Sławomir Jastrzębski, Fotolia

Cloud computing with Amazon's Elastic Compute Cloud

EXPANDABLE CLOUD

Cloud computing systems like Amazon's Elastic Compute Cloud (EC2) save power and overhead by taking the peak out of your server load. **BY DAN FROST**

One would expect Amazon to guard their infrastructure jealously, but piece by piece, Amazon has been opening up their infrastructure so that the rest of us can get our hands dirty playing with file storage, virtual servers, and even physical deliveries on the same kind of ludicrous scale Amazon uses every day.

Amazon Web Services (AWS) makes these systems available over a web services framework so that everything from using more storage space, to creating virtual servers, to requesting physical deliveries happens over SOAP. Instead of filling in forms each time you want more, less, or a different infrastructure, your code can stay as is and AWS provides the necessary services as needed.

Each of the Amazon web services comes with tools developed by Amazon, and a growing number are developed by third parties. Increasingly, third parties are building new and complex services on top of these basic services – for example, hugely scalable databases and web indexing. Amazon's "Elastic Compute Cloud" (EC2) provides virtual servers charged at an hourly rate from US$ 0.10 an hour, running on Amazon's huge number of servers spread across their data centers. EC2 gives you computing in a "cloud."

The term *cloud computing* can mean many different things – from Software as a Service (SaaS) to highly integrable services – but it also means that you don't worry about the infrastructure. EC2 runs on the Xen virtualization layer, but you don't have to worry about this – you just request more virtual servers and they appear. Cloud computing changes the way you provision servers because it makes rapid scaling easier and cheaper at peak demand times. Instead of spending several thousand dollars on five machines that spend 90% of their lives doing nothing, you can use five EC2 instances only when you need them.

One of the biggest differences EC2 presents is the use of Amazon Machine Images (AMIs), which are like server configurations or perhaps bootable CDs. EC2 uses these AMIs when it creates a new virtual server. In this article, I describe how to create your own AMI.

Getting Started

To start with EC2, set up an AWS account [1], then go the the EC2 homepage [2], where you can get the various keys you'll need. The easiest way to control your EC2 instances is with ElasticFox, a plugin for Firefox. Install ElasticFox from the Amazon Web Services site [3] and have a look around. The first step is to set up a virtual machine. In the center of the window, you'll see a list of AMIs. To create a new instance, select the appropriate AMI and click the I/O button.

A good place to start is by selecting *ec2-public-images/fedora-core4-apache-mysql* with the AMI ID *ami-25b6534c*. The new instance will appear in the list at the bottom after a few moments. When it says "running," right-click on it and copy the Public DNS into a browser. Now you should see what looks like a normal website running from your EC2 instance.

A host of AMIs are publicly available for PHP, Rails, Java, specialized number crunching, and other uses. The beauty of using AMIs is that they are honed for a particular purpose, so when your EC2 instance is running, it doesn't need any unnecessary software. This setup is somewhat different from traditional hosting, in which the server tends to contain all the software to run all your apps.

Creating an AMI

Creating an AMI takes a while, which can make things tricky at first, but once you have cracked it, the steps are pretty easy. AMIs can contain anything from a single service to all your applications and databases, so all your EC2 instances will look just like you want them to look. For example, if you deploy lots of websites that are based on the same software, you can push the software into the AMI so that you only have to upload the site itself to the EC2 instance.

Once the working AMI is set up, you can use it to create as many instances as

you like. To create an AMI, create a Linux image containing all the required files and settings, bundle the image, and upload it to EC2, then register the uploaded image.

Creating the Linux Image

The first step is to create the Linux image in a "loopback file," which is used to simulate a hard disk and avoids the need to create the operating system on a separate drive. The *dd* command copies raw data into a file of a specified size – in this case, 1GB:

```
dd if=/dev/zero ➥
of=myimage.fs count=1024 ➥
bs=1M
```

The *dd* program works in units of blocks; *count* is the number of blocks to be copied and *bs* is the size of the blocks to use. Running this produces a completely empty file, inside which you'll create the linux image.

Next, you need to create a filesystem with *mke2fs*, which adds an ext3 filesystem in the file you just created:

```
mke2fs -F -j myimage.fs
```

If you haven't created filesystems in loopback devices before, this step might seem a bit weird. Think of it like this: The file *myimage.fs*_ now contains a filesystem that can be mounted just like an external hard drive:

```
sudo mount -o loop ➥
myimage.fs /mnt
```

This step mounts the filesystem on */mnt*. The *-o loop* options makes you mount the filesystem as a loopback, rather than as a real disk drive. The whole loopback device arrangement should begin to fall into place now. Look around */mnt*. All you'll see so far is the usual ext3 *lost+found* directory. Now you can create files and directories – this filesystem will hold a small version of Linux to use on EC2.

To begin your basic OS, you can use debootstrap, a program that sets up a basic Debian system on a given filesystem. If you need to install this program, it is simply a matter of entering *apt-get install debootstrap*. If you're doing this on another flavor of Linux, the steps are similar, although you might need to set up the OS differently, but for the example here, run debootstrap:

```
sudo debootstrap --arch ➥
i386 edgy mnt
```

While debootstrap runs, you'll see various retrieving and validating messages as it gets the required files and installs them in the loopback filesystem. When the program finishes, have another look at the filesystem in */mnt*. Things should look familiar. With the next two commands, you finish off this task and move inside the new Linux image as though it has been booted all along:

```
sudo cp /etc/apt/sources.list ➥
/mnt/etc/apt/sources.list
sudo chroot /mnt
mount -t proc none proc
```

Now that you're in the image, change the password:

```
passwd
```

What you have now is an empty Debian image, which has pretty limited use. With the use of Aptitude, update the image and install an SSH server:

```
aptitude update
aptitude upgrade
aptitude install openssh-server
```

If you'd like Apache, enter:

```
aptitude install apache2
```

Next, you should address the network settings. Note that you'll edit */etc/network/interfaces* in the image – it isn't your local machine any more. Use an editor to put the following into */etc/network/interfaces*

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
```

and then add the following to */etc/fstab*:

```
/dev/sda2     /mnt     ext3 ➥
defaults 1 2
/dev/sda3     swap     swap ➥
defaults 0 0
```

Now you're done with the image, but you can play around and perhaps install some software (e.g., subversion, MySQL, or anything else you use frequently). When you're finished, enter:

```
exit
sudo umount /mnt
```

Now you have Linux in a file. By this stage, you can imagine how EC2 will use this to create instances. Any programs, code, or files required by your applications can be added to the new Linux image at this stage. Anything you add now will be on every EC2 instance you create – for example, just copy files directly into the mounted image. If you use some standard CMS software, you might grab a copy for the image: *svn co http://svn.server/my_project/trunk mnt/var/www/html/my_project*

Now you can set up databases, symlinks, config files, or anything else as you normally would. When all the files are set up, your Linux image is ready.

Preparing and Uploading

Amazon provides two sets of tools. The first bundle of software you need is the AMI Tools package [4], which contains the tools for creating AMIs and uploading them to Amazon. The second is the EC2 command-line tools bundle [5], which performs more generic tasks, such as creating and controlling EC2 instances. To start, download both files and extract them into a directory. Although you can install these in a system directory (*/usr/local* for example), for this example, install in the home directory. With the files in place, set some en-

Understanding the Keys

EC2 comes with a whole stack of keys, access ids, certificates, and the like:

- Access key id – a 20-character string that identifies all the requests you make to Amazon's Web Services.
- Secret Access Key – a 40-character string that validates the access key.
- X.509 certificate file – a public key and a private key.
- Private key file.

KeyPairs connect you to running instances without the need to send your password in the clear. In the examples here, you generate them through ElasticFox.

vironment variables. The EC2 software requires a couple of custom variables:

```
export EC2_HOME=➥
~/ec2-api-tools/
export EC2_AMITOOL_HOME=➥
~/ec2-ami-tools/
```

For more information on these variables, see the Readme file *ec2-ami-tools/readme-install.txt*.

Now, make sure *JAVA_HOME* is set, and add the EC2 directories to the *PATH* variable.

```
export JAVA_HOME=➥
/usr/lib/jvm/cacao/jre/
export PATH=$PATH:➥
ec2-api-tools/bin:➥
ec2-ami-tools/bin
```

To check that everything is working, enter:

```
ec2-bundle-image --help
```

To use your Linux image, you need to bundle it, upload it to EC2, and register it. To bundle the image, use the ec2-bundle-image tool, which is provided by AMI tools:

```
ec2-bundle-image ➥
  -i myimage.fs ➥
  --cert ec2-keys/cert-XXX.pem ➥
  --privatekey ec2-keys/➥
     pk-XXX.pem ➥
  -u 1234-2345-1234
```

This takes your Linux image, splits it into several files, and creates a manifest file, which tells EC2 where your image is hosted in Amazon Simple Storage Services (S3) and how to use it. The split image files are created in */tmp/* by default – have a look once the ec2-bundle-image process is complete.

Next, upload the image with the ec2-upload-bundle tool, which takes all the files you just created on your local machine and uploads them to S3:

```
ec2-upload-bundle ➥
    -b my-image \
    -m /tmp/myimage.➥
         fs.manifest.xml ➥
    -a access-key-here ➥
    -s secret-key-here ➥
    --ec2cert ~/test1518.pem
```

This might take some time, so make sure your terminal won't timeout while you're waiting (e.g., use *screen*). After the upload has completed, look in your S3 account and notice that the bucket named *my-image* contains the files that you created with ec2-bundle-image.

Your Linux image is now sitting on S3 with a manifest file.

Registering and using AMI

The last step is to register and use the Linux image:

```
ec2-register ➥
    my-ubuntu-df/➥
    myimage.fs.manifest.xml ➥
    -K  ~/.ec2/pk-XXXX.pem ➥
    -C ~/.ec2/cert-XXXX.pem
```

Note that *ec2-register* refers to the manifest file on S3, not on your local machine – hence, the path *my-ubuntu-df/myimage.fs.manifest.xml*. Also, you can register through ElasticFox by clicking the green plus icon in the AMI listing and entering the path to the manifest file.

To use the image, fire up ElasticFox, refresh the list of AMIs, and find your new AMI using the filter box to the top right of the AMI list. Create a new instance of the AMI, and there you have it: You're running your own Linux image on EC2 for US$ 0.10 an hour.

Once the instance is running, *ssh* onto it and play around. Very quickly you'll decide what software and content files you want on all your EC2 instances, and you can then push the files and programs into your AMI using the steps I took you through above.

If you think your image is really good, you can share it for free or charge others for the use of it through Amazon.

Playing in the Clouds

Like any new technology, cloud computing is fun to play with, but you'll like it even better if you can get some really good use out of it.

So, what is EC2 good for?

Cloud computing makes it easier to throw vast amounts of hardware at a problem without having to worry about the details of hosting, networking connectivity, cooling, or the boredom of 100 hosting contracts. This makes EC2 great for anything that requires lots of servers – processing millions of images, searching and cataloging tasks, and so on. Anything that can be done quicker by throwing more computing power at it can use EC2.

And because you can requisition servers on the fly, cloud computing is good for time-sensitive tasks, such as sending hundreds of items of email over lunch or preparing lots of video files while the user waits. Scaling on the fly means you don't have dozens of servers sitting around doing nothing (and costing you money).

The cloud is also suited to any service that might need to scale, but you don't know the number of end users – for example, social networks, intranets, extranets, or online applications. Also, you can use EC2 to test new server configurations, and you can use the cloud to test applications [6].

Cloud computing is set to change the way applications are built and deployed. Anything that is impossible now because you can't afford the servers becomes wonderfully possible – or at least much cheaper. Creating custom AMIs will allow you to get the most out of the service by launching EC2 instances fine-tuned for your particular applications. Building and uploading images can take time, but once you have them, it is easy to tweak the images to contain exactly what your applications need and no more.

And once you can create 1,000 copies of your application, you can stop worrying about those server loads. ■

INFO

[1] Creating an AWS account: *https://aws-portal.amazon.com/gp/aws/developer/registration/index.html*

[2] EC2 homepage: *http://www.amazon.com/ec2/*

[3] Amazon web services: *http://developer.amazonwebservices.com/connect/entry.jspa?entryID=609*

[4] EC2 AMI tools: *http://developer.amazonwebservices.com/connect/entry.jspa?externalID=368&categoryID=88*

[5] Amazon command-line tools: *http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351&categoryID=88*

[6] Selenium: *http://selenium-grid.openqa.org/*