:	ubuntu/images/ebs/ubuntu-precise-12.04-amd64-server-20130411.1 (ami-ce7b6fba)	**Alarm Status:**	*none*
Zone:	eu-west-1c	**Security Groups:**	sample-play-aws. view rules
Type:	m1.small	**State:**	running
Scheduled Events:		**Owner:**	497420748264
VPC ID:	-	**Subnet ID:**	-
Source/Dest. Check:		**Virtualization:**	paravirtual
Placement Group:		**Reservation:**	r-29ce5e63
RAM Disk ID:	-	**Platform:**	-
Key Pair Name:	sample-play-aws	**Kernel ID:**	aki-71665e05
Monitoring:	basic	**AMI Launch Index:**	0
Elastic IP:	-	**Root Device:**	sda1
Root Device Type:	ebs	**Tenancy:**	default

At this stage, your instance is ready to be configured using the Ansible scripts.

Execute Ansible scripts

To run the scripts you need to have Ansible installed. If you have not installed it yet, please follow the instructions corresponding to your development environment. Ansible 1.2 is required to run the scripts in this sample.

The first step to run Ansible is to copy the IP of the EC2 instance to the corresponding hosts files. In this scenario, edit *testing.hosts* and copy the IP under the *ireland-webservers* entry, as follows:

```
# Contains the addresses of all EC2 instances used in testing in each region

# Webservers in Ireland region
[ireland-webservers]
ec2-54-216-124-229.eu-west-1.compute.amazonaws.com
```

You can edit *production.hosts* and *development.hosts* in a similar way if you are configuring those environments.

The next step is to set the key pair downloaded in the previous section for use in SSH. First we have to modify the attributes of the key to be read only, and then we add it to SSH via *ssh-add*:

```
$chmod 600 sample-play-aws.pem
$ssh-add sample-play-aws.pem
```

Ensure that all the Ansible configuration is as expected. Check the following files and update their values if needed:

- **group_vars/all:** set the proper configuration details, including your email and deployment paths
- **roles/webtier/files/start:** add the proper values to connect to the RDS db and AWS credentials via *-D* JVM properties

At this point we are ready to run the scripts. Go to the folder where you have deployed your Ansible files and execute the following command:

```
$ansible-playbook -i testing.hosts site.yml -u ubuntu --sudo
```

The script configures the server executing the Playbooks described above. It takes several minutes to run, and once it finishes you can access the deployed application at:

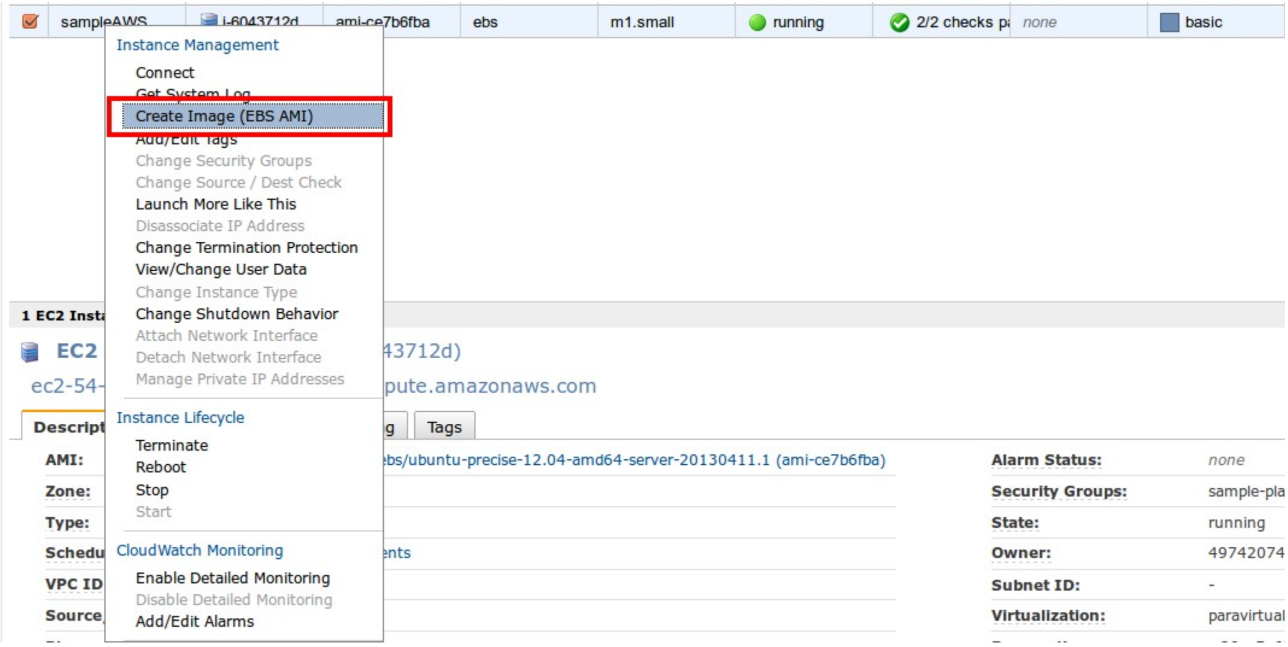
http://<amazon_ec2_ip>/

You should see the main page of the application loading on the browser.

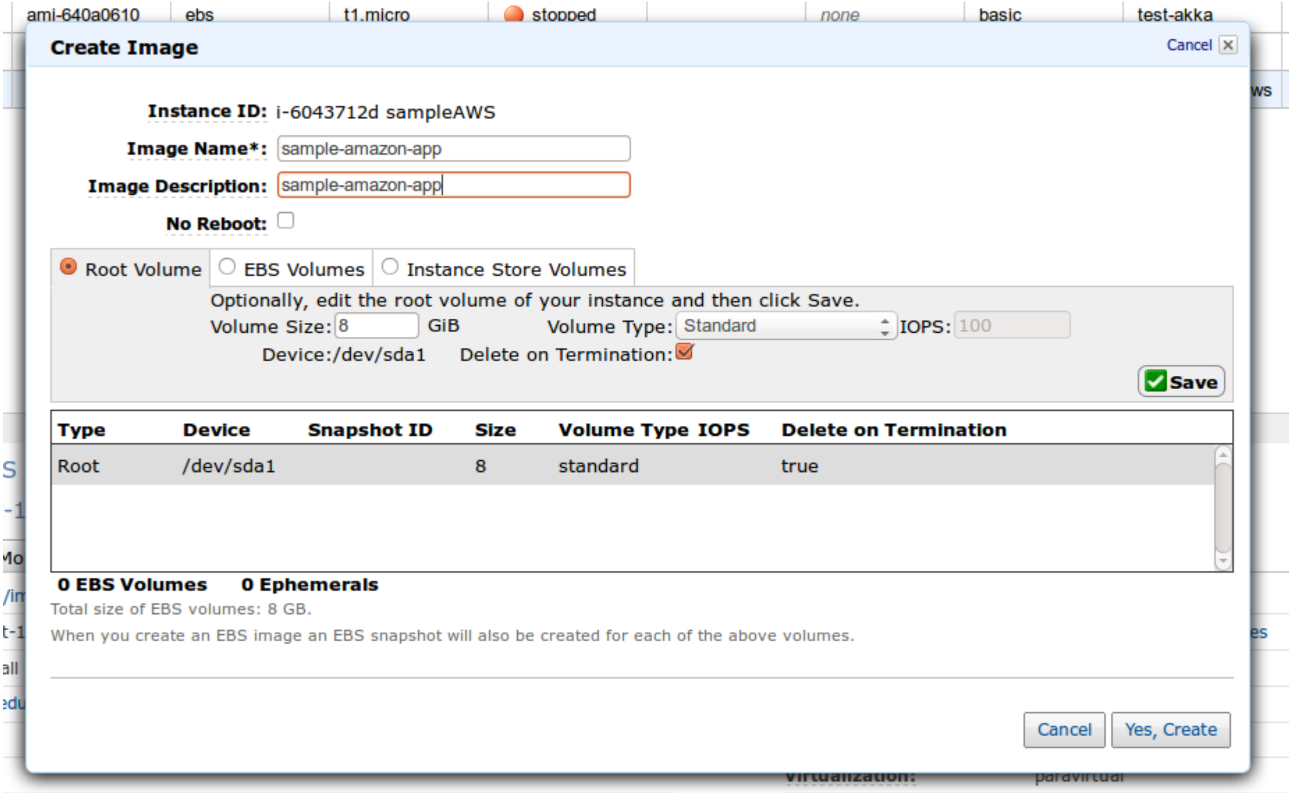
With the server configured and the application deployed, we are ready to create the custom AMI for our environment.

Create custom AMI

Creating a custom AMI is very simple. Go back to the Amazon EC2 console and right click on the instance name. A menu to manage the instance opens:



Select the *Create Image (EBS AMI)* option. This opens a window in which you can give a name to your AMI and select which volumes to export. In this example we export the *Root* volume that we have configured with Ansible:



Instance ID: i-6043712d sampleAWS

Image Name*: sample-amazon-app

Image Description: sample-amazon-app

No Reboot: □

● Root Volume ○ EBS Volumes ○ Instance Store Volumes

Optionally, edit the root volume of your instance and then click Save.

Volume Size: 8 GiB Volume Type: Standard IOPS: 100

Device:/dev/sda1 Delete on Termination: ☑

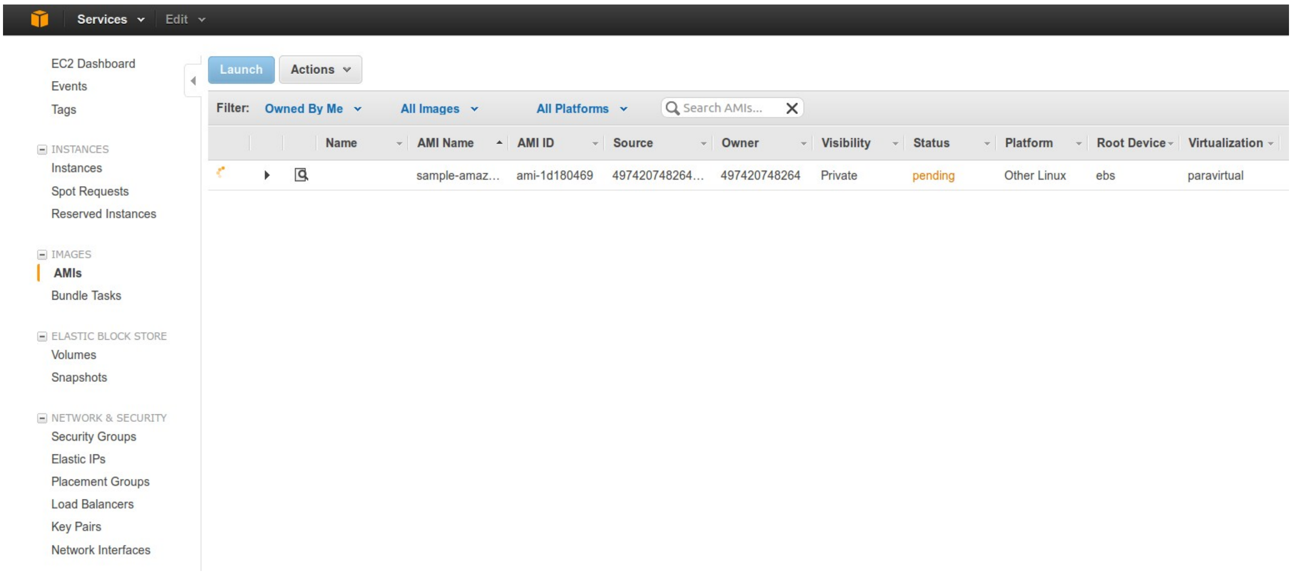
Type	Device	Snapshot ID	Size	Volume Type	IOPS	Delete on Termination
Root	/dev/sda1		8	standard		true

0 EBS Volumes 0 Ephemerals

Total size of EBS volumes: 8 GB.

When you create an EBS image an EBS snapshot will also be created for each of the above volumes.

After pressing *Yes, Create,* AWS starts the process to create the AMI. This takes a while. You can see the AMI in the console by selecting the *Images* > *AMIs* menu:



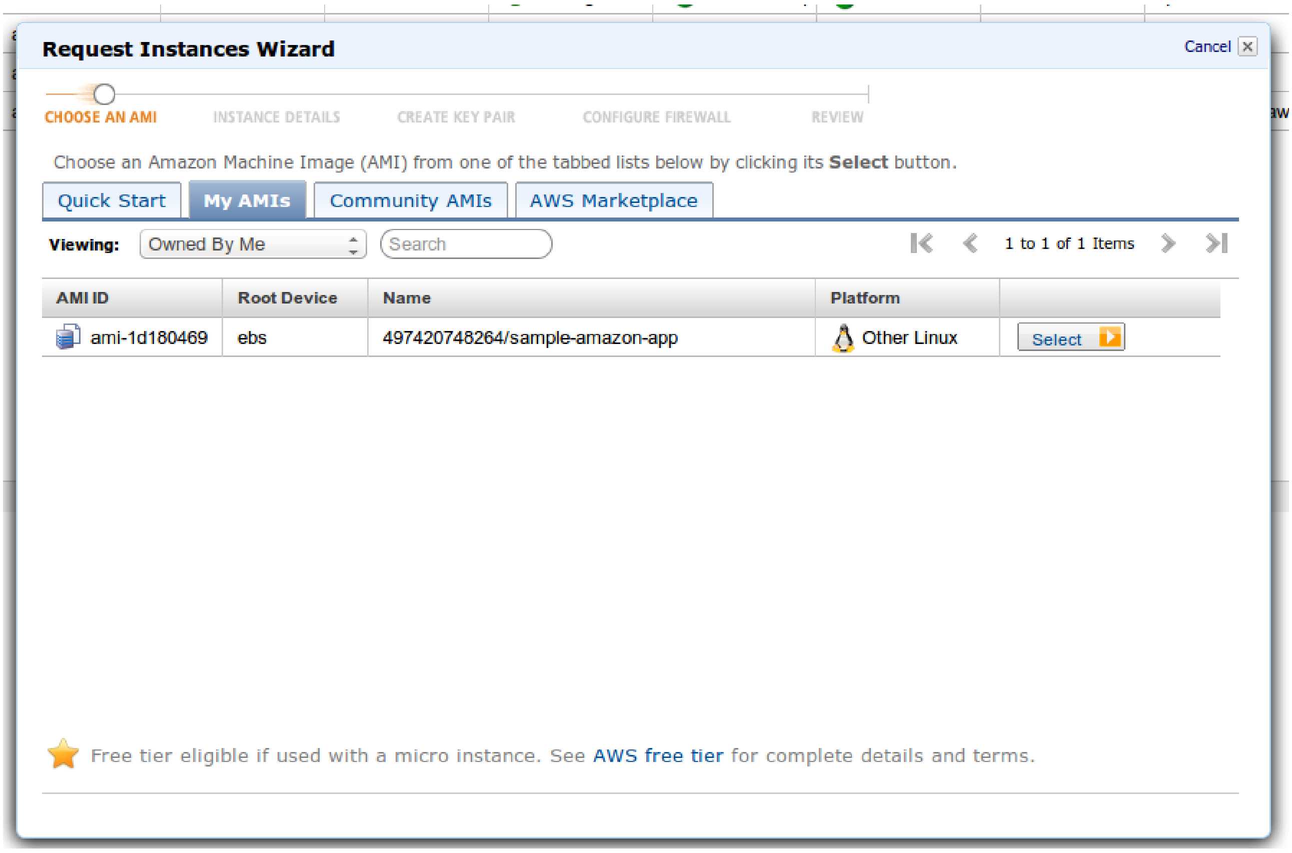
Filter: Owned By Me | All Images | All Platforms

Name	AMI Name	AMI ID	Source	Owner	Visibility	Status	Platform	Root Device	Virtualization
	sample-amaz...	ami-1d180469	497420748264...	497420748264	Private	pending	Other Linux	ebs	paravirtual

Once the creation process finishes the status will change to *Complete (green)* and at that moment you can use this AMI on your deployments.

Deploy other instances

As mentioned above, you can create more instances using Ansible. An alternative is to create a new instance using the custom AMI generated in the previous step. To do this, create a new instance as mentioned before, but when choosing the AMI click on the tab *My AMIs*. There you can see a list of your custom AMIs:



Request Instances Wizard

Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

Viewing: Owned By Me

1 to 1 of 1 Items

AMI ID	Root Device	Name	Platform	
ami-1d180469	ebs	497420748264/sample-amazon-app	Other Linux	Select

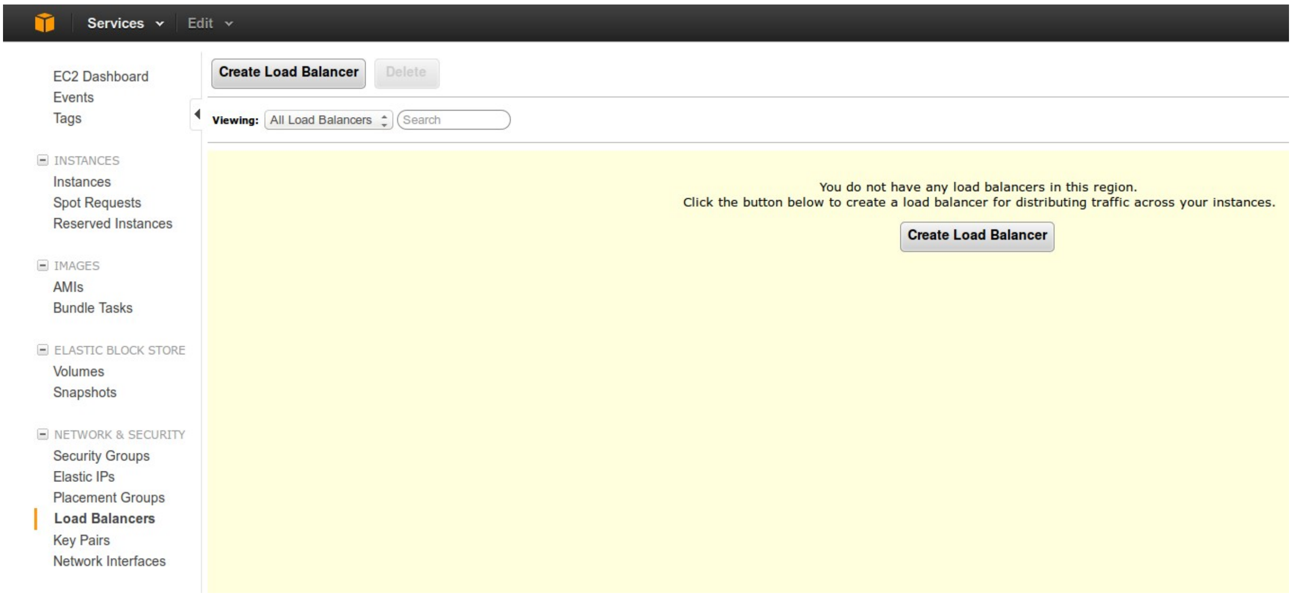
Free tier eligible if used with a micro instance. See AWS free tier for complete details and terms.

Choose the AMI you created and complete the other steps of the wizard as indicated before. Remember that this new EC2 instance already contains the server configuration and the application deployment, as specified by the Ansible scripts we executed, so there is no need to run Ansible again on this host.

Configure the Load Balancer

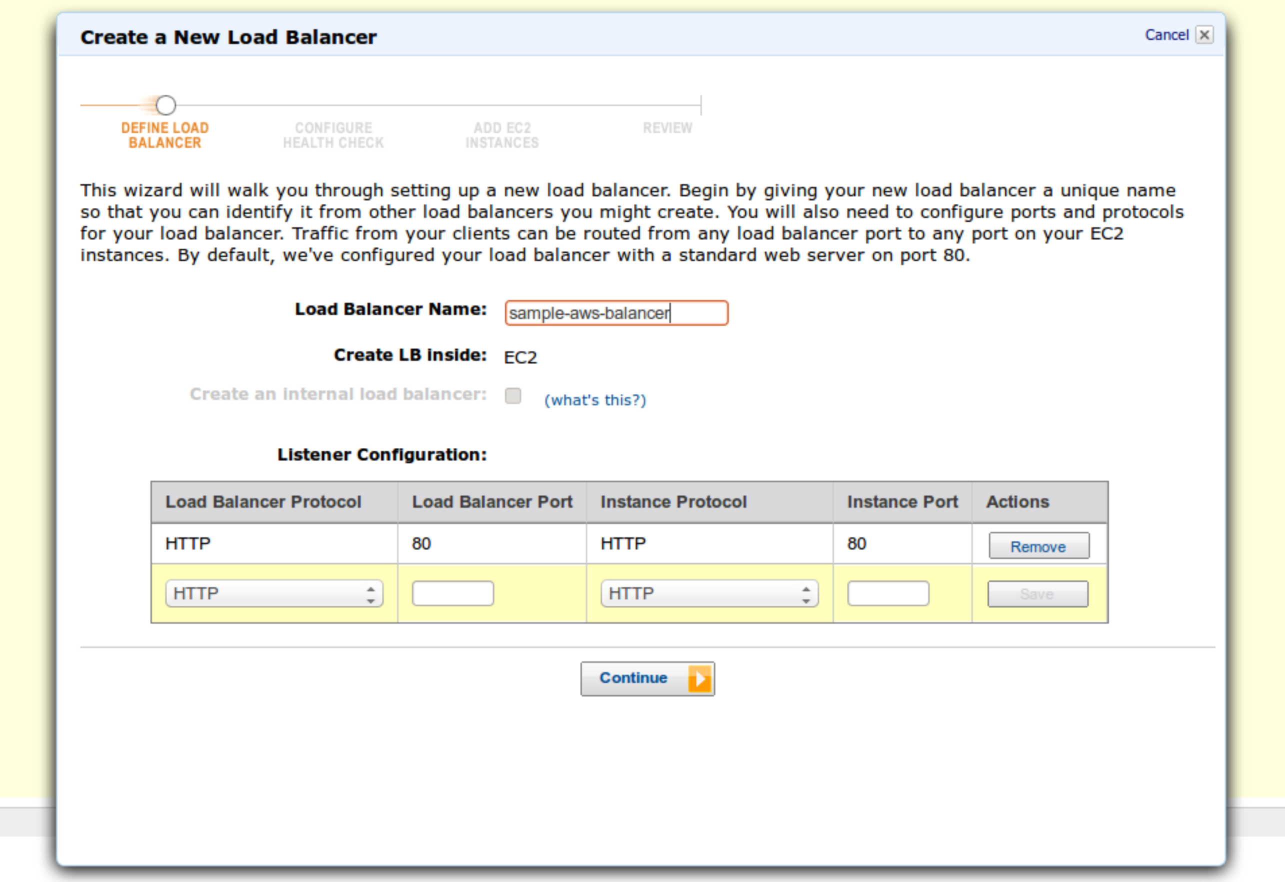
Amazon Elastic Load Balancing is an AWS component that distributes the load evenly along your EC2 instances. Not only that, the balancer detects the health of the associated instances and it can restart faulty instances, ensuring high availability for your application.

The Load Balancer can be created from the EC2 Console by selecting the *Network & Security > Load Balancers* menu:



You do not have any load balancers in this region.
Click the button below to create a load balancer for distributing traffic across your instances.

The first window of the wizard allows you to select which protocols are managed by the balancer and the mappings between the ports accessible via the internet and the internal ports accessible in your EC2 instances. In this sample, we only want to map the port 80, redirecting all the requests to the corresponding port 80 of one of our running instances:



Create a New Load Balancer

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer Name: sample-aws-balancer

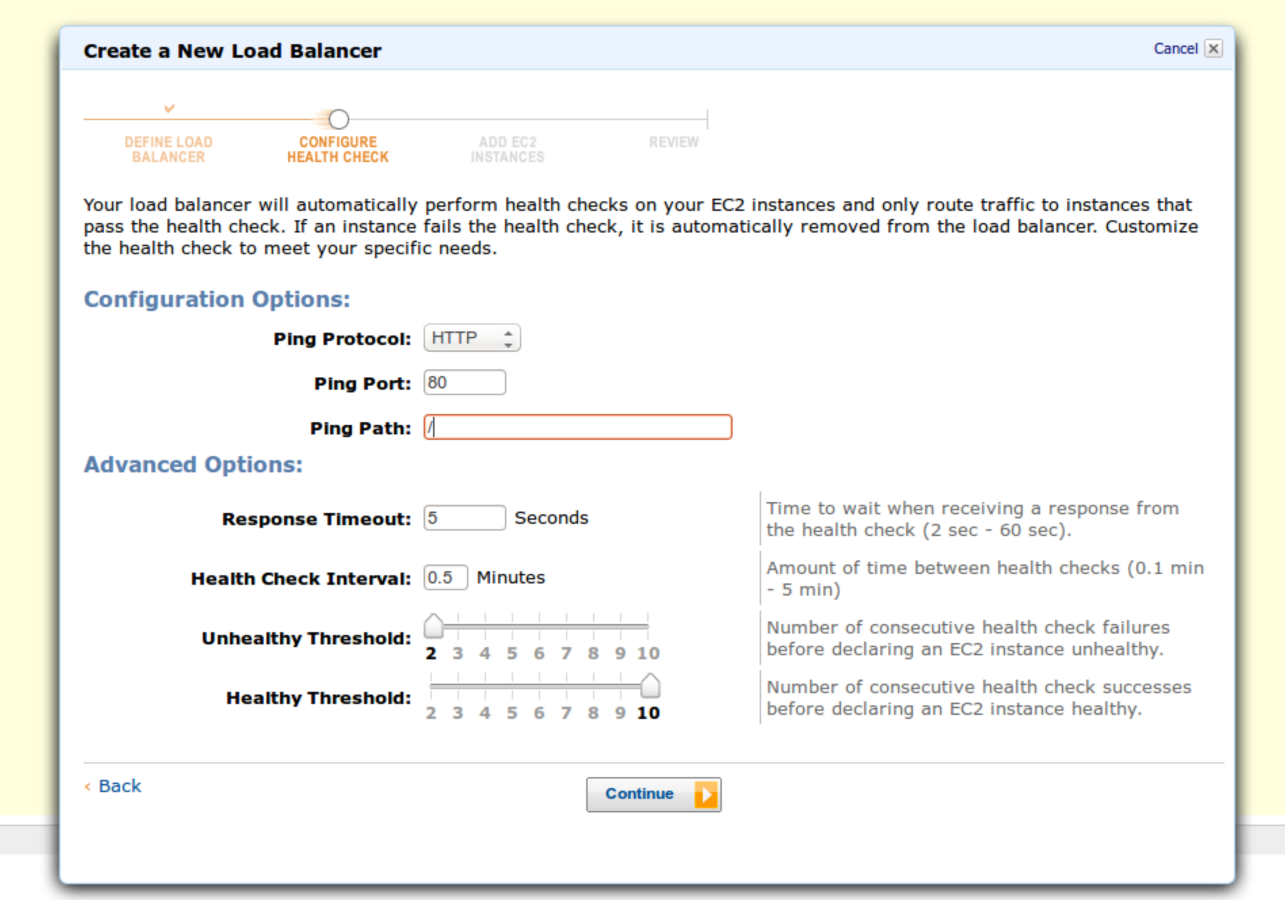
Create LB inside: EC2

Create an internal load balancer: □ (what's this?)

Listener Configuration:

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port	Actions
HTTP	80	HTTP	80	Remove
HTTP		HTTP		Save

The next step sets the *Health Check* parameters, including which protocol, port and path has to be used for the checks and the thresholds to consider an instance as healthy or unhealthy. Unhealthy instances are automatically removed from the balancer:



Create a New Load Balancer

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Configuration Options:

Ping Protocol: HTTP

Ping Port: 80

Ping Path: /

Advanced Options:

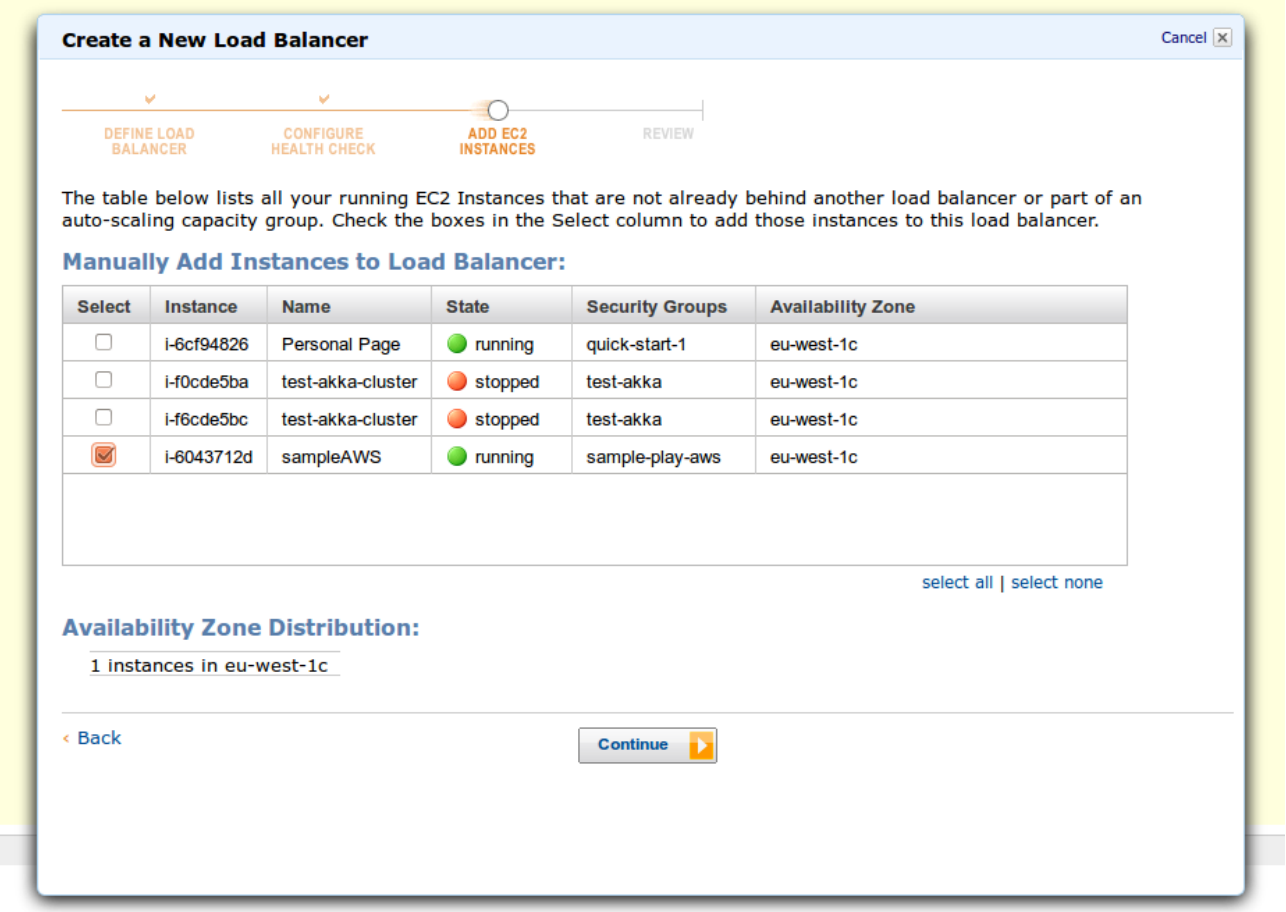
Response Timeout: 5 Seconds — Time to wait when receiving a response from the health check (2 sec - 60 sec).

Health Check Interval: 0.5 Minutes — Amount of time between health checks (0.1 min - 5 min)

Unhealthy Threshold: **2** 3 4 5 6 7 8 9 10 — Number of consecutive health check failures before declaring an EC2 instance unhealthy.

Healthy Threshold: 2 3 4 5 6 7 8 9 **10** — Number of consecutive health check successes before declaring an EC2 instance healthy.

After pressing *Continue*, we can select the instances to be managed by the Load Balancer:



Create a New Load Balancer

The table below lists all your running EC2 Instances that are not already behind another load balancer or part of an auto-scaling capacity group. Check the boxes in the Select column to add those instances to this load balancer.

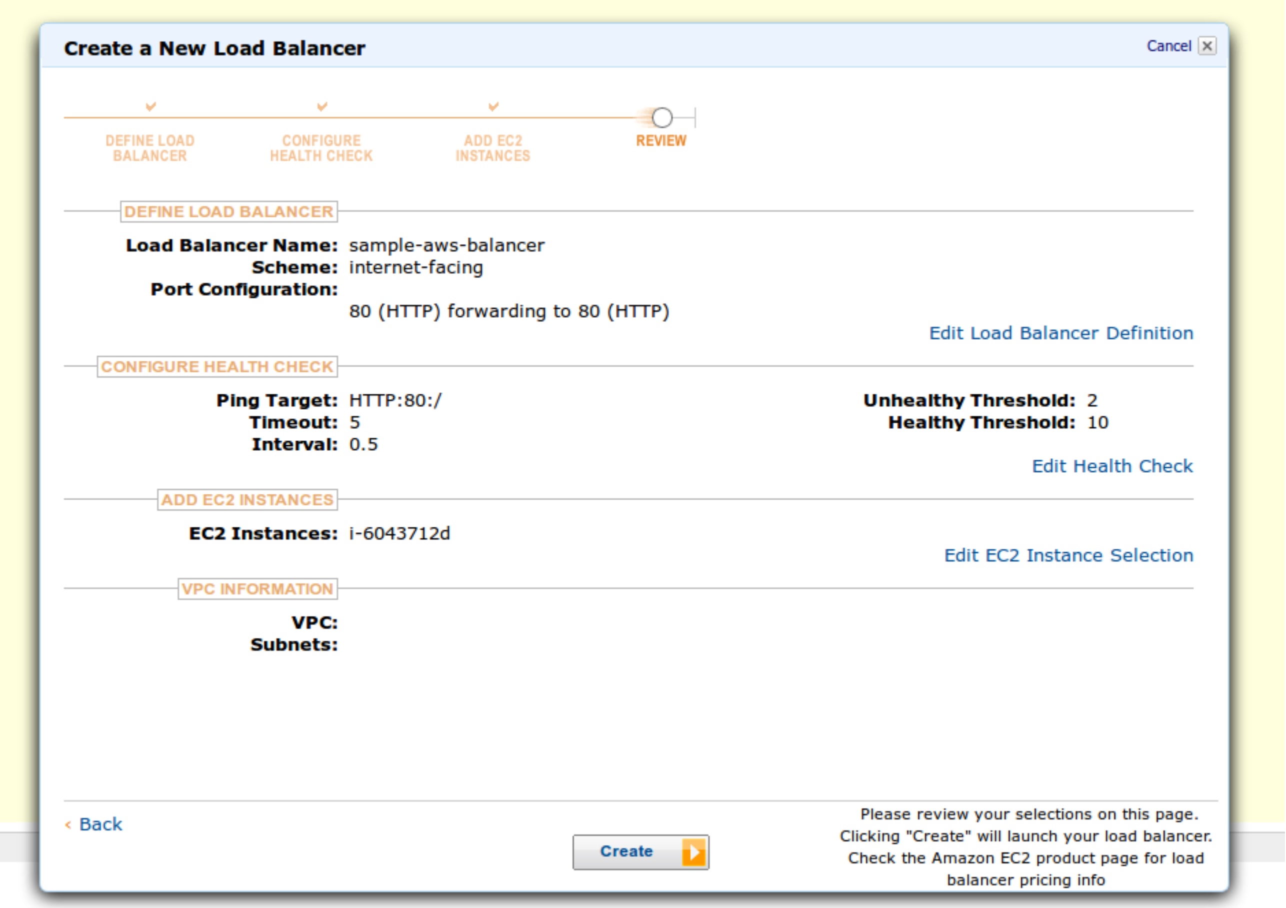
Manually Add Instances to Load Balancer:

Select	Instance	Name	State	Security Groups	Availability Zone
□	i-6cf94826	Personal Page	● running	quick-start-1	eu-west-1c
□	i-f0cde5ba	test-akka-cluster	● stopped	test-akka	eu-west-1c
□	i-f6cde5bc	test-akka-cluster	● stopped	test-akka	eu-west-1c
☑	i-6043712d	sampleAWS	● running	sample-play-aws	eu-west-1c

Availability Zone Distribution:

1 instances in eu-west-1c

The last screen shows a summary of the values selected for the Load Balancer:



Create a New Load Balancer

DEFINE LOAD BALANCER

Load Balancer Name: sample-aws-balancer
Scheme: internet-facing
Port Configuration:
80 (HTTP) forwarding to 80 (HTTP)

CONFIGURE HEALTH CHECK

Ping Target: HTTP:80:/
Timeout: 5
Interval: 0.5

Unhealthy Threshold: 2
Healthy Threshold: 10

ADD EC2 INSTANCES

EC2 Instances: i-6043712d

VPC INFORMATION

VPC:
Subnets:

Please review your selections on this page. Clicking "Create" will launch your load balancer. Check the Amazon EC2 product page for load balancer pricing info

At this stage we have created a load balancer that will distribute the load evenly across our selected EC2 instances. We can manually add or remove instances to increase the capacity of our application and having total control of the costs.

Having created the Elastic Load Balancer, you can obtain its public Elastic IP. This allows you to map the balancer to a custom domain via Route 53 or your own DNS management system.

Manual creation is fine for a testing or development environment, but in production we want to create as many instances as we need, on demand, to automatically scale the application and resolve all the requests. The Elastic Load Balancer we configured doesn't allow it, but the next sections explain how to achieve this using AWS.

Autoscaling the Application

Scaling based on demand is something basic for a production environment. This gives an application the capacity to answer all the requests received, while saving money in off-peak times by stopping unused instances.

Amazon AWS allows you to define an Autoscaling Group which can be used for this purpose. Unfortunately, there is no GUI to create it. You have to download the Auto Scaling Command Line Tool and follow the instructions to install the tool in your environment. After that, a set of commands have to be executed to enable Scaling based on demand.

The following sections show the main steps in the process, but please read the Scaling based on demand instructions to have a full understanding of the steps involved.

Creating the Autoscaling Group

An Autoscaling Group facilitates autoscaling of your application by deploying new instances using a preselected AMI. To create the autoscaling group you need the id of your custom AMI. You can find it in the AMI section of the EC2 console. In this section we refer to that id as *ami-id*.

The first step is to create the *launch configuration*. A *launch configuration* defines the instance type to create and the AMI to use for autoscaling. If a new EC2 instance is needed, AWS reads the *launch configuration* and creates a new EC2 instance of the given type using the selected AMI. To create your *launch configuration*, execute:

```
  $as-create-launch-config MyLaunchConfig --image-id ami-id --instance-type m1.small
```

The next step is to define the *autoscaling group*. The group defines the number of instances we want executing our application (minimum and maximum), the AWS region in which we want them and the *launch configuration* to use when spinning new instances. We can also link the group to an Elastic Load Balancer, to ensure that all the instances created are accessed via a common entry point. In our example we want to link the new group to the Load Balancer created previously, this way the new instances are accessible via our custom domain.

To create the group, execute:

```
  $as-create-auto-scaling-group MyGroup --launch-configuration MyLaunchConfig
--availability-zones "eu-west-1" --min-size 1 --max-size 3 –desired-capacity 2 --load-balancers
sample-aws—balancer
```

The command indicates AWS that we create the instances in *eu-west-1* and we link them to the load balancer defined in the previous section using its name as the reference (*sample-aws-balancer*). We notify AWS that we want between 1 and 3 instances, with a desired capacity of 2. This means that we will always have at least one instance running the application and, if the demand requires it, we can have up to three *m1.small* EC2 instances serving requests.

The last step is to create *scaling policies* that manage the resources. For example, we can create a policy that increases the capacity of our application by 30% and it is linked to the previous *autoscaling group* via the command:

```
   $as-put-scaling-policy my-scaleout-policy -–auto-scaling-group MyGroup --adjustment=30
--type PercentChangeInCapacity
```

Each scaling policy command returns the ARN associated to the policy, for example:

```
arn:aws:autoscaling:eu-west-1:123456789012:scalingPolicy:ac542982-cbeb-4294-891
c-a5a941dfa787:autoScalingGroupName/MyGroup:policyName/my-scaleout-policy
```

Keep these names as they are necessary to create the Cloudwatch alarms.

We can create a second policy to decrease the number of running instances if there is not enough traffic:

```
   $as-put-scaling-policy my-scalein-policy –auto-scaling-group MyGroup --adjustment=-2  --type
ChangeInCapacity
```

As you can see, several scaling policies may exist for a given group, controlling different aspects of the application. Please check the AWS documentation for more information about scaling policies.

Using Cloudwatch to Trigger Autoscaling

In the previous section, we created scaling policies that provide instructions to the Auto Scaling group about how to scale in and scale out when the specified conditions change. In this section we create two alarms, associated with the two scaling policies defined before. These alarms identify the metrics to watch and define the conditions for scaling.

The first alarm increases the size of the group if the average CPU usage is over 80% in a period of 2 minutes:

```
   $mon-put-metric-alarm --alarm-name AddCapacity  --metric-name CPUUtilization --namespace
“AWS/EC2” --statistic Average --period 120 --threshold 80 --comparison-operator
GreaterThanOrEqualToThreshold --dimensions “AutoScalingGroupName=MyGroup”
--evaluation-periods 2 --alarm-actions <ARN_policy_scalein>
```

The second alarm decreases the size of the group when the average CPU usage goes below 40% in a period of 2 minutes:

```
   $mon-put-metric-alarm --alarm-name RemoveCapacity --metric-name CPUUtilization
--namespace “AWS/EC2”  --statistic Average  --period 120  --threshold 40  --comparison-operator
LessThanOrEqualToThreshold  --dimensions “AutoScalingGroupName=MyGroup”
--evaluation-periods 2 –alarm-actions <ARN_policy_scaleout>
```

With both alarms we ensure the creation or removal of instances according to CPU usage, not wasting resources by having underutilized servers while ensuring we have capacity to fulfil the requests we receive.

You can verify your *Cloudwatch alarms* by running the command:

```
   $mon-describe-alarms --headers
```

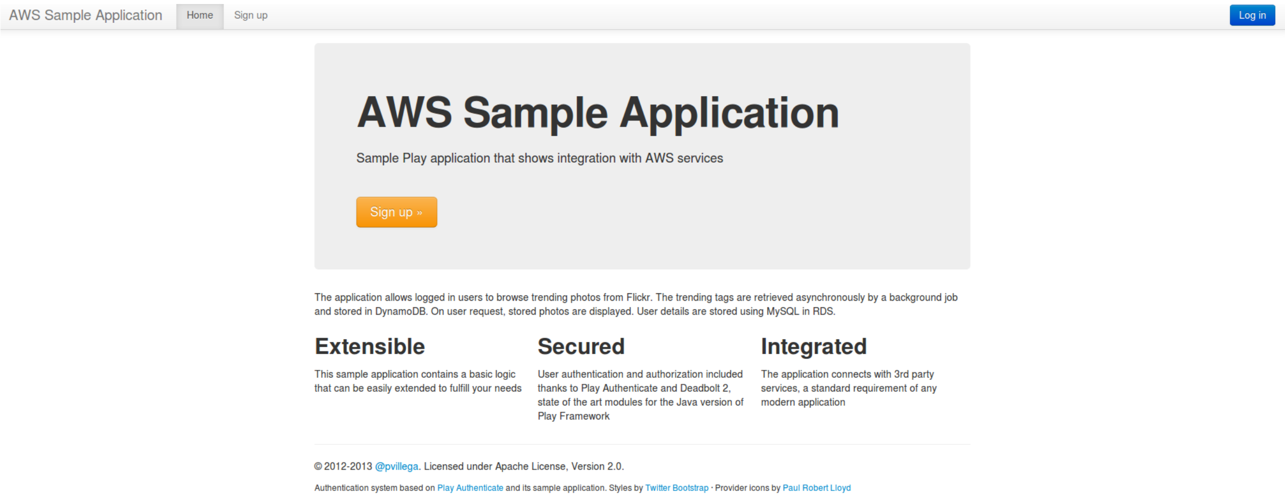
which returns a list of the active alarms and its associated scaling policies.

After running these steps, you have configured an automatic scaling system. This system adds new instances if the traffic increases by a certain threshold and automatically removes instances which are not being fully utilized.

Amazon AWS provides an extensive set of tools to define the autoscaling of your application. This example has shown the necessary steps to configure our sample application, but there are many other options available. Please read the documentation to know more about them.

Testing the deployment

At this stage we have deployed the application in EC2 and enabled scaling based on demand. If we access the application via the custom domain we associated to our Elastic Load Balancer, we see the main page of the application:



AWS Sample Application

Sample Play application that shows integration with AWS services

The application allows logged in users to browse trending photos from Flickr. The trending tags are retrieved asynchronously by a background job and stored in DynamoDB. On user request, stored photos are displayed. User details are stored using MySQL in RDS.

Extensible

This sample application contains a basic logic that can be easily extended to fulfill your needs

Secured

User authentication and authorization included thanks to Play Authenticate and Deadbolt 2, state of the art modules for the Java version of Play Framework

Integrated

The application connects with 3rd party services, a standard requirement of any modern application

© 2012-2013 @pvillega. Licensed under Apache License, Version 2.0.

Authentication system based on Play Authenticate and its sample application. Styles by Twitter Bootstrap · Provider icons by Paul Robert Lloyd

To test the scaling of the application I recommend using Siege. Usage of Siege is outside the scope of this document, but you can find an example in here.

This concludes the second chapter of the article. At this point we have created a Play Framework application that interacts with Amazon AWS and we have deployed this application in Amazon EC2 using Ansible. The next chapter explains how to deploy the same application in Elastic BeanStalk.