

# Using Amazon Web Service

Kansas City Java Users Group

October 8, 2008

Steve Mitchell and Matt Wilson

Byteworks, Inc

[www.ByteworksInc.com](http://www.ByteworksInc.com)

# Using Amazon Web Services

---

What we will discuss:

## Part 1 - Steve Mitchell

- Overview of Amazon Web Services
- Getting Started with EC2
- Getting Started with S3
- Getting Started with SQS

## Part 2 - Matt Wilson

- Planning for Resiliency
- Planning for Scalability

# Using Amazon Web Services

---

## Audience

You should be familiar with the following:

- Programming in Java
- Hosting Java applications on Linux
- Using Web Services

No experience with Amazon Web Services required

# Introductions

---

**Steven Mitchell** is President of Byteworks, Inc and has worked in IT for 26 years--specializing in Java since 1999. Steve lead the KC Java Users Group in 2001.

**Matthew Wilson** is Professional Services Consultant for Byteworks. Matt develops systems software and web applications in several programming languages and has extensive experience in Windows and \*n\*x system administration.

# How did we get involved in AWS?

---

A start-up client asked us to use Amazon WS:

- Needed Amazon Affiliate integration (AS2).
- Wanted scalability without huge upfront costs.
- Did not want to operate a data center.
- Liked the benefits of SQS (Simple Queue Service).

---

# Amazon Web Services

Overview of Service Offerings

# Amazon Web Services

## Infrastructure Services

- Amazon EC2 (Elastic Compute Cloud)
- Amazon S3 (Simple Storage Solution)
- Amazon SQS (Simple Queue Services)
- Amazon SimpleDB (Simple Database)
- Amazon EBS (Elastic Block Store)

## Billing and Payments

- Amazon FPS (Flexible Payment Service)
- Amazon DevPay (To monetize Amazon WS Apps)

## More...

- See <http://aws.amazon.com/>

# Amazon EC2 Overview

---

## **Virtual services in the cloud:**

- Elastic - Change capacity in minutes.
- Flexible - Choice of processor/memory size.
- Works with other Amazon Web Services.
- Features for failure resilient apps (next slide)

# Amazon EC2 Overview

---

## **Elastic Compute Cloud**

- Xen Hypervisor virtual services (Windows Server and SQL Server are in the works).
- Launched from AMI (Amazon Machine Image).
- Instance storage is ephemeral, meaning it is like a ram disk that goes away when the server instance is terminated.

# Amazon EC2 Resilient Features

---

Several features are available to help you make cloud applications more resilient:

- **Elastic IP Addresses:**  
Quickly move IP from one instance to another.
- **Multiple Availability Zones:**  
Deploy to geographically dispersed zones.
- **EBS (Elastic Block Store):**  
Off-instance volumes to persist data. In limited beta.

# Amazon EC2 Pricing

---

Several instances sizes are available:

## **Standard Instance:**

- Small: 1.7 GB, 1 32-bit core, 160 GB. \$0.10/hr
- Medium: 7.5 GB, 2 32-bit cores, 350 GB \$0.20/hr
- Large: 15 GB, 8 64-bit cores, 1690 GB \$0.80/hr

## **Data Transfer:**

- \$0.100/GB transferred in.
- \$0.170/GB first 10TB/mo transferred out.

See site for more details and options.

# Amazon S3 Overview

---

## Simple Storage System

### Internet-based storage:

- Read, write, and delete objects 1 byte to 5GB.
- Store in bucket using developer-defined key.
- Authentication mechanisms (bucket and object)
- REST or SOAP Web Services.
- HTTP or BitTorrent protocol.

# Amazon S3 Pricing

---

Pricing model similar to EC2

## **Storage:**

- \$0.15/GB for storage used

## **Data Transfer:**

- \$0.100/GB for transfer in.
- \$0.170/GB for first 10TB/mo. transferred out.
- Transfers to/from EC2 instances are "free"

## **Requests:**

- \$0.01 per 1000 PUT, POST, or LIST requests.
- \$0.01 per 10,000 GET and all other requests.

# Amazon SQS Overview

---

## Simple Queue Service

Exposes Amazon's web-scale Infrastructure:

- Create Unlimited Number of SQS queues.
- Message body can contain up to 8 KB of text.
- Message is locked, but not deleted while being processed (more on this under fault tolerance).
- Message can remain on queue for up to 4 days.
- Queues can be accessed by SOAP or Query interfaces.

# Amazon SQS Overview

---

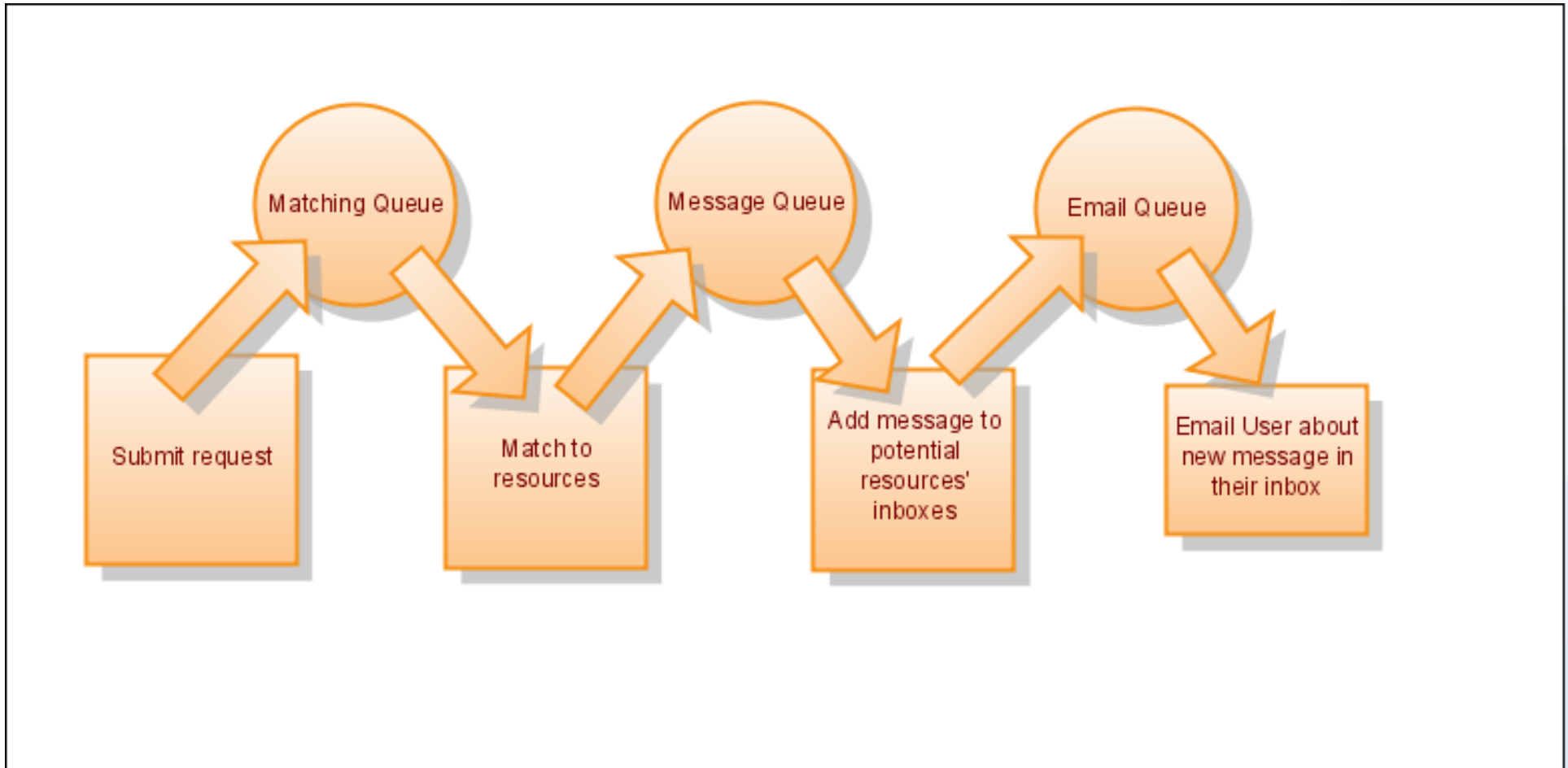
SQS is not JMS

Some unique features:

- Receiving a message does not delete it from the queue, but just makes it invisible/locked.
- Receivers must explicitly delete the message after successful completion.
- Messages become visible again after timeout period (default is 30 seconds)
- Users can query for approximate number of messages on the queue.

# Amazon SQS Overview

Architecture - Fault tolerant, self-healing work flow.



# Amazon SQS Pricing

---

## Simple Queue Service

### Requests

- \$0.01 per 10,000 SQS requests.

### Data Transfer

- \$0.100 per GB transferred in.
- \$0.170/GB for first 10 TB/mo. transferred out.

# Amazon SimpleDB Overview

This service works in conjunctions with S3 and EC2 providing the ability to store, process and query data sets in the cloud.

# Amazon SimpleDB Overview

---

## Features:

- Simple, schemaless structured query image.
- Items stored in "bags" of key/value pairs.
- Developers choose unique key at create time
- Keys and values always stored as Strings.
- Supports PUT, GET, DELETE, and QUERY.
- Items are partitioned in domains.
- Keys must be unique within a domain.
- Automatically indexes your data.

# Amazon SimpleDB Overview

---

## The Data Model:

- Domains, Items, Attributes and Values.
- Analogous to concepts in a traditional spreadsheet table.

itemID	description	color	Material
123	sweater	Blue, red	
456	dress shirt	white, blue	
789	shoes	black	Leather

PUT (item, 123), (description, sweater), (color, blue), (color, red)

PUT (item, 456), (description, dress shirt), (color, white), (color, blue)

PUT (item, 789), (description, shoes), (color, black), (material, leather)

# Amazon SimpleDB Overview

---

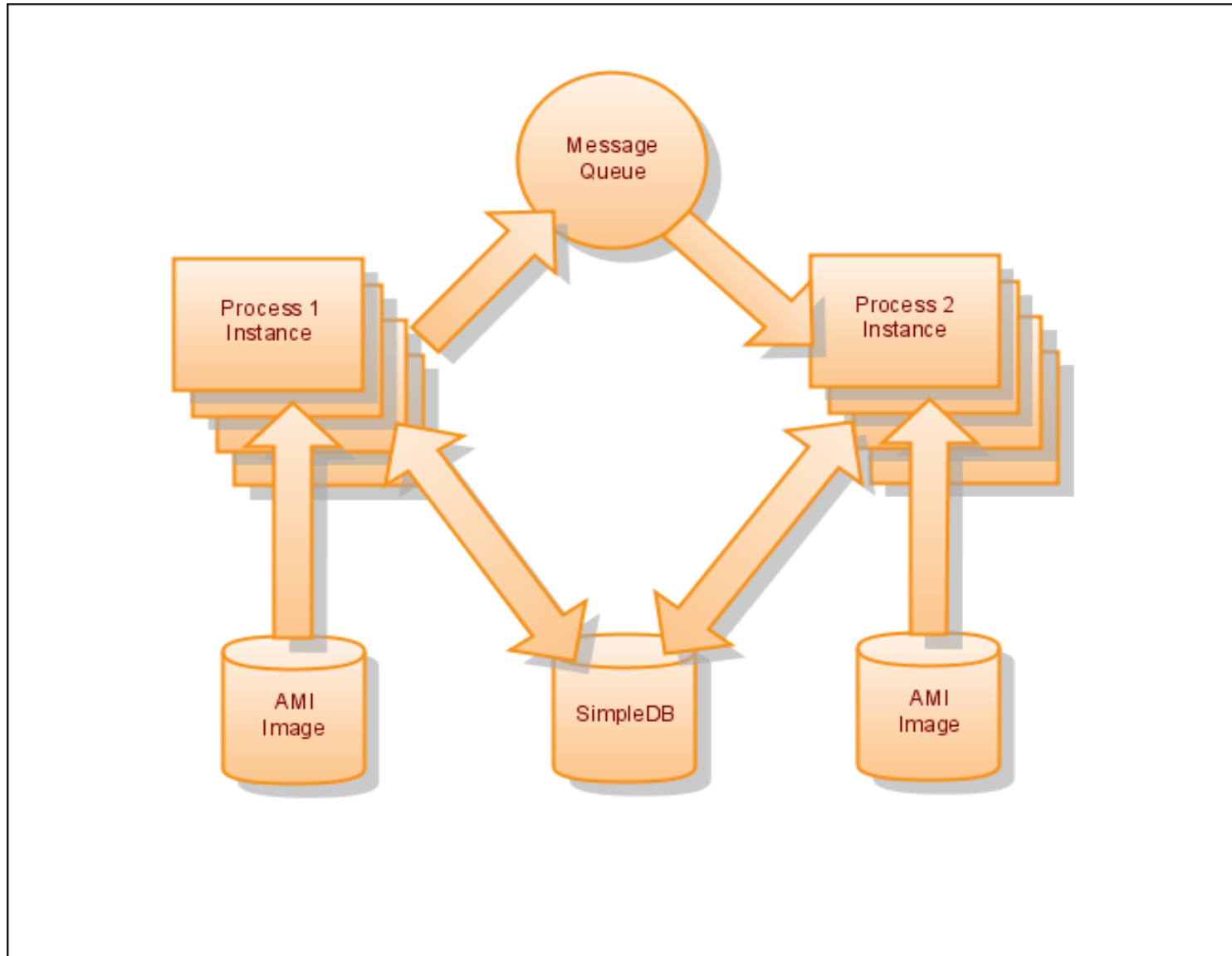
Does not replace relational database.

## **Example Uses:**

- Reduce SQS message size by passing a references to detail data in SimpleDB.
- Provides shared storage between SQS processors to provided message status throughout lifecycle.

# Amazon SimpleDB Overview

## Using SimpleDB as Shared Storage



---

# Getting Started with EC2

Creating your first instance.

# Getting Started with EC2

---

## Use the Resources Available

- There is an excellent Getting Started Guide to walk you through set-up.
- This presentation does not attempt to recreate that.

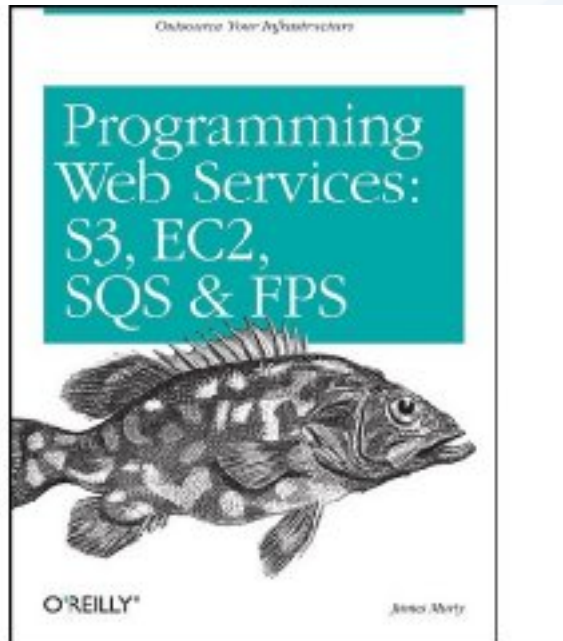
<http://docs.amazonwebservices.com/AWSEC2/2008-02-01/GettingStartedGuide/>

# Getting Started with EC2

## Learning about EC2

There is a lot of information available online:

- <http://aws.amazon.com/ec2/>
- <http://developer.amazonwebservices.com>



# Getting Started with EC2

---

## Understanding the AMI (Amazon Machine Image)

- Encrypted file stored on Amazon S3.
- Contains all the information necessary to boot your software.
- Can be saved as a custom AMI bundle.

# Getting Started with EC2

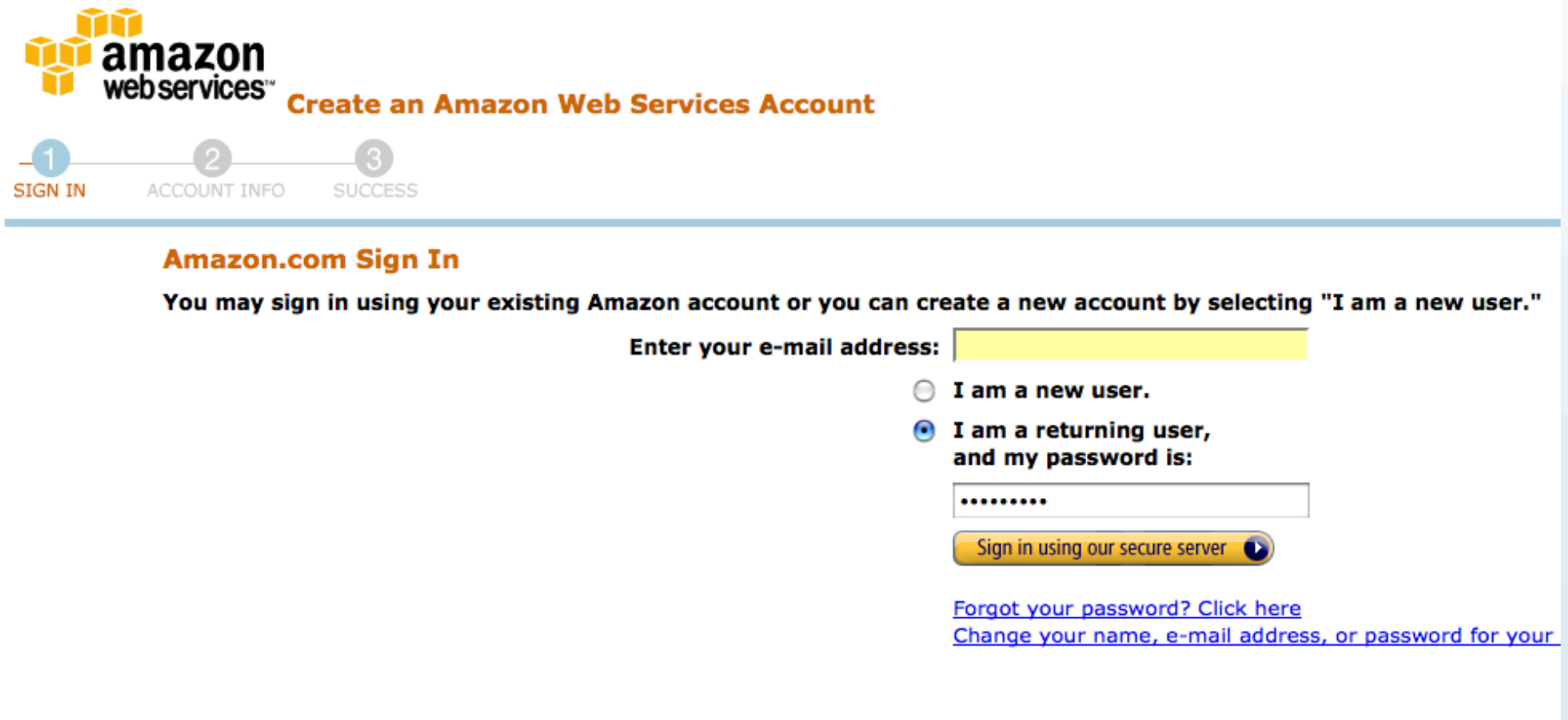
---

## Picking an AMI Image

- Via Web services: `ec2dim -o amazon | grep mysql`
- Amazon: <http://developer.amazonwebservices.com>
- Sun: <http://www.sun.com/third-party/global/amazon/>
- Red Hat
- Eric Hammond (Debian/Ubuntu)- <http://alestic.com/>

# Getting Started with EC2

Create a new Amazon account or use an existing one at <http://aws.amazon.com/>:



The screenshot shows the 'Create an Amazon Web Services Account' page. At the top left is the Amazon Web Services logo. Below it is a progress bar with three steps: 1. SIGN IN (highlighted), 2. ACCOUNT INFO, and 3. SUCCESS. The main heading is 'Amazon.com Sign In'. Below this is a paragraph: 'You may sign in using your existing Amazon account or you can create a new account by selecting "I am a new user."' followed by a form. The form includes a label 'Enter your e-mail address:' with a yellow input field. Below that are two radio button options: 'I am a new user.' (unselected) and 'I am a returning user, and my password is:' (selected). The selected option has a password input field with masked characters. At the bottom of the form is a yellow button labeled 'Sign in using our secure server' with a right-pointing arrow. Below the button are two blue links: 'Forgot your password? Click here' and 'Change your name, e-mail address, or password for your account'.

**amazon web services™** Create an Amazon Web Services Account

1 SIGN IN 2 ACCOUNT INFO 3 SUCCESS

## Amazon.com Sign In

You may sign in using your existing Amazon account or you can create a new account by selecting "I am a new user."

Enter your e-mail address:

I am a new user.

I am a returning user, and my password is:

[Forgot your password? Click here](#)  
[Change your name, e-mail address, or password for your account](#)

# Getting Started with EC2

Create a new Amazon account or use an existing one at <http://aws.amazon.com/>:



## **Thank You**

**You Have Created an Amazon Web Services Account**

We have e-mailed you a confirmation and information about your new account.

# Getting Started with EC2

## Sign-in and view Account Identifiers



Contact Us | Create an AWS Account

About AWS | Products | Solutions | Resources | Support | Your Account

Account Activity

Usage Reports

Access Identifiers

Personal Information

Payment Method

DevPay

Home > Your Account > Access Identifiers

Welcome, Steve Mitchell | Sign Out

Account Number

### Access Identifiers

You use your Access Identifiers to identify yourself as the sender of a request to an AWS web service. Access identifiers are also used to authenticate requests to AWS. For services that require authenticated requests, where you need to prove that you are authorized to make the request, you must sign the request by including a "signature" in the request. A request signature is calculated using the pair of public / private Access Identifiers.

#### AWS supports two types of Request Identifiers:

[AWS Access Key Identifiers](#)

[X.509 Certificates](#)

If you're not sure which type of identifier you should use, click [here](#) to see which identifiers can be used with which AWS Services.

#### Access Key ID and Secret Access Key

##### Access Key ID

Use your Access Key ID as the value of the `AWSAccessKeyId` parameter in requests you send to Amazon Web Services (when required). Your Access Key ID identifies you as the party responsible for the request.

Your Access Key ID:

##### Secret Access Key

Since your Access Key ID is not encrypted in requests to AWS, it could be discovered and used by anyone. Services that are not free require you to provide additional information, a request signature, to verify that a request containing your unique Access Key ID could only have come from you.

Your Secret Access Key:

[+ Show](#)

[Generate](#)

Generate a new Secret Access Key  
(You will be asked to confirm this selection before a new Secret Access Key will be generated.)

You use your Secret Access Key to calculate a signature to include in requests to web services that require authenticated requests. To learn more about request signatures, including when to use them and how you calculate them, please refer to the technical documentation for the specific web service(s) you are using.

# Account #

# Access Key Secret Key

# 509 Cert

# Getting Started with EC2

## Types of Account Identifiers

### **Account Number:**

- Identifies your Amazon account.
- Required to create new AMI images.
- Used to identify users to grant permissions.

### **Access Key Identifiers:**

- Used to authenticate with Web Services.

### **X.509 Certificate (pk\*.pem & cert\*.pem):**

- Keypair used by the Java ec2-api-tools

### **RSA keypair (~/.ssh/id\_rsa-name-keypair):**

- Keypair used by ssh, PuTTY, scp, sftp

# Getting Started with EC2

---

## Account Number:

- `ec2-modify-image-attribute ami-12345 -l -a 123456789`

## Access Key/Secret Access Key:

- `new AWSCredentials(accessKey, secretKey);`

## X.509 Certificate:

- `ec2-bundle-vol -d /mnt -k /mnt/pk.pem -c /mnt/cert.pem -u 495219933132 -r i386 -p sampleimage`

## RSA keypair:

- `ec2-run-instance ami-26bc584f -k gsg-keypair`

# Getting Started with EC2

## Setting up your environment

Just follow the getting started guide:

- <http://docs.amazonwebservices.com/AWSEC2/2008-05-05/GettingStartedGuide/>

1. Download Amazon EC2 tools.
2. Generate your 509 and RSA keys.
3. Configure environment variables.

```
EC2_CERT=/Users/stevecmitchell/.ec2/cert-SIOCYXMN3UGMXFQSABV2DE55BQO26LJ2.pem  
EC2_HOME=/usr/local/ec2  
EC2_PRIVATE_KEY=/Users/stevecmitchell/.ec2/pk-SIOCYXMN3UGMXFQSABV2DE55BQO26LJ2.pem  
EC2_URL=https://ec2.amazonaws.com
```

# Getting Started with EC2

## Exploring the EC2 tools - Starting an Instance

1. Start the instance.
2. Check if a URL has been assigned (takes a minute or two).
3. Connect to the instance using the URL.

```
ec2-run-instances ami-26bc584f -k gsg-keypair
```

```
ec2-describe-instances
```

```
ssh -i id_rsa-keypair root@ec2-67-202-53-123.  
compute-1.amazonaws.com
```

# Getting Started with EC2

## Elastic IP addresses

1. Allocate an IP address.
2. Assign it to an instance.
3. Verify assignment.

```
ec2-allocate-address
```

```
ADDRESS      77.102.143.105
```

```
ec2-describe-addresses
```

```
ADDRESS      77.102.143.105
```

```
ec2-associate-address -i i-f12ef198 77.102.143.105
```

```
ADDRESS      77.102.143.105      i-f12ef198
```

# Getting Started with EC2

## Creating a custom AMI

- See online reference:
- <http://docs.amazonwebservices.com/AWSEC2/2008-02-01/GettingStartedGuide/index.html?creating-an-image.html>

Granting others authorization to launch your image.

```
ec2-modify-image-attribute ami-139f7b7a -l -a  
210987654321
```

```
ec2-describe-image-attribute ami-139f7b7a -l
```

```
launchPermission ami-139f7b7a userId 210987654321
```

```
launchPermission ami-139f7b7a userId 123456789012
```

# Getting Started with EC2

---

Demo

A decorative graphic consisting of several concentric circles in shades of light blue, positioned in the bottom right corner of the slide.

---

# Using S3

Using the JetS3t API

# Using the JetS3t API

---

## Connecting to S3

```
/**
 * Returns the RestS3Service.
 *
 * @return RestS3Service
 * @throws S3ServiceException e
 */
private RestS3Service getRestS3Service() throws S3ServiceException {
    AWSCredentials credentials =
        new AWSCredentials(getAmazonAccessKey(), getAmazonSecretKey());
    return new RestS3Service(credentials);
}
```

# Using the JetS3t API

## Getting a Bucket

```
/**
 * Returns the image bucket, creating one if necessary.
 * @return S3Bucket
 * @throws S3ServiceException e
 */
private S3Bucket getS3Bucket() throws S3ServiceException {
    if (s3Bucket == null) {
        S3Bucket[] buckets = getRestS3Service().listAllBuckets();
        for (S3Bucket bucket : buckets) {
            if (bucket.getName().equals(amazonBucketName)) {
                s3Bucket = bucket;
                break;
            }
        }
    }
    if (s3Bucket == null) {
        s3Bucket = createS3Bucket();
    }
    return s3Bucket;
}
```

# Using the JetS3t API

## Creating a Bucket

```
/**
 * Creates the image bucket.
 * @return S3Bucket
 * @throws S3ServiceException e
 */
private S3Bucket createS3Bucket() throws S3ServiceException {
    S3Bucket rval =
        getRestS3Service().createBucket(new S3Bucket(amazonBucketName));
    AccessControlList acl = new AccessControlList();
    acl.grantPermission(GroupGrantee.ALL_USERS, Permission.PERMISSION_READ);
    rval.setAcl(acl);
    return rval;
}
```

# Using the JetS3t API

## Storing an Object

```
S3Bucket bucket = getS3Bucket();
AccessControlList acl = getRestS3Service().getBucketAcl(bucket);
if (photo.getPhotoId() != null) {
    s3object = getRestS3Service().getObject(getS3Bucket(), photo.getPhotoId());
}
if (s3object == null) {
    S3object toStore = new S3object(bucket, imageFile);
    toStore.setAcl(acl); // use the bucket's acl.
    Map<String, Object> metadata = new HashMap<String, Object>();
    metadata.put("PhotoType", photo.getPhotoType());
    metadata.put("FileName", photo.getFileName());
    toStore.addAllMetadata(metadata);
    String key = generateKey(photo.getFileName());
    toStore.setKey(key);
    photo.setPhotoId(key);
    getRestS3Service().putObject(bucket, toStore);
} else {
    Map<String, Object> metadata = s3object.getMetadataMap();
    metadata.put("PhotoType", photo.getPhotoType());
    metadata.put("FileName", photo.getFileName());
    s3object.replaceAllMetadata(metadata);
    getRestS3Service().putObject(bucket, s3object);
}
```

# Using the JetS3t API

## Retrieving an Object

```
/**
 * The method retrieves the photo content as an InputStream.
 *
 * @param photoId - The ID of the photo to retrieve.
 * @return InputStream
 * @throws Exception e
 */
public InputStream getInputStream(final String photoId) throws Exception {
    S3Object s3Object = getRestS3Service().getObject(getS3Bucket(), photoId);
    if (s3Object != null) {
        return s3Object.getDataInputStream();
    }
    return null;
}
```

# Using the JetS3t API

---

Demo

A decorative graphic consisting of several concentric circles in shades of light blue, positioned in the bottom right corner of the slide.

---

# Using SQS

Using the Typica API

# Using SQS with Typica API

## Connecting to SQS Queue

```
/**
 * Looks for a queue by name: if found, return a MessageQueue object for it.
 * Else, return null.
 * @param queueName - Name of the queue to return.
 * @return MessageQueue
 * @throws Exception e
 */
private MessageQueue getMessageQueue(final String queueName) throws Exception {
    return SQSUtils.connectToQueue(queueName, getAmazonAccessKey(), getAmazonSecretKey());
}
```

# Using SQS with Typica API

## Managing Queues

```
/**
 * Creates a new message queue. The queue name must be unique within the scope of the
 * queues you own. Optionally, you can supply a queue that might be one that belongs to
 * another user that has granted you access to the queue. In that case, supply the fully
 * qualified queue name (i.e. "/A98KKI3K0RJ7Q/grantedQueue").
 * @param queueName - The name of the queue to get or create.
 * @throws Exception e
 */
public void createQueue(final String queueName) throws Exception {
    QueueService queueService = new QueueService(getAmazonAccessKey(), getAmazonSecretKey(), true);
    queueService.getOrCreateMessageQueue(queueName);
}

/**
 * Deletes the specified queue.
 * You cannot delete a queue for at least 60 seconds after it is created.
 * @param queueName - Name of queue to be deleted.
 * @throws Exception e
 */
public void deleteQueue(final String queueName) throws Exception {
    MessageQueue messageQueue = getMessageQueue(queueName);
    if (messageQueue != null) {
        messageQueue.deleteQueue();
    }
}
```

# Using SQS with Typica API

## Sending and Receiving Messages

```
/**
 * Sends a message to a specified queue. The message must be between 1 and 256K bytes long.
 * @param textMessage - The message to send.
 * @throws Exception e
 */
public void sendMessage(final TextMessage textMessage) throws Exception {
    MessageQueue msgQueue = getMessageQueue(textMessage.getDestination());
    if (msgQueue != null) {
        String messageId = msgQueue.sendMessage(textMessage.getText());
        textMessage.setMessageID(messageID);
    }
}

/**
 * Attempts to receive a message from the queue. The queue default visibility timeout is used.
 * @param queueName - the name of the queue to query.
 * @return TextMessage
 * @throws Exception e
 */
public TextMessage receiveMessage(final String queueName) throws Exception {
    TextMessage textMessage = null;
    MessageQueue msgQueue = getMessageQueue(queueName);
    if (msgQueue != null) {
        textMessage = transform(queueName, msgQueue.receiveMessage());
    }
    return textMessage;
}
```

# Using SQS with Typica API

## How Byteworks choose to decouple app from Typica

```
/**
 * Transforms a Typica Message into a TextMessage model object.
 * @param queueName - The queue from where the message was fetched.
 * @param message - The Typica Message to transform.
 * @return TextMessage
 */
private TextMessage transform(final String queueName, final Message message) {
    TextMessage textMessage = null;
    if (message != null) {
        textMessage = new TextMessage();
        textMessage.setDestination(queueName);
        textMessage.setMessageID(message.getMessageId());
        textMessage.setText(message.getMessageBody());
        textMessage.setReceiptHandle(message.getReceiptHandle()); // Required for delete
    }
    return textMessage;
}
```

# Using SQS with Typica API

## Deleting Messages after successful processing

```
/**
 * Deletes the message identified by message object on the queue this object represents.
 * @param textMessage - Text Message to delete.
 * @throws Exception e
 */
public void deleteMessage(final TextMessage textMessage) throws Exception {
    MessageQueue msgQueue = getMessageQueue(textMessage.getDestination());
    if (msgQueue != null) {
        msgQueue.deleteMessage(textMessage.getReceiptHandle());
    }
    textMessage.setMessageID(null);
}
```

# Using SQS with Typica API

---

Demo

A decorative graphic consisting of several concentric circles in shades of light blue, positioned in the bottom right corner of the slide.

---

# Break

10 minutes



---

# Planning for Resiliency

## Infrastructure Considerations

# Planning for Resiliency

---

## Elastic Block Storage

- Persistent storage for EC2
- Independent of particular instances  
(and instance sizes, enabling instance upgrades)
- Mountable block device
- Faster IO than ephemeral storage and local disk

from **<http://tinyurl.com/esh-on-ebs>**  
**(Eric Hammond's EBS Tutorial)**

# Planning for Resiliency

---

## Elastic Block Storage

- Up to 20 devices per AWS account
- Up to 1TB per volume
- \$0.10 per GB-month of provisioned storage
- \$0.10 per 1 million I/O requests

# Planning for Resiliency

---

## Elastic IP Addresses

- replace the initial public IP on the instance
- One instance per IP address
- up to 5 addresses per AWS account (by default)
- free while allocated (in use)
- \$0.01 per hour while unallocated
- \$0.10 per remap ( > 100/month)

# Planning for Resiliency

---

## Elastic IP Addresses

- DNS (and reverse DNS) name
- Inside the cloud, the DNS names resolve to the internal (private, non-routable) IP addresses
- Remaps can take up to several minutes, so this (used solely) is a resiliency strategy of last resort

# Planning for Resiliency

## Availability Zones and the 1 Region

- optionally select during image instantiation
- \$0.01 per GB (each way) to transfer between zones
- named differently per AWS account (us-east-1b is not necessarily the same zone as us-east-1b on another account)

# Planning for Resiliency

## Availability Zones and the 1 Region

- availability zones on entirely distinct supporting infrastructure, but not geographically isolated
- multiple regions would provide geographic isolation of instances. But, EC2 has only 1 region (us-east).

# Planning for Resiliency

## Recent AWS Failures

- 2008-09-14 - catastrophic failure; partial data loss
- 2008-09-10 - SQS partly down for 1 hour
- 2008-07-20 - S3 down several hours
- 2008-07 - blocked by spam source lists
- 2008-04-07 - EC2 down 1 hour
- 2008-02-15 - AWS (S3 mostly) down 2 hours

See <http://status.aws.amazon.com/>

---

# Scaling with EC2

Infrastructure Considerations

# Infrastructure Considerations

## Load Distribution Strategies - DNS

- Round Robin DNS
  - different public IP addresses provide one service
  - problem of stale caches (and chains of caches)
  - useful for scaling only if you have direct programmatic control over your DNS server
- Geographic DNS
  - clients get the IP address closest (or otherwise best/most available) to them
  - not useful for EC2

# Infrastructure Considerations

## Load Distribution Strategies - Switching

### "IP Sprayer" Options

- use an expensive appliance
- use an EC2 instance as your switch
  - Linux Virtual Server
    - <http://www.linuxvirtualserver.org/>
    - Kernel patches - tcp load balancer
  - stunnel/HAProxy Reverse Proxy
    - SSL termination, failover, load balancing
    - hot reconfiguration allows cloud expansion without *\*any\** service interruption

# Infrastructure Considerations

## stunnel/lighttpd/HAProxy Reverse Proxy

- SSL Termination (stunnel)
  - ease the encryption/decryption load on the application servers
  - add `X-Forwarded-SSL-Encrypted: True` so your application knows whether to make the other URLs `https://`
- Static Content Caching (lighttpd)
- Load Balancing (HAProxy)
  - add/remove back-ends dynamically (at runtime)
  - add `X-Forwarded-For: 12.13.14.15` so your application can log and secure things properly\
  - "Sticky Sessions", if you need them, but don't

# Infrastructure Considerations

## Resilient and Scalable EC2 Architecture

- SSL/Caching/Balancing "Gateway" Nodes:
  - 1 (or more) in each Availability Zone
  - several medium or large instances should provide enough scaling for tens of thousands of hits per second (billions of hits per day)
  - Provide enough of these "Gateway" nodes to support the maximum you could ever scale to in the time it would take for you to bring up another one
- Application Server Nodes: tomcat6, for instance
- Database Server Nodes: mysql5, for instance

# Infrastructure Considerations

---

## Application Server Nodes

- Ubuntu Intrepid Ibex (alestic x86 ami)
- custom startup script to prepare the environment
- optionally, save the prepared image as our own
- tomcat6 (currently 6.0.18-0ubuntu1)
- sun-java6-jdk/sun-java6-bin (6-07-4ubuntu2)

# Infrastructure Considerations

## Database Server Nodes

- MySQL Cluster 5.0 (NDB)
  - In-memory distributed/replicated RDBMS
- MySQL Master/Slave Replication
  - MySQLProxy (do not use DNS) for load distribution
    - Sticky sessions
    - Can configure to send "read" transactions to slaves and "write" transactions to master (so that all writes go through 1 node)
    - Can be configured to failover the master, or use failover with EBS to store the data

---

# EC2 Third-Party Vendors

Commercial and Open Source Scalability

# EC2 Third-Party Vendors

---

## RightScale

- monitoring, auto-scaling, backups, access control
- Website Edition
  - MySQL Master/Slave, Load Balancers, Application Servers
- Grid Edition
  - Batch Processing, SQS

## Enomaly - [enomalism.com](http://enomalism.com)

- Elastic Computing Platform
- Geographic load balancing

## WeoGeo - WeoGeo

# EC2 Third-Party Vendors

---

Morph Labs - <http://mor.ph/>

- Ruby on Rails hosting

Atlantic Dominion Solutions

- Rails monitoring

MySQL

- Official EC2 Support

SnapLogic

- EC2 Storage Image

Oracle

- Oracle Enterprise Linux, Unbreakable Support

---

# Creating Images Smartly

Leveraging Startup Scripts

# Leveraging Startup Scripts

---

- Restore configuration (to access EBS) from S3
- Install and upgrade standard packages
- Install custom packages and applications

Demo

# Contact Info

---

We welcome your comments and questions:

Steve Mitchell

(913) 825-1285

[SMitchell@ByteworksInc.com](mailto:SMitchell@ByteworksInc.com)

Matt Wilson

(913) 952-2173

[MWilson@ByteworksInc.com](mailto:MWilson@ByteworksInc.com)

[www.ByteworksInc.com](http://www.ByteworksInc.com)