CLOUD ARCHITECTURE

[And the case of AMAZON AWS]

By : Arunbalaji Sivakumar, SUID: 594227528

Graduate Student, Computer Engineering, Syracuse University

Date: December 09, 2014

Instructor: Yuzhe Tang, PHD

Subject CSE 661 - Advanced Computer Architecture

Table of Contents

[**1.** **Abstract** 2](#_Toc406183194)

[**1.1** **Introduction:** 2](#_Toc406183195)

[**2.** **Cloud computing architecture** 3](#_Toc406183196)

[**2.1** **Software as a service (SaaS)** 3](#_Toc406183197)

[**3.** **Compute and Networking** 4](#_Toc406183198)

[**3.1** **Amazon Elastic Compute Cloud (EC2):** 4](#_Toc406183199)

[**3.2** **Amazon virtual private cloud (VPC):** 5](#_Toc406183200)

[**3.3** **Amazon EC2 VPC:** 5](#_Toc406183201)

[**3.4** **AWS EC2 reboot: (Web-source)** 5](#_Toc406183202)

[**4.** **Storage** 6](#_Toc406183203)

[**4.1** **Amazon Simple Storage Service (S3):** 6](#_Toc406183204)

[**4.2** **Amazon Glacier:** 6](#_Toc406183205)

[**5.** **Analytics** 7](#_Toc406183206)

[**5.1** **Amazon Elastic Map Reduce (EMR):** 7](#_Toc406183207)

[**5.2** **Amazon Kinesis:** 8](#_Toc406183208)

[**6.** **A detailed view on Amazon’s Simple Storage Service (S3)** 8](#_Toc406183209)

[**6.1** **Starting Amazon S3:** 8](#_Toc406183210)

[**6.2** **Authenticating requests in the Amazon S3:** 10](#_Toc406183211)

[**6.3** **Signing and authentication:** 10](#_Toc406183212)

[6.3.1 Signing: 10](#_Toc406183213)

[6.3.2 Authentication: 11](#_Toc406183214)

[**6.4** **Authentication Methods:** 12](#_Toc406183215)

[6.4.1 HTTP Authorization Header: 12](#_Toc406183216)

[6.4.2 Query string parameters: 13](#_Toc406183217)

[6.4.3 Example: 17](#_Toc406183218)

[**7.** **Reference** 19](#_Toc406183219)

# **Abstract**

In this paper, the concept of cloud architecture and its implementation in the Amazon Web service is illustrated.

## **Introduction:**

Cloud architecture refers to the way of separating an application from hardware and the operating system. So when a hardware or operating system crashes the application can migrate to a different hardware and operating system without losing any valuable data. The architecture of cloud computing is of multi-tenancy in which a single application can be used for different software or business purposes by customizing it according to the requirement. The application providers need not spend huge amount of money on resources which they will be using only for a certain period of time. Cloud in cloud computing refers to a collection of recourses such as storage, database, security, collection of servers (in case of web hosting companies). Cloud computing means that all these resources available in the cloud and one can acquire these resources according to their requirement and release them when their requirement is fulfilled. The cloud architecture is divided into two parts, the front end and the back end and they are connected through a network, mostly internet. The front end mostly contains the application and back end contains all the components that make up the cloud. The front end need not worry about how the storage, the security, service, networking or management. The front end receives the required information and puts it on its application.

Amazon, Inc. is one of the largest internet companies and a major provider of cloud computing services. In 2006 Amazon launched its web service as Amazon Web Service (AWS) to provide an online service to its clients their applications. AWS provides cloud platforms such as database, Cross-Service, Analytics, storage, Content Delivery Network, networking, App services, deployment and management. The AWS is more flexible, secure, cost-effective, scalable and elastic.

# **Cloud computing architecture**

**Components:**

* front end platform : fat client, thin client, mobile service
* Back end platform : servers, storage
* Cloud based delivery
* Network.

## **Software as a service (SaaS)**

The [software-as-a-service](http://en.wikipedia.org/wiki/Software-as-a-service) is a service where the vendor of cloud handles the installations and maintenances of software in the cloud and users can run that software from the cloud over the intranet or internet. There are no installations required in the client’s machine for any application-specific software - cloud applications run on the server (in the cloud). In earlier days, customers would have to purchase and load their own copy of the software or application on their own servers and rum it and after their job is done they have to continue to maintain that software even though they are not using it. SaaS involves a monthly or annual fee and users need only to pay when they use. Eg: drop box.

**Down side:**

A major drawback of using SaaS is that the users must put a blind trust into the online software vendors as all their productivity and documentation are now in the vendor's hands. Thus, protection of the file and privacy becomes even more necessary, as the internet has now become a major part the business development.

**Instances:**single instance: One copy is made and multiple users can share it.

Eg: an email system might contain 1000 instances of the same one kilobyte of file attachment. With single instance only one instance of the attachment wil be stored; and every other instance is referenced back to the one master copy.

Multi instance: Separate copy of instance and multiple users run it parallel.

Eg: skype (not cloud application)

Multi-tenancy: Customers will share the same cloud service, platform and infrastructure.

All their data are mixed together. But this commingling does not mean that other customers can access or modify other’s data. The multi-tenant environment is strictly controlled by software and customers can view only their commingled data.

# **Compute and Networking**

## **Amazon Elastic Compute Cloud (EC2):**

Amazon Elastic Compute is a web service that provides resizable compute capacity in the cloud. They uses Dom0, the Xen management domain which is based either on Linux openSolaris. Ec2 uses two type of storage – local storage (Instance storage)-get gets lost when instance terminates. Elastic Block Storage (EBS) –network based which is attached to the running instance and used as a boot medium.

Amazon EC2 instance: It’s a virtual computer implemented in the cloud.

Types:

|  |  |  |  |
| --- | --- | --- | --- |
| FEATURES | Small | Large | Micro |
| MEMORY | Memory 1.7 GB | Memory : 7.5 GB | Memory : 613 MB |
| NUMBER OF CPUs | 1 EC2 CPU | 4 EC2 CPU | Up to 2 EC2 |
| INSATNCE STORAGE | 160 GB instance storage | 850 GB instance storage |  |
| PLATFORM | 32-bit platform | 64-bit platform | 32-bit or 64-bit |
| OPERATING SYSTEM | Linux or windows | Linux or windows | Linux or windows |
| I/O PERFROMANCE | I/O performance : moderate | I/O performance : high | I/O performance : low |
| COST(depends of many factors) | Cost: 70-150/month | 250-400/month | 15-25/month |

EC2 has four data centres:

US west north (CA)

US east (north VA)

EU (Ireland)

Asia pacific (Singapore)

Companies using Amazon EC2: Reddit, Heroku

**Reddit:**

Website that contains anything that is new in the internet It works as people registered to Reddit post photos, videos and articles and other people vote to it. There are various categories based on the people’s interest.

Initially Reddit had its own servers to manage its network and when its circle grew it needed to manage more servers. Its applications servers and data servers run on EC2, traffic process and traffic analysing in amazon EMR, internal search function using cloud search, customs sub Reddit style sheets on S3, back up the databases to glacier.

Older: Amazon EC2 classic: it uses a set of gateway servers to connect to client servers which requires a fixed IP addresses to client’s firewall. With EC2 classic internal IP addresses cannot be fixed, so every time the server restarts a new internal IP address is given.

## **Amazon virtual private cloud (VPC):**

Allows us to create subnets that are either routable or not routable to the internet. Helps to control the network fabric that the amazon EC2 runs on.

Put the web server on a public subnet, the database server on a private subnet. Can create subnets, routing tables, security gateways, access control lists and vpns. It provides more security than EC classic.

## **Amazon EC2 VPC:**

Amazon VPC allocates a private network subnet to client’s servers which separates those servers into different subnets inside their own. This gives the clients a lot of flexibility and control over security and traffic routing.

## **AWS EC2 reboot: (Web-source)**

Recently AWS notified its customers that massive rebooted on its EC2 due to Xen hypervisor issue and it affected many customers. It is believed that when Xen hypervisor was being updated the security experts identified a major vulnerability in Linux code known as the Bash bug which affects the hypervisor from creating new virtual machines and the only way to patch it is to reboot the system. One example of how Netflix, one of the largest AWS customers tackled the issues. Architects of Netflix made their services to be resilient by conducting random tests on their servers, so that if any one of their data centres went down, operations could be readily switched over to another data centre with barely a noticeable loss of data to customers. They also looked for ways to minimize lag time that occurred when its services needed to be rebooted/restarted. The company would often test their service by disrupting or blocking or breaking its own services [through a set of tools called the Simian Army](http://www.infoworld.com/article/2612341/amazon-web-services/three-lessons-from-netflix-on-how-to-live-in-the-cloud.html) which was designed to randomly kill Netflix services.  So when Amazon informed about the reboot Netflix was prepared. Netflix chose to use the Cassandra database. Cassandra is a NoSQL database, and its services could be spread across multiple servers in such a way that even if any one of the nodes failed, the database could kept running from other reachable servers. Over the past year, Cassandra also tested its servers using Chaos Monkey testing which had promising results.

# **Storage**

## **Amazon Simple Storage Service (S3):**

Amazon S3 is one of the famous online file storage web services. It provides storage through web services interfaces.

S3 vs DropBox:

Cost:

DB: 2GB free, 1 TB for 199.9/month.

S3: 5 GB free, 1 TB for 125/month

A new folder called bucket are created and files are stored in those buckets. It also lets the user choose the data centre region to store those buckets. Users can also different sync applications (goodsync) to sync the files in their system to their S3 account. They can also partition the storage space to allow other users to use that sub space. One slight drawback of S3 is that when a large (say 1 GB) existing file is modified and re uploaded back to the S3 the older version of the file still keeps showing for certain hours. This is because the newly uploaded file takes some time to propagate across the system. The simple way around is to rename the updated file with a new name so that the updated file is immediately visible.

## **Amazon Glacier:**

It is a low cost storage service that provides a secure and durable storage for data archiving and backup. It is designed with the same durability and reliability as Amazon S3. Glacier is mainly used to store infrequently accessed data for which the retrieval time spans to several hours (3-4 hours). Files are stored in glaciers as archives and a collection of archives are stored in a vault. Every day an inventory is generated for each vault detailing the number of contents and their size.

Cost: 1 TB $10/month

**Issues in cloud storage that customers’ face:**

* Importing data into cloud storage service:
* Security for data being transferred and at rest.
* Ways being used to save the data.
* Traditional data protection
* Total monthly cost including charges on provider’s rate card
* Process and cost if the customer decide to move the data out of the cloud and into his/her own data centre.

Amazon S3 offers server side encryption that encrypts data at rest and in transit. AWS provides its customers to use their own encryption keys which helps to make the private to public cloud transition easier. Customers can also use AWS CloudHSM, a physical hardware that will manage encryption keys for them.

AWS provides Multi Factor Authentication which requires two or more types of authentication entities. MFA is based on username-password authentication and one-time passwords.

# **Analytics**

## **Amazon Elastic Map Reduce (EMR):**

Amazon’s EMR is used for developing various applications, including financial, marketing, log analysis and bio informatics. It uses an open source framework called Hadoop to distribute the data across the clusters of Amazon EC2 services. The advantage of running EMR to traditional Hadoop is that the configuration of EMR is much easier get up than the traditional Hadoop. EMR gets better support from Amazon and there is no need to administer the clusters, the user can resize the clusters on the runtime. EMR handles the termination of clusters once the analysis is done on them, but if analysis is done through ECD2-Hadoop setup then the user must implement a way to terminate the clusters. One way to analyse big data through Hadoop is to use Hive, an open source data warehouse which runs on top of Hadoop.

## **Amazon Kinesis:**

It is new real time event processing service for streaming data at massive scale such as application logs (such as user action, code fault or any such event which should be kept track of), click stream (such patterns or navigations that web users move from one webpage to another webpage, what links are clicked heavily), etc. It is designed to capture tiny records which are aggregated into a huge amount of backend data and it can be stored in AWS cloud storage services.

# **A detailed view on Amazon’s Simple Storage Service (S3)**

Simply put the S3 is a huge (in terabytes) online hard-drive used to store large amount of data at a very low cost and there isn’t any long term contract so whenever the user decides to stop using S3 they can just remove all their files. All these actions can be accomplished using the AWS Management console.

Delete objects and buckets

Move objects across multiple buckets

View objects in the bucket

Add files or objects to the bucket

Create a Bucket

Sign up for amazon S3 account

## **Starting Amazon S3:**

Amazon is highly concerned about the security of the user’s storage files so when a new user wants to use S3 he/she should open an Amazon S3 account online. Users are given with their Access keys and a secret access key.

Access Key ID:

Access key is a username which is not essentially to be kept secret such that even if a third party comes to know the user’s access key that party cannot access/modify the user files.

Secret Access Key:

The secret access key is a password which is a long string of characters which must be kept confidential. When a user sends a request to access Amazon S3 storage a digital signature is generated and sent along with the request and the user’s access key ID. When the amazon server receives the request they use the user’s access key ID to find the corresponding secret key and try to validate the digital signature. If the validation is successful then they confirm that the request is sent by the valid user and the request is processed.

Temporary Access Keys:

The user can also have temporary access keys in addition to access keys and secret access keys. These temporary access keys have security tokens which are submitted when using the temporary access keys. The advantage of using temporary access keys is that they have a limited life time and once they are expired they cannot be used later on. These temporary access keys can be used in less secure environments and can be distributed to select user to grant them temporary access the resources.

Once the account gets activated a notification is sent to the user’s email indicating that the account is available to be used.

**Create a bucket and add files:**

Once the amazon S3 account is created the user can upload data into the cloud storage. In order to store the data in the cloud a folder should be created. The folders are referred as buckets in the S3. After creating a bucket data can be uploaded into the bucket. The user can create up to 100 buckets but any number of objects can be stored in each of the buckets. However a bucket cannot be created within another bucket. Each bucket names should be unique. There are certain rules for naming buckets. The basic operations that can be performed on the buckets are get, put, list and delete bucket. The S3 uses the APIs to manage requests to add objects into the buckets.

**View and move objects across the buckets:**

Just like a normal file explorer the Amazon APIs provide a simple way to view and access objects in the buckets.

**Delete objects and buckets:**

Bucket names are globally unique regardless of the region under which the buckets are created. Since the bucket names should be unique a bucket name can be reused only if that is deleted. However deleted bucket names cannot be reused most of the time since another user might have created a bucket with the deleted bucket name so it is always recommended to use a new bucket name when creating new buckets.

## **Authenticating requests in the Amazon S3:**

Previously S3 used AWS Signature version 2 for signing and authenticating requests. Now the S3 has started implementing Signature version 4 for signing and authenticating requests to the cloud storage. The signing and authentication process is handled automatically or manually. The authentication uses the contents in the request, access key ID and signature developed using secret access key.

Amazon Command Line Interface (CLI) and AWS SDKs tools can automatically sign user’s requests based on the credential information that the user specifies when configuring those tools.

## **Signing and authentication:**

### Signing:

If the user makes any direct HTTP or HTTPS calls to any AWS’s service then the user should manually sign and send those request to the AWS.

Steps for signing a request manually:

* A signature is calculated by combining the information in the requests and user’s secret access key. These information include the type of AWS service the user is using, the region in which the user storage was created, and a time stamp of when the request was created.
* The signature is appended with the user’s access key Id and sent to the AWS.
* When AWS receives those request, it performs the same steps previously used to calculate the signature. When creating the signature the AWS uses the user’s access key ID to get the corresponding secret access key.
* When both signature matches, the request is processed or else the request is denied.

Generation of signature:

To generate a signature the user needs some contents from the request and a signing key. The contents that are selected from the request must be same as the content that will be used by AWS to validate the signature. The signing key is a combination of various keys. The advantage of the signing key is that its validity will expire within seven days after being created and the scope of the signing key is limited to the region and service.

When both the signing key and specific contents are available the signing key is used to calculate the hash-based authentication code (HMAC) of the specific contents. This HMAC code is used to sign the request and then the request is sent to the AWS.

Signing Key

A string based on selected elements of the request

Signature

The signing key is generated using the following keys.

DateKey = HMAC-SHA256 (“AWS4” + “<SecretAccessKey>”, “<yyyymmdd>”)

DateRegionKey = HMAC-SHA256 (“<DateKey >”, “<aws-region>”)

DateRegionServiceKey = HMAC-SHA256 (“<DateRegionKey >”, “<aws-region>”)

SigningKey = HMAC-SHA256 (“<DateRegionServiceKey >”, “aws4\_request”)

The signature is generated using the following equation.

Signature = HMAC-SHA256 (SigningKey, Selected\_contents\_from\_request)

### Authentication:

**Need for enabling authentication:**

**Verification of the identity of the requester:**

Being a cloud storage system anyone with access to other user’s access key would try to access their files. So the authentication is required. The authenticated request contains signature which can be validated only if the secret success key is known.

**In-transit data protection:**

In order to prevent malicious users from tampering the request while it being transmitted to AWS, various request elements are used to calculate the signature. When AWS receives the request, it calculates the signature by using the same request elements. If any request element received by AWS does not match with the elements that was used to calculate the signature the request will be denied.

**Protection against potential replay attack:**

A request much reach within 15 minutes of the timestamp in the request or else the AWS S3 servers will deny the request.

## **Authentication Methods:**

### HTTP Authorization Header:

The HTTP Authorization header is the most common method of signing the S3 request. When a new object is created and uploaded into the bucket, a request is sent to the AWS. The request is in form of a signature which also contains the checksum of the data. It can used to identify if any data is lost during transmission. In case of large uploads, files are chunked and request is generated for each chunks which also contains a checksum.

The following is an example of the Authentication Header value.

Authorization: AWS4-HMAC-SHA256 Credential = NDHUQMLOEBAF3EXAMPLE/21040912/us-west-2/s3/aws4\_request,

SignedHeaders = host; range; x-amz-date,

Signature = fyw1475abgrt257ateon634agtendi21783abfkhurva5781avfyin279necgt187

AWS4-HMAC-SHA256: this is the algorithm used to calculate the signature. AWMMddS4 indicates that the signature used Amazon Signature Version 4, HMAC-SHA256 is the signing algorithm.

Credential: the credentials include the user’s access key ID and scope information such as date (format-yyyymmdd), region (format-<aws-region>) and services (format-<aws-service>)

SignatureHeaders: a semicolon separated list of headers are used to compute Signature. The SignatureHeaders include header names only and they must be in lowercases.

Signature: it is a 256-bit signature expressed as 64 lowercase hexadecimal characters.

When the request is received by the AWS S3 it recreates the original string using the Authorization Header and date header.

**Transferring payloads using authentication headers:**

To prevent data loss during transmission the contents of must be protected. Depending upon the size of the payload or request there are two ways of transmission

**Compute checksum of the entire payload prior to transmission:**

The file must be read first in order to compute the payload hash value for calculating the signature and file must be again read to create a request and transmitting. This approach can be used for payload with small size.

**Payload transmission in chunks:**

The file must be broken into chunks of fixed size or variable size. For the first chuck of data a seed signature is generated which uses only the request headers. In the second chunk of data the signature of the first chunk is generated and for next subsequent chunks its previous chunks signature is generated. In this way payloads that are lost during transmission can be identified and they can requested to be resent. The final chuck contains 0 bytes of data and the signature of that payload.

### Query string parameters:

A query string can be used to express a request in a very short way such as pre-signed URL. The query parameters are used to provide request information along with the authentication information.

The pseudo code for the pre-signed URL is as following,

URL = [https://s3.amazonaws.com/<bucketNAme>/<fileName](https://s3.amazonaws.com/%3cbucketNAme%3e/%3cfileName)>

Amz-Algorithm = AWS4-HMAC-SHA256

Amz-Credential = <user Access Key ID>/<currentDate>/<Region>/<Amazon-

Service>/aws4\_request

Amz-Date = <date in ISO 8601 format>

Amz-Expirees = <time\_to\_expire\_in\_seconds>

Amz-SignedHeaders = <List\_of\_headers>

Amz-Signature = <Signature\_value>

These query parameters are used to provide authentication information such as signatures to help AWS to calculated the same signature using there parameters.

The diagram illustrates the signature calculation process.

String to be signed

Canonical Request

Signature

Canonical Request:

To start the sign process a string needs to be created which contains the request information in a standardized (canonical) format. By the formatting the request into canonical form before signing will help AWS to generate the same signature when it receives the request.

The pseudo code of the canonical request is as follows,

CanonicalRequest =

HTTPRequestMethod + ‘\n’ +

CanonicalURI + ‘\n’ +

CanonicalQueryString + ‘\n’ +

CanonicalHeaders + ‘\n’ +

SignatureHeaders + ‘\n’ +

HexEncode (Hash (RequestPayload))

HTTPRequestMethod :

Format: “GET”

This parameter get the request’s method such as GET, PUT, POST

CanonicalURI:

Format: (URI-ENCODE (<resource>))

This parameter gets the URI-ENCODE version of the absolute path of the URI. If the absolute path is empty then ‘/’ is used. The normal URI start from HTTP host to the question mark parameter.

CanonicalQueryString:

Format: Uri-Encode (<QueryParameter1>) + “=” + Uri-Encode (<value>) + “&” +

Uri-Encode (<QueryParameterN>) + “=” + Uri-Encode (<value>)

This parameter gets the list of all query parameters and their corresponding values. There are certain rules to construct the canonical query string. If there is not canonical query string constructed then a balck space is added.

CanonicalHeaders:

Format: Lowercase (<HeaderName1>) + “:” + Trim(<value>) + “\n”

Lowercase (<HeaderNameN>) + “:” + Trim(<value>) + “\n”

This parameter uses a list of HTTP headers that was included as a part of AWS request. Different services might require different headers. CanonicalHeaders convert all header names into lowercases and trim if there are excess spaces into a single valued space. Below is the pseudo code to construct canonical header lists.

CanonicalHeaders = CanonicalHeadersEntry0 + CanonicalHeadersEntry+…….. +

CanonicalHeadersEntryN

CanonicalHeadersEntry = Lowercase (HeaderName) + ‘:’ + Trimall (HeaderValue) + ‘\n’

SignatureHeaders:

Format: Lowercase (<HeaderName1>) + “;” ……… + Lowercase (<HeaderNameN>)

This parameter lists all the names of headers that were included in the canonical headers. By reading this list the AWS can identify which headers were used in signing process. Below is the pseudo code of the Signature headers,

Content-type;host;x-amz-date

HexEncode (Hash (RequestPayload)):

Format: Lowercase (HexEncode (Hash (RequestPayload)))

A hash value is generated using hash functions like SHA256 for the payloads which is present in the body of the HTTP or HTTPS requests.

Consider a payload which contains the following strings,

Action = ListUsers&Version=2014-09-12

By using the SHA256 hash function the payload is hashed into “heya178htrab1485369abgrthj75369abgtywb12avrpl5afr5896472abof14a5”

**String to be signed:**

When a request is generated a metadata information about that request and canonical request are included in the string to be signed process.

The structure of string to be signed is as follows,

StringToSign =

Algorithm + ‘\n’ +

RequestDate + ‘\n’ +

CredentialScope + ‘\n’ +

HashedCanonicalRequest

Algorithm:

The StringToSign begins with the name of the algorithm used to calculate the hash value. Mostly AWS4-HMAC-SHA256 algorithm is used.

RequestDate:

The date and time when the request was created must be added in the RequestDate parameter. The basic format uses the ISO 8601 through x-amz-date header. The format is YYYYMMDD’T’HHMMSS’Z’. This value must be same as the value used in SignatureHeader in canonical request.

CredentialScope:

Format: “<yyyymmdd>/<aws region>/s3/aws4\_request>”

This parameter includes the date of request, region the user is targeting, type of service requested and a terminating string – aws4\_request all in lowercases. The service name and the target region must be UTF-8 encoded.

HashedCanonicalRequest:

The hashed value generated from the canonical request is added to this parameter. This hash value must be of lowercase base-16 encoded.

Signature:

To generate a signature, a signed key must be derived from the user’s Secret Access Key. Once the signing key is ready the string generated from StringToSign is passed into a keyed hash function along with the signed key. The hexadecimal-encoded result from the keyed hash function is the final signature.

Deriving a Signed Key:

The secret access key of the user is used to generate a series of Hash-based Message Authentication Codes (HMACs). The same pseudo code used generate signed key is same as the pseudo code mentioned in generating a signature above.

The signature is generated by using HMAC on signed key and StringToSign string.

Signature = HMAC-SHA256 (signed key, StringToSign).

### Example:

The example below demonstrates how the signature is generated using the canonical request, signed key, User access key and secret access key.

User Access Key ID : VFESXCYUIMLO7SAMPLE

User Secret Access Key : dgethWXUtnXODE/M7MDENG/bPxRfiCYSAMPLEKEY

The user wants to download/read a file named “sample1.txt” from the user’s bucket named “sample”. The corresponding canonical request is generated using its pseudo code. The location of the bucket is servers located in the east region USA. The date and time of request is 2014/08/12 00:00:00. The request will expire in 86400 seconds.

Canonical Request:

GET

/sample1.txt

X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=VFESXCYUIMLO7SAMPLE%2F20141208%2Fus-east-1%2Fs3%2Faws4\_request&X-Amz-

Date=20141208T000000Z&X-Amz-Expires=86400&X-Amz-SignedHeaders=host

host:samplebucket.s3.amazonaws.com

host

UNSIGNED-PAYLOAD

A signing key is generated using the signing key’s pseudo code and some contents from the request.

Signing Key:

Signing key = HMAC-SHA256(HMAC-SHA256(HMAC-SHA256(HMAC-SHA256

("AWS4" + " dgethWXUtnXODE/M7MDENG/bPxRfiCYSAMPLEKEY

","20141208"),"us-east-1"),"s3"),"aws4\_request")

Signature:

Using the HMAC-SHA256 hashed key the signature is generated.

Signature = aeeed9bbccd4d02ee5c0109b86d86835f995330da4c265957d157751f604d404

This signature is coupled with the request and sent to the AWS servers which try to generate the same signature using the selected contents in the request. If signature matches then request is processed and transmitted using the same process.

# **Reference**

**Web resources:**

<http://docs.aws.amazon.com/general/latest/gr/sigv4-signed-request-examples.html>

<http://docs.aws.amazon.com/AWSSecurityCredentials/1.0/AboutAWSCredentials.html#AccessKeys>

<http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html>

<http://docs.aws.amazon.com/general/latest/gr/signature-version-4.html>

**Research papers:**

1. Benchmarking availability and reliability in the cloud: AWS, by Nucleus Research.
2. Use of formal methods at AWS.

**Technical Blog:**

1. The Forrester Wave: Enterprise Public Cloud Platform, Q3 2013.

**White Papers:**

1. AWS: Overview of Security Processes.
2. AWS: Storage options in the AWS cloud.
3. Amazon Web Services: An Overview, by Global Knowledge.