Hosting Static Websites on AWS

Prescriptive Guidance

February 2019

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS's current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS's products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS's responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

This whitepaper covers comprehensive architectural guidance for developing, deploying, and managing static websites on Amazon Web Services (AWS) while keeping operational simplicity and business requirements in mind. We also recommend an approach that provides 1) insignificant cost of operation, 2) little or no management required, and 3) a highly scalable, resilient, and reliable website.

This whitepaper first reviews how static websites are hosted in traditional hosting environments. Then, we explore a simpler and more cost-efficient approach using Amazon Simple Storage Service (Amazon S3). Finally, we show you how you can enhance the AWS architecture by encrypting data in transit and to layer on functionality and improve quality of service by using Amazon CloudFront.

Introduction

As enterprises become more digital operations, their websites span a wide spectrum, from mission-critical e-commerce sites to departmental apps, and from business-to-business (B2B) portals to marketing sites. Factors such as business value, mission criticality, service level agreements (SLAs), quality of service, and information security drive the choice of architecture and technology stack.

The simplest form of website architecture is the *static website*, where users are served static content (HTML, images, video, JavaScript, style sheets, and so on). Some examples include brand microsites, marketing websites, and intranet information pages. Static websites are straightforward in one sense, but they can still have demanding requirements in terms of scalability, availability, and service-level guarantees. For example, a marketing site for a consumer brand may need to be prepared for an unpredictable onslaught of visitors when a new product is launched.

Static Website

A static website delivers content in the same format in which it is stored. No server-side code execution is required. For example, if a static website consists of HTML documents displaying images, it delivers the HTML and images as-is to the browser, without altering the contents of the files.

Static websites can be delivered to web browsers on desktops, tablets, or mobile devices. They usually consist of a mix of HTML documents, images, videos, CSS style sheets, and JavaScript files. Static doesn’t have to mean boring—static sites can provide client-side interactivity as well. Using HTML5 and client-side JavaScript technologies such as jQuery, AngularJS, React, and Backbone, you can deliver rich user experiences that are engaging and interactive.

Some examples of static sites include:

- Marketing websites
- Product landing pages
- Microsites that display the same content to all users
- Team homepages

- A website that lists available assets (e.g., image files, video files, and press releases) allows the user to download the files as-is
- Proofs-of-concept used in the early stages of web development to test user experience flows and gather feedback

Static websites load quickly since content is delivered as-is and can be cached by a content delivery network (CDN). The web server doesn’t need to perform any application logic or database queries. They’re also relatively inexpensive to develop and host. However, maintaining large static websites can be cumbersome without the aid of automated tools, and static websites can’t deliver personalized information.

Static websites are most suitable when the content is infrequently updated. After the content evolves in complexity or needs to be frequently updated, personalized, or dynamically generated, it's best to consider a dynamic website architecture.

Dynamic Website

Dynamic websites can display dynamic or personalized content. They usually interact with data sources and web services, and require code development expertise to create and maintain. For example, a sports news site can display information based on the visitor's preferences, and use server-side code to display updated sport scores. Other examples of dynamic sites are e- commerce shopping sites, news portals, social networking sites, finance sites, and most other websites that display ever-changing information.

Core Architecture

In a traditional (non-AWS) architecture, web servers serve up static content. Typically, content is managed using a content management system (CMS), and multiple static sites are hosted on the same infrastructure. The content is stored on local disks, or on a file share on network-accessible storage. The following example shows a sample file system structure.

```
├─ css/
│  ├─ main.css
│  └─ navigation.css
├─ images/
│  ├─ banner.jpg
│  └─ logo.jpg
├─ index.html
├─ scripts/
│  ├─ script1.js
│  └─ script2.js
├─ section1.html
└─ section2.html
```

A network firewall protects against unauthorized access. It’s common to deploy multiple web servers behind a load balancer for high availability (HA) and scalability. Since pages are static, the web servers don’t need to maintain any state or session information and the load balancer doesn’t need to implement session affinity (“sticky sessions”). The following diagram shows a traditional (non-AWS) hosting environment:

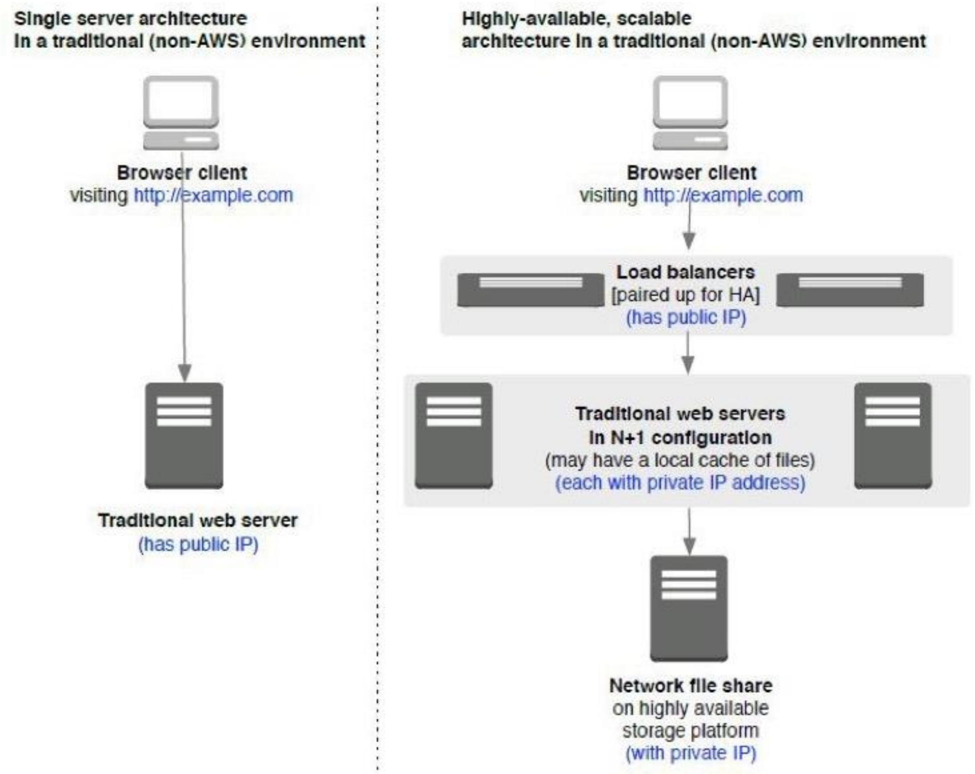

Single server architecture
in a traditional (non-AWS) environment
Browser client
visiting http://example.com
Traditional web server
(has public IP)
Highly-available, scalable
architecture in a traditional (non-AWS) environment
Browser client
visiting http://example.com
Load balancers
[paired up for HA]
(has public IP)
Traditional web servers
in N+1 configuration
(may have a local cache of files)
(each with private IP address)
Network file share
on highly available
storage platform
(with private IP)

Figure 1: Basic architecture of a traditional hosting environment

Moving to an AWS Architecture

To translate a traditional hosting environment to an AWS architecture, you could use a "lift-and-shift" approach where you substitute AWS services instead of using the traditional environment.

In this approach, you can substitute the following AWS services:

- Amazon Elastic Compute Cloud (Amazon EC2) to run Linux or Windows based servers
- Elastic Load Balancing (ELB) to load balance and distribute the web traffic
- Amazon Elastic Block Store (Amazon EBS) or Amazon Elastic File System (Amazon EFS) to store static content.
- Amazon Virtual Private Cloud (Amazon VPC) to deploy Amazon EC2 instances. Amazon VPC is your isolated and private virtual network in the AWS Cloud and gives you full control over the network topology, firewall configuration, and routing rules.
- Web servers can be spread across multiple Availability Zones for high availability, even if an entire data center were to be down.
- AWS Auto Scaling automatically adds servers during high traffic periods and scales back when traffic decreases.

The following diagram shows the basic architecture of a "lift and shift" approach.

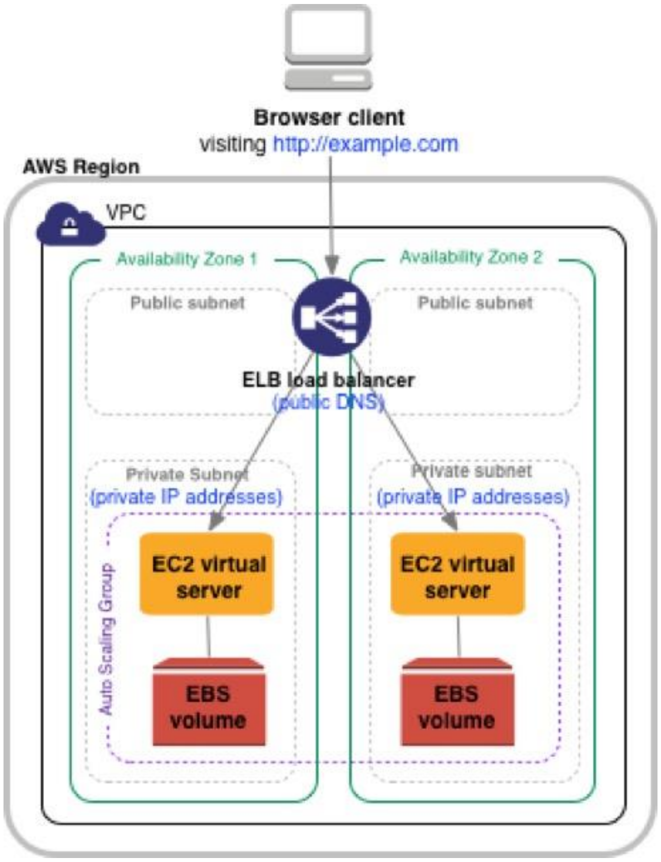


Figure 2: AWS architecture for a "Lift and Shift"

Using this AWS architecture, you gain the security, scalability, cost, and agility benefits of running in AWS. This architecture benefits from AWS world-class infrastructure and security operations. By using Auto Scaling, the website is ready for traffic spikes, so you are prepared for product launches and viral websites. With AWS, you only pay for what you use, and there's no need to over-provision for peak capacity. In addition, you gain increased agility because AWS services are available on demand. (Compare this to the traditional process in which provisioning servers, storage, or networking can take weeks.) You don't have to manage infrastructure, so this frees up time and resources to create business differentiating value.

AWS challenges traditional IT assumptions and enables new "cloud-native" architectures. You can architect a modern static website without needing a single web server.

Use Amazon S3 Website Hosting to Host Without a Single Web Server

Amazon Simple Storage Service (Amazon S3) can host static websites without a need for a web server. The website is highly performant and scalable at a fraction of the cost of a traditional web server. Amazon S3 is storage for the cloud, providing you with secure, durable, highly scalable object storage. A simple web services interface allows you to store and retrieve any amount of data from anywhere on the web.[1]

You start by creating an Amazon S3 bucket, enabling the Amazon S3 website hosting feature, and configuring access permissions for the bucket. After you upload files, Amazon S3 takes care of serving your content to your visitors.

Amazon S3 provides HTTP web-serving capabilities, and the content can be viewed by any browser. You must also configure Amazon Route 53, a managed Domain Name System (DNS) service, to point your domain to your Amazon S3 bucket. Figure 3 illustrates this architecture, where *http://example.com* is the domain.

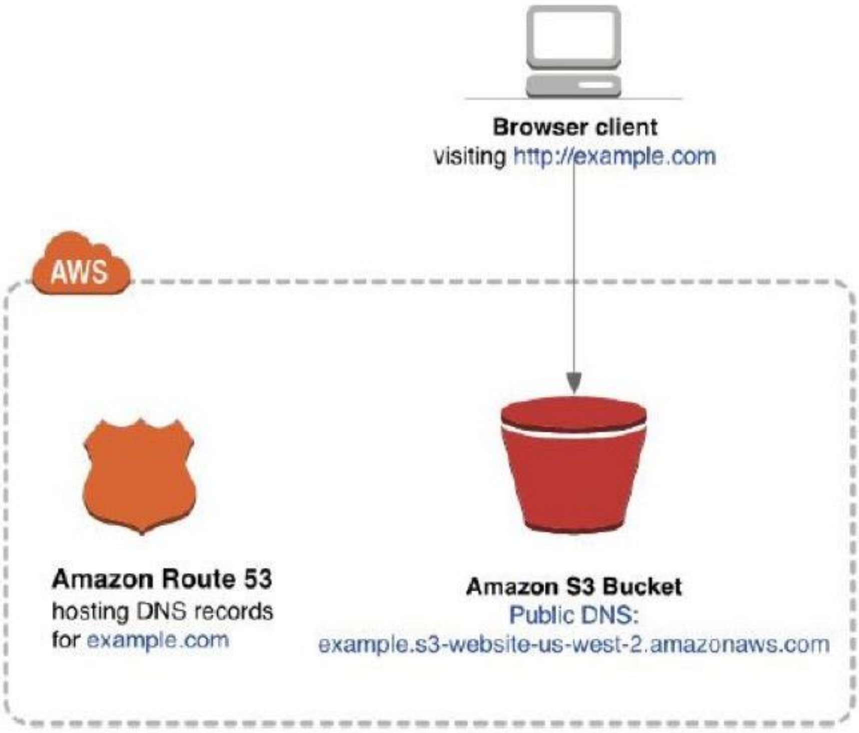


Figure 3: Amazon S3 website hosting

In this solution, there are no Windows or Linux servers to manage, and no need to provision machines, install operating systems, or fine-tune web server configurations. There's also no need to manage storage infrastructure (e.g., SAN, NAS) because Amazon S3 provides practically limitless cloud-based storage. Fewer moving parts means fewer troubleshooting headaches.

Scalability and Availability

Amazon S3 is inherently scalable. For popular websites, Amazon S3 scales seamlessly to serve thousands of HTTP or HTTPS requests per second without any changes to the architecture.

In addition, by hosting with Amazon S3, the website is inherently highly available. Amazon S3 is designed for 99.999999999% durability, and carries a service level agreement (SLA) of 99.9% availability. Amazon S3 gives you access to the same highly scalable, reliable, fast, and inexpensive infrastructure that Amazon uses to run its own global network of websites. As soon as you upload files to Amazon S3, Amazon S3 automatically replicates your content across multiple data centers. Even if an entire AWS data center were to be impaired, your static website would still be running and available to your end users.

Compare this solution with traditional non-AWS costs for implementing "active-active" hosting for important projects. Active-active, or deploying web servers in two distinct data centers, is prohibitive in terms of server costs and engineering time. As a result, traditional websites are usually hosted in a single data center, because most projects can't justify the cost of "active-active" hosting.

Encrypt Data in Transit

We recommend you use HTTPS to serve static websites securely. HTTPS is the secure version of the HTTP protocol that browsers use when communicating with websites. In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS). TLS protocols are cryptographic protocols designed to provide privacy and data integrity between two or more communicating computer applications. HTTPS protects against man-in-the-middle (MITM) attacks. MITM attacks intercept and maliciously modify traffic.

Historically, HTTPS was used for sites that handled financial information, such as banking and e-commerce sites. However, HTTPS is now becoming more of the norm rather than the exception. For example, the percentage of web pages loaded by Mozilla Firefox using HTTPS has increased from 49% to 75% in the past two years.[2]

AWS Certificate Manager (ACM) is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer (SSL)/TLS certificates for use with AWS services and your internal connected resources. See Using HTTPS with Amazon CloudFront for more implementation information.

Configuration Basics

Configuration involves these steps:

1. Open the AWS Management Console.
2. On the Amazon S3 console, create an Amazon S3 bucket.
    a. Choose the AWS Region in which the files will be geographically stored.[3] Select a Region based on its proximity to your visitors, proximity to your corporate data centers, and/or your regulatory or compliance requirements (e.g., some countries have restrictive data residency regulations).

    b. Choose a bucket name that complies with DNS naming conventions. If you plan to use your own custom domain/subdomain, such as *example.com* or *www.example.com*, your bucket name must be the same as your domain/subdomain. For example, a website available at *http://www.example.com* must be in a bucket named www.example.com*.*

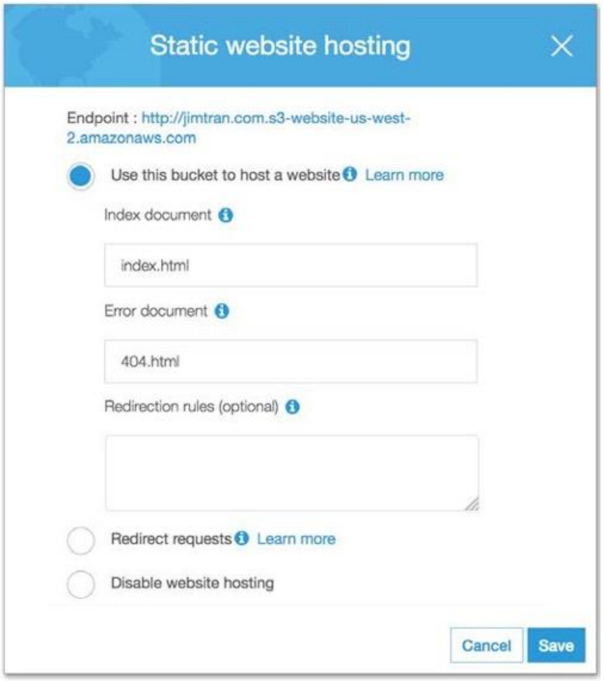
    Note: Each AWS account can have a maximum of 1000 buckets.
3. Toggle on the static website hosting feature for the bucket. This generates an Amazon S3 website endpoint.

You can access your Amazon S3-hosted website at the following URL:

```
http://<bucket-name>.s3-website-<AWS-region>.amazonaws.com
```

Domain Names

For small, non-public websites, the Amazon S3 website endpoint is probably adequate. You can also use internal DNS to point to this endpoint. For a public facing website, we recommend using a custom domain name instead of the provided Amazon S3 website endpoint. This way, users can see user-friendly URLs in their browsers. If you plan to use a custom domain name, your bucket name must match the domain name. For custom root domains (such as *example.com*), only Amazon Route 53 can configure a DNS record to point to the Amazon S3 hosted website. For non-root subdomains (such as *www.example.com*), any DNS service (including Amazon Route 53) can create a CNAME entry to the subdomain. See the Amazon Simple Storage Service Developer Guide for more details on how to associate domain names with your website.


Static website hosting

Endpoint : http://jimtran.com.s3-website-us-west-2.amazonaws.com

● Use this bucket to host a website ℹ Learn more

Index document ℹ

index.html

Error document ℹ

404.html

Redirection rules (optional) ℹ

○ Redirect requests ℹ Learn more

○ Disable website hosting

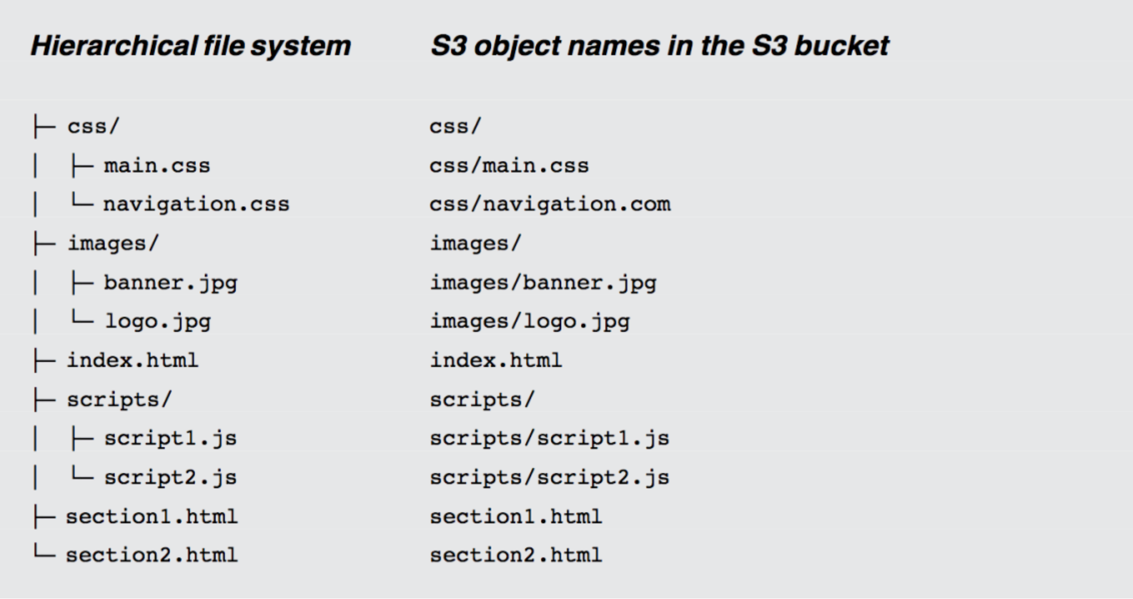
Figure 4: Configuring static website hosting using Amazon S3 console

The Amazon S3 website hosting configuration screen in the Amazon S3 console presents additional options to configure. Some of the key options are as follows:

- You can configure a default page that users see if they visit the domain name directly (without specifying a specific page).[4] You can also specify a custom `404 - Page Not Found` error page if the user stumbles onto a non-existent page.
- You can enable logging to give you access to the raw web access logs. (By default, logging is disabled.)
- You can add tags to your Amazon S3 bucket. These tags help when you want to analyze your AWS spend by project.

Amazon S3 Object Names

In Amazon S3, a bucket is a flat container of objects. It doesn't provide a hierarchical organization the way the file system on your computer does. However, there is a straightforward mapping between a file system's folders/files to Amazon S3 objects. The example that follows shows how folders/files are mapped to Amazon S3 objects. Most third-party tools, as well as the AWS Management Console and AWS Command Line Interface (AWS CLI), handle this mapping transparently for you. For consistency, we recommend that you use lowercase characters for file and folder names.



Hierarchical file system	***S3 object names in the S3 bucket***
`├ css/`	`css/`
`│  ├ main.css`	`css/main.css`
`│  └ navigation.css`	`css/navigation.com`
`├ images/`	`images/`
`│  ├ banner.jpg`	`images/banner.jpg`
`│  └ logo.jpg`	`images/logo.jpg`
`├ index.html`	`index.html`
`├ scripts/`	`scripts/`
`│  ├ script1.js`	`scripts/script1.js`
`│  └ script2.js`	`scripts/script2.js`
`├ section1.html`	`section1.html`
`└ section2.html`	`section2.html`

Uploading Content

On AWS, you can design your static website using your website authoring tool of choice. Most web design and authoring tools can save the static content on your local hard drive. Then, upload the HTML, images, JavaScript files, CSS files, and other static assets into your Amazon S3 bucket. To deploy, copy any new or modified files to the Amazon S3 bucket. You can use the AWS API, SDKs, or CLI to script this step for a fully automated deployment.

You can upload files using the AWS Management Console. You can also use AWS partner offerings such as CloudBerry, S3 Bucket Explorer, S3 Fox, and other visual management tools. The easiest way, however, is to use the AWS CLI. The S3 sync command recursively uploads files and synchronizes your Amazon S3 bucket with your local folder.[5]

Making Your Content Publicly Accessible

For your visitors to access content at the Amazon S3 website endpoint, the Amazon S3 objects must have the appropriate permissions. Amazon S3 enforces a security-by-default policy. New objects in a new bucket are private by default. For example, an Access Denied error appears when trying to view a newly uploaded file using your web browser. To fix this, configure the content as publicly accessible. It's possible to set object-level permissions for every individual object, but that quickly becomes tedious. Instead, define an Amazon S3 bucket-wide policy.

The following sample Amazon S3 bucket policy enables everyone to view all objects in a bucket:

```
{
   "Version":"2012-10-17",
   "Statement":[{
      "Sid":"PublicReadGetObject",
      "Effect":"Allow",
      "Principal": "*",
      "Action":["s3:GetObject"],
      "Resource":["arn:aws:s3:::S3_BUCKET_NAME_GOES_HERE/*"]
    }
  ]
}
```

This policy defines who can view the contents of your Amazon S3 bucket. See Managing Administrative Access to Your AWS Resources for the AWS Identity and Access Management (IAM) policies to manage permissions for your team members.

Together, Amazon S3 bucket policies and IAM policies give you fine-grained control over who can manage and view your website.

Requesting a Certificate through ACM

You can create and manage public, private, and imported certificates with ACM. This section focuses on creating and using public certificates to be used with ACM-integrated services, specifically Amazon Route 53 and Amazon CloudFront.

To request a certificate:

1. Add in the qualified domain names (e.g. example.com) you want to secure with a certificate.
2. Select a validation method. ACM can validate ownership by using DNS or by sending email to the contact addresses of the domain owner.
3. Review the domain names and validation method.
4. Validate. If you used the DNS validation method, you must create a CNAME record in the DNS configuration for each of the domains. If the domain is not currently managed by Amazon Route 53, you can choose to export the DNS configuration file and input that information in your DNS web service. If the domain is managed by Amazon Route 53, you can click “Create record in Route 53” and ACM can update your DNS configuration for you.

After validation is complete, return to the ACM console. Your certificate status changes from Pending Validation to Issued.

Low Costs Encourage Experimentation

Amazon S3 costs are storage plus bandwidth. The actual costs depend upon your asset file sizes, and your site’s popularity (the number of visitors making browser requests). There’s no minimum charge and no setup costs.

When you use Amazon S3, you pay for what you use. You’re only charged for the actual Amazon S3 storage required to store the site assets. These assets include HTML files, images, JavaScript files, CSS files, videos, audio files, and any other downloadable files. Your bandwidth charges depend upon the actual site traffic. More specifically, the number of bytes that are delivered to the website visitor in the HTTP responses. Small websites with few visitors have minimal hosting costs.

Popular websites that serve up large videos and images incur higher bandwidth charges. The Estimating and Tracking AWS Spend section describes how you can estimate and track your costs.

With Amazon S3, experimenting with new ideas is easy and cheap. If a website idea fails, the costs are minimal. For microsites, publish many independent microsites at once, run A/B tests, and keep only the successes.

Evolving the Architecture with Amazon CloudFront

Amazon CloudFront content delivery web service integrates with other AWS products to give you an easy way to distribute content to users on your website with low latency, high data transfer speeds, and no minimum usage commitments.

Factors Contributing to Page Load Latency

To explore factors that contribute to latency, we use the example of a user in Singapore visiting a web page hosted from an Amazon S3 bucket in the US West (Oregon) Region in the United States. From the moment the user visits a web page to the moment it shows up in the browser, several factors contribute to latency:

- **FACTOR (1)** Time it takes for the browser (Singapore) to request the web page from Amazon S3 (US West [Oregon] Region).
- **FACTOR (2)** Time it takes for Amazon S3 to retrieve the page contents and serve up the page.
- **FACTOR (3)** Time it takes for the page contents (US West [Oregon] Region) to be delivered across the Internet to the browser (Singapore).
- **FACTOR (4)** Time it takes for the browser to parse and display the web page.

This latency is illustrated in the following figure.

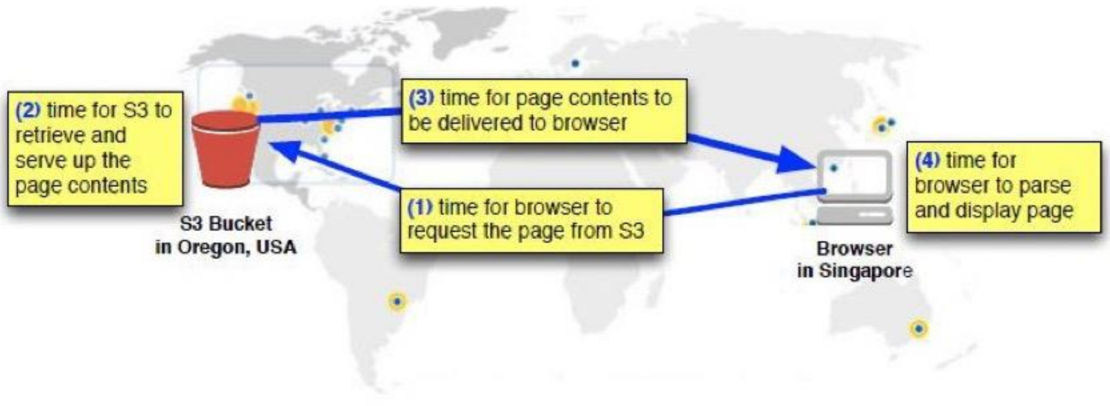

(2) time for S3 to retrieve and serve up the page contents
S3 Bucket in Oregon, USA
(3) time for page contents to be delivered to browser
(1) time for browser to request the page from S3
Browser in Singapore
(4) time for browser to parse and display page

Figure 5: Factors affecting page load latency

AWS addresses **FACTOR (2)** by optimizing Amazon S3 to serve up content as quickly as possible. You can improve **FACTOR (4)** by optimizing the actual page content (e.g., minifying CSS and JavaScript, using efficient image and video formats). However, page-loading studies consistently show that most latency is due to **FACTOR (1)** and **FACTOR (3)**.[6] Most of the delay in accessing pages over the internet is due to the round-trip delay associated with establishing TCP connections (the infamous three-way TCP handshake) and the time it takes for TCP packets to be delivered across long Internet distances).

In short, serve content as close to your users as possible. In our example, users in the USA will experience relatively fast page load times, whereas users in Singapore will experience slower page loads. Ideally, for the users in Singapore, you would want to serve up content as close to Singapore as possible.

Speeding Up Your Amazon S3 Based Website Using Amazon CloudFront

Amazon CloudFront is a CDN that uses a global network of edge locations for content delivery. Amazon CloudFront also provides reports to help you understand how users are using your website.

As a CDN, Amazon CloudFront can distribute content with low latency and high data transfer rates. There are multiple CloudFront edge locations all around the world. Therefore, no matter where a visitor lives in the world, there is an Amazon CloudFront edge location that is relatively close (from an Internet latency perspective).

The Amazon CloudFront edge locations cache content from an origin server and deliver that cached content to the user. When creating an Amazon CloudFront distribution, specify your Amazon S3 bucket as the origin server. The Amazon CloudFront distribution itself has a DNS. You can refer to it using a CNAME if you have a custom domain name. To point the A record of a root domain to an Amazon CloudFront distribution, you can use Amazon Route 53 alias records, as illustrated in the following diagram.

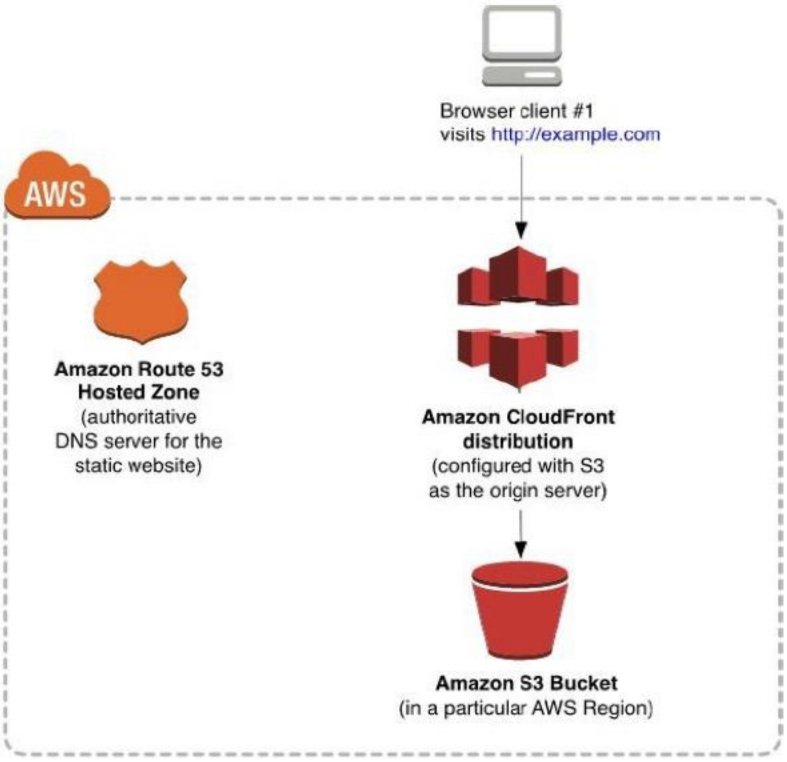


Figure 6: Using Amazon Route 53 alias records with an Amazon CloudFront distribution

Amazon CloudFront also keeps persistent connections with your origin servers so that those files can be fetched from the origin servers as quickly as possible. Finally, Amazon CloudFront uses additional optimizations (e.g., wider TCP initial congestion window) to provide higher performance while delivering your content to viewers.

When an end user requests a web page using that domain name, CloudFront determines the best edge location to serve the content. If an edge location doesn't yet have a cached copy of the requested content, CloudFront pulls a copy from the Amazon S3 origin server and holds it at the edge location to fulfill future requests. Subsequent users requesting the same content from that edge location experience faster page loads because that content is already cached. The following diagram shows the flow in detail.

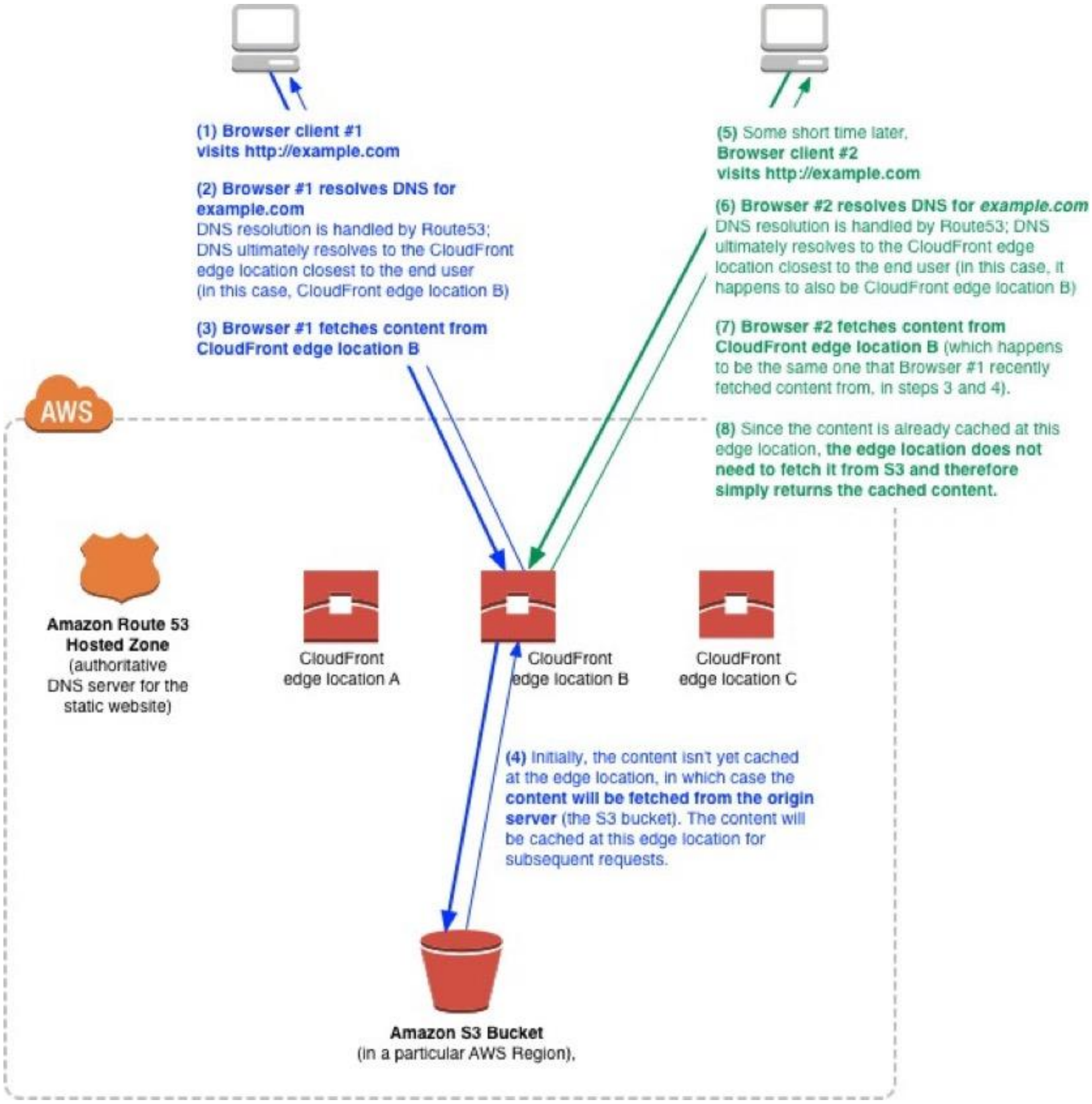

(1) Browser client #1
visits http://example.com
(2) Browser #1 resolves DNS for
example.com
DNS resolution is handled by Route53;
DNS ultimately resolves to the CloudFront
edge location closest to the end user
(in this case, CloudFront edge location B)
(3) Browser #1 fetches content from
CloudFront edge location B
(5) Some short time later,
Browser client #2
visits http://example.com
(6) Browser #2 resolves DNS for example.com
DNS resolution is handled by Route53; DNS
ultimately resolves to the CloudFront edge
location closest to the end user (in this case, it
happens to also be CloudFront edge location B)
(7) Browser #2 fetches content from
CloudFront edge location B (which happens
to be the same one that Browser #1 recently
fetched content from, in steps 3 and 4).
(8) Since the content is already cached at this
edge location, the edge location does not
need to fetch it from S3 and therefore
simply returns the cached content.
AWS
Amazon Route 53
Hosted Zone
(authoritative
DNS server for the
static website)
CloudFront
edge location A
CloudFront
edge location B
CloudFront
edge location C
(4) Initially, the content isn't yet cached
at the edge location, in which case the
content will be fetched from the origin
server (the S3 bucket). The content will
be cached at this edge location for
subsequent requests.
Amazon S3 Bucket
(in a particular AWS Region),

Figure 7: How Amazon CloudFront caches content

Using HTTPS with Amazon CloudFront

You can configure Amazon CloudFront to require that viewers use HTTPS to request your objects, so that connections are encrypted when Amazon CloudFront communicates with viewers. You can also configure Amazon CloudFront to use HTTPS to get objects from your origin, so that connections are encrypted when Amazon CloudFront communicates with your origin. If you want to require HTTPS for communication between Amazon CloudFront and Amazon S3, you must change the value of the Viewer Protocol Policy to Redirect HTTP to HTTPS or HTTPS Only.

We recommend using Redirect HTTP to HTTPS. Viewers can use both protocols. HTTP GET and HEAD requests are automatically redirected to HTTPS requests. Amazon CloudFront returns HTTP status code 301 (Moved Permanently) along with the new HTTPS URL. The viewer then resubmits the request to Amazon CloudFront using the HTTPS URL.

Amazon CloudFront Reports

Amazon CloudFront includes a set of reports that provide insight into and answers to the following questions:

- What is the overall health of my website?
- How many visitors are viewing my website?
- Which browsers, devices, and operating systems are they using?
- Which countries are they coming from?
- Which websites are the top referrers to my site?
- What assets are the most popular ones on my site?
- How often is CloudFront caching taking place?

Amazon CloudFront reports can be used alongside other online analytics tools, and we encourage the use of multiple reporting tools. Note that some analytics tools may require you to embed client-side JavaScript in your HTML pages. Amazon CloudFront reporting does not require any changes to your web pages. See the Amazon CloudFront Developer Guide for more information on reports.

Estimating and Tracking AWS Spend

With AWS, there is no upper limit to the amount of Amazon S3 storage or network bandwidth you can consume. You pay as you go and only pay for actual usage. Because you’re not using web servers in this architecture, you have no licensing costs or concern for server scalability or utilization.

Estimating AWS Spend

To estimate your monthly costs, you can use the AWS Simple Monthly Calculator. Pricing sheets for Amazon Route 53, Amazon S3, and Amazon CloudFront are available online.[7] AWS pricing is Region specific. See the following links for the most recent pricing information: Amazon Route 53, Amazon CloudFront, Amazon S3.

Tracking AWS Spend

The AWS Cost Explorer can help you track cost trends by service type. It's integrated in the AWS Billing console and runs in your browser. The Monthly Cost by Service chart allows you to see a detailed breakdown by service. The Daily Cost report helps you track your spending as it happens. If you configured tags for your Amazon S3 bucket, you can filter your reports against specific tags for cost allocation purposes. See Using the Default Cost Explorer Reports.

Integration with Your Continuous Deployment Process

Your website content should be managed using version control software (such as Git, Subversion, or Microsoft Team Foundation Server) to make it possible to revert to older versions of your files.[8] AWS offers a managed source control service called AWS CodeCommit that makes it easy to host secure and private Git repositories. Regardless of which version control system your team uses, consider tying it to a continuous build/integration tool so that your website automatically updates whenever the content changes.

For example, if your team is using a Git-based repository for version control, a Git post-commit hook can notify your continuous integration tool (e.g., Jenkins) of any content updates. At that point, your continuous integration tool can perform the actual deployment to synchronize the content with Amazon S3 (using either the AWS CLI or the Amazon S3 API), and notify the user of the deployment status.

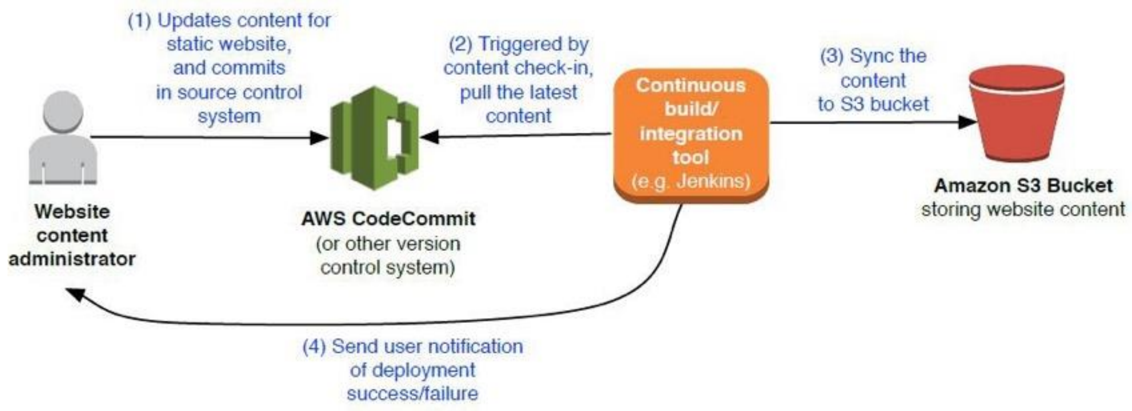

(1) Updates content for static website, and commits in source control system
(2) Triggered by content check-in, pull the latest content
(3) Sync the content to S3 bucket
(4) Send user notification of deployment success/failure
Website content administrator
AWS CodeCommit (or other version control system)
Continuous build/ integration tool (e.g. Jenkins)
Amazon S3 Bucket storing website content

Figure 8: Example continuous deployment process

If you don’t want to use version control, then be sure to periodically download your website and back up the snapshot. The AWS CLI lets you download your entire website with a single command:

```
aws s3 sync s3://bucket /my_local_backup_directory
```

Access Logs

Access logs can help you troubleshoot or analyze traffic coming to your site. Both Amazon CloudFront and Amazon S3 give you the option of turning on access logs. There’s no extra charge to enable logging, other than the storage of the actual logs. The access logs are delivered on a best-effort basis; they are usually delivered within a few hours after the events are recorded.

Analyzing Logs

Amazon S3 access logs are deposited in your Amazon S3 bucket as plain text files. Each record in the log files provides details about a single Amazon S3 access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. You can open individual log files in a text editor or use a third-party tool that can interpret the Amazon S3 access log format.

CloudFront logs are deposited in your Amazon S3 bucket as GZIP-compressed text files. CloudFront logs follow the standard W3C extended log file format and can be analyzed using any log analyzer.

You can also build out a custom analytics solution using AWS Lambda and Amazon Elasticsearch Service (Amazon ES). AWS Lambda functions can be hooked to an Amazon S3 bucket to detect when new log files are available for processing. AWS Lambda function code can process the log files and send the data to an Amazon ES cluster. Users can then analyze the logs by querying Amazon ES or using the Kibana visual dashboard. Both AWS Lambda and Amazon ES are managed services, and there are no servers to manage.

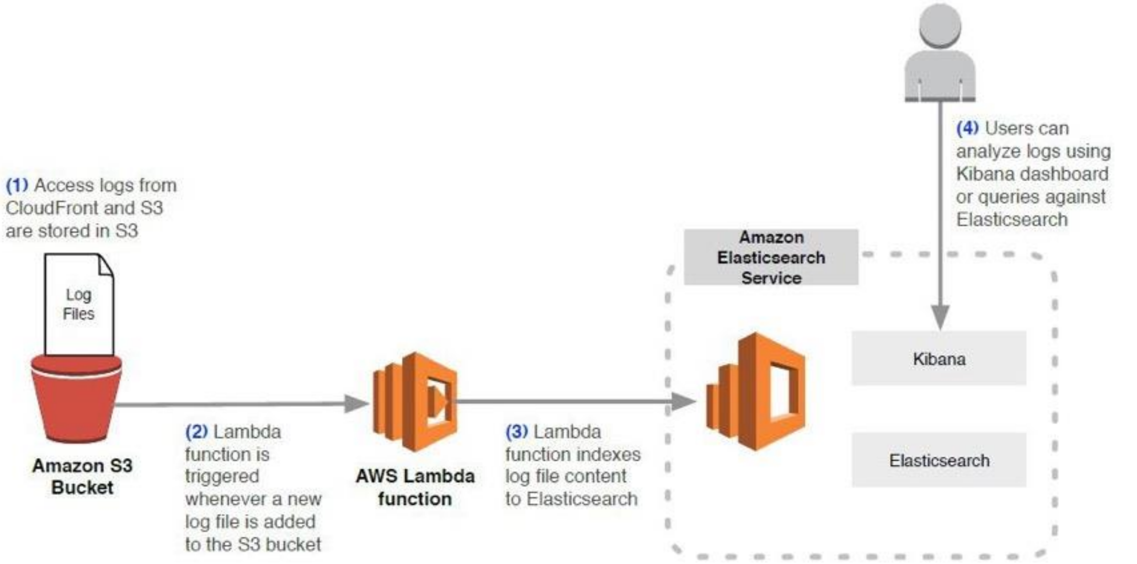

(1) Access logs from CloudFront and S3 are stored in S3
Log Files
Amazon S3 Bucket
(2) Lambda function is triggered whenever a new log file is added to the S3 bucket
AWS Lambda function
(3) Lambda function indexes log file content to Elasticsearch
Amazon Elasticsearch Service
Kibana
Elasticsearch
(4) Users can analyze logs using Kibana dashboard or queries against Elasticsearch

Figure 9: Using AWS Lambda to send logs from Amazon S3 to Amazon Elasticsearch Service

Archiving and Purging Logs

Amazon S3 buckets don’t have a storage cap, and you’re free to retain logs for as long as you want. However, an AWS best practice is to archive files into Amazon S3 Glacier.

Amazon S3 Glacier is suitable for long-term storage of infrequently accessed files. Like Amazon S3, Amazon S3 Glacier is also designed for 99.999999999% data durability, and you have practically unlimited storage. The difference is in retrieval time. Amazon S3 supports immediate file retrieval. With Amazon S3 Glacier, after you initiate a file retrieval request, there is a delay before you can start downloading the files. Amazon S3 Glacier storage costs are cheaper than S3, disks, or tape drives. See the Amazon S3 Glacier page for pricing.

The easiest way to archive data into Amazon S3 Glacier is to use Amazon S3 lifecycle policies. The lifecycle policies can be applied to an entire Amazon S3 bucket or to specific objects within the bucket (e.g., only the log files). A minute of configuration in the Amazon S3 console can reduce your storage costs significantly in the long run. Here's an example of setting up data tiering using lifecycle policies:

- Lifecycle policy #1: After X days, automatically move logs from Amazon S3 into Amazon S3 Glacier.
- Lifecycle policy #2: After Y days, automatically delete logs from Amazon S3 Glacier.

Data tiering is illustrated in the following figure.

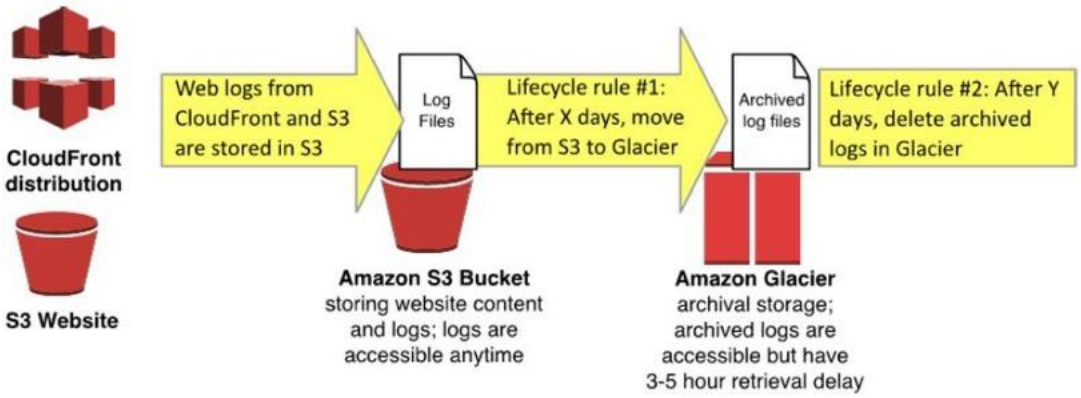

CloudFront distribution
S3 Website
Web logs from CloudFront and S3 are stored in S3
Log Files
Lifecycle rule #1: After X days, move from S3 to Glacier
Archived log files
Lifecycle rule #2: After Y days, delete archived logs in Glacier
Amazon S3 Bucket
storing website content and logs; logs are accessible anytime
Amazon Glacier
archival storage; archived logs are accessible but have 3-5 hour retrieval delay

Figure 10: Data tiering using Amazon S3 lifecycle policies

Securing Administration Access to Your Website Resources

Under the AWS shared responsibility model, the responsibility for a secure website is shared between AWS and the customer (you). AWS provides a global secure infrastructure and foundation compute, storage, networking, and database services, as well as higher level services. AWS also provides a range of security services and features that you can use to secure your assets.

As an AWS customer, you're responsible for protecting the confidentiality, integrity, and availability of your data in the cloud, and for meeting your specific business requirements for information protection. We strongly recommend working closely with your Security and Governance teams to implement the recommendations in this whitepaper as well as the ones covered in the AWS Security Best Practices whitepaper.

A benefit of using Amazon S3 and Amazon CloudFront as a serverless architecture is that the security model is also simplified. You have no operating system to harden, servers to patch, or software vulnerabilities to generate concern. Also, Amazon S3 offers security options such as server-side data encryption and access control lists. This results in a significantly reduced surface area for potential attacks.

Managing Administrator Privileges

Enforcing the principle of *least privilege* is a critical part of a security and governance strategy. In most organizations, the team in charge of DNS configurations is separate from the team that manages web content. You should grant users appropriate levels of permissions to access only the resources they need. In AWS, you can use IAM to lock down permissions.

You can create multiple IAM users under your AWS account, each with their own login and password.[9] An IAM user can be a person, service, or application that requires access to your AWS resources (in this case, Amazon S3 buckets, Amazon CloudFront distributions, and Amazon Route 53 hosted zones) through the AWS Management Console, command line tools, or APIs. You can then organize them into IAM groups based on functional roles. When an IAM user is placed in an IAM group, it inherits the group's permissions.

The fine-grained policies of IAM allow you to grant administrators the minimal privileges needed to accomplish their tasks. The permissions can be scoped to specific Amazon S3 buckets and Amazon Route 53 hosted zones.

The following is an example *separation of duties*:

IAM configuration can be managed by:

- *Super_Admins*

Amazon Route 53 configuration can be managed by:

- *Super_Admins*
- *Network_Admins*

CloudFront configuration can be managed by:

- *Super_Admins*
- *Network_Admins*
- *Website_Admin*

Amazon S3 configuration can be managed by:

- *Super_Admins*
- *Website_Admin*

Amazon S3 content can be updated by:

- *Super_Admins*
- *Website_Admin*
- *Website_Content_Manager*

An IAM user can belong to more than one IAM group. For example, if someone must manage both Amazon Route 53 and Amazon S3, that user can belong to both the *Network_Admins* and the *Website_Admins* groups.

The best practice is to require all IAM users to rotate their IAM passwords periodically. Multi-factor authentication (MFA) should be enabled for any IAM user account with administrator privileges.

Auditing API Calls Made in Your AWS Account

You can use AWS CloudTrail to see an audit trail for API activity in your AWS account. Toggle it on for all AWS Regions, and the audit logs will be deposited to an Amazon S3 bucket. You can use the AWS Management Console to search against API activity history. Or, you can use a third-party log analyzer to analyze and visualize the CloudTrail logs.

You can also build a custom analyzer. Start by configuring CloudTrail to send the data to Amazon CloudWatch Logs. CloudWatch Logs allows you to create automated rules that notify you of suspicious API activity. CloudWatch Logs also has seamless integration with Amazon ES, and you can configure the data to be automatically streamed over to a managed Amazon ES cluster. Once the data is in Amazon ES, users can query against that data directly or visualize the analytics using a Kibana dashboard.

Controlling How Long Amazon S3 Content is Cached by Amazon CloudFront

It is important to control how long your Amazon S3 content is cached at the CloudFront edge locations. This helps make sure that website updates appear correctly. If you're ever confused by a situation in which you've updated your website, but you are still seeing stale content when visiting your CloudFront powered website, one likely reason is that CloudFront is still serving up cached content. You can control CloudFront caching behavior with a combination of Cache-Control HTTP headers, CloudFront Minimum Time-to-Live (TTL) specifications, Maximum TTL specifications, content versioning, and CloudFront Invalidation Requests. Using these correctly will help you manage website updates.

CloudFront will typically serve cached content from an edge location until the content expires. After it expires, the next time that content is requested by an end user, CloudFront goes back to the Amazon S3 origin server to fetch the content and then cache it. CloudFront edge locations automatically expire content after Maximum TTL seconds elapse (by default, this is 24 hours).

However, it could be sooner because CloudFront reserves the flexibility to expire content if it needs to, before the Maximum TTL is reached. By default, the Minimum TTL is set to 0 (zero) seconds, but this value is configurable. Therefore, CloudFront may expire content anytime between the Minimum TTL (default is 0 seconds) and Maximum TTL (default is 24 hours).

For example, if Minimum TTL=60s and Maximum TTL=600s, then content will be cached for *at least* 60 seconds and *at most* 600 seconds.

For example, say you deploy updates to your marketing website, with the latest and greatest product images. After uploading your new images to Amazon S3, you immediately browse to your website DNS, and you still see the old images! It is likely that one and possibly more CloudFront edge locations are still holding onto cached versions of the older images and serving the cached versions up to your website visitors. If you’re the patient type, you can wait for CloudFront to expire the content, but it could take up to Maximum TTL seconds for that to happen. There are several approaches to address this issue, each with its pros and cons.

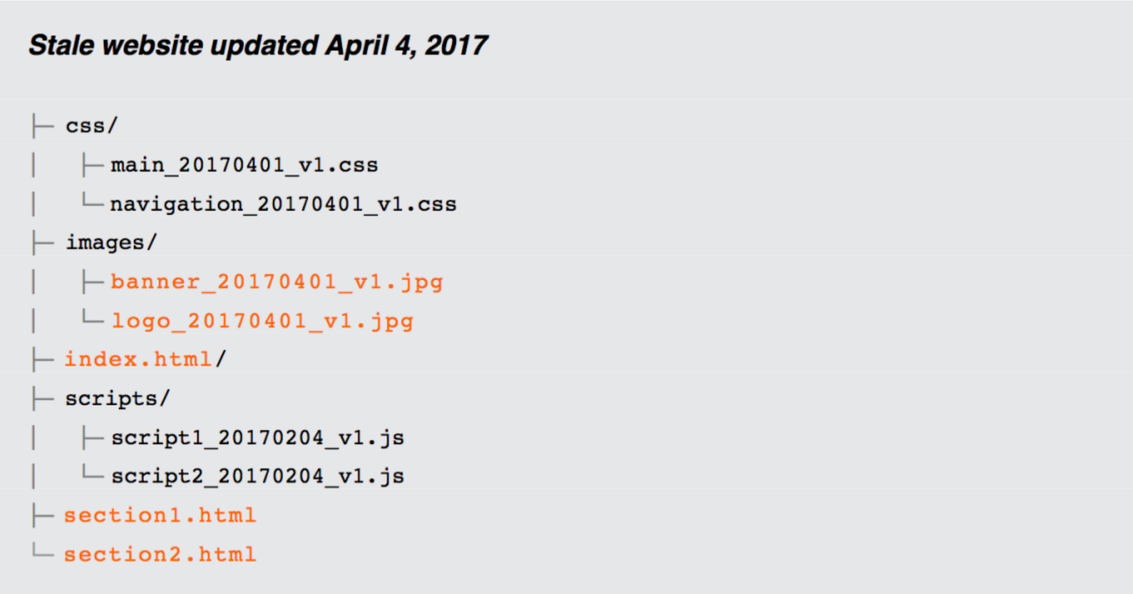
Set Maximum TTL Value

Set the Maximum TTL to be a relatively low value. The tradeoff is that cached content expires faster because of the low Maximum TTL value. This results in more frequent requests to your Amazon S3 bucket because the CloudFront caches need to be repopulated more often. In addition, the Maximum TTL setting applies across the board to all CloudFront files, and for some websites you might want to control cache expiration behaviors based on file types.

Implement Content Versioning

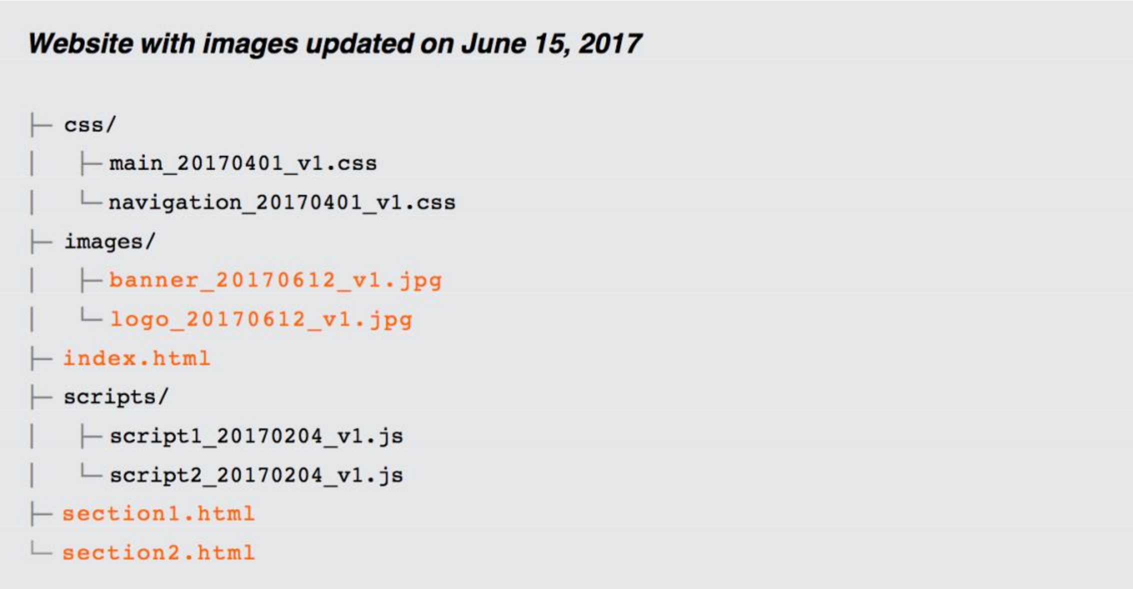
Every time you update website content, embed a version identifier in the file names. It can be a timestamp, a sequential number, or any other way that allows you to distinguish between two versions of the same file. For example, instead of banner.jpg, call it `banner_20170401_v1.jpg`. When you update the image, name the new version `banner_20170612_v1.jpg` and update all files that need to link to the new image.

In the following example, the banner and logo images are updated and receive new file names. However, because those images are referenced by the HTML files, the HTML markup must also be updated to reference the new image file names. Note that the HTML file names shouldn’t have version identifiers in order to provide stable URLs for end users.



Stale website updated April 4, 2017

```
├─ css/
|   ├─ main_20170401_v1.css
|   └─ navigation_20170401_v1.css
├─ images/
|   ├─ banner_20170401_v1.jpg
|   └─ logo_20170401_v1.jpg
├─ index.html/
├─ scripts/
|   ├─ script1_20170204_v1.js
|   └─ script2_20170204_v1.js
├─ section1.html
└─ section2.html
```



Website with images updated on June 15, 2017

```
├─ css/
|   ├─ main_20170401_v1.css
|   └─ navigation_20170401_v1.css
├─ images/
|   ├─ banner_20170612_v1.jpg
|   └─ logo_20170612_v1.jpg
├─ index.html
├─ scripts/
|   ├─ script1_20170204_v1.js
|   └─ script2_20170204_v1.js
├─ section1.html
└─ section2.html
```

Content versioning has a clear benefit: it sidesteps CloudFront expiration behaviors altogether. Since new file names are involved, CloudFront immediately fetches the new files from Amazon S3 (and afterwards, cache them).

Non-HTML website changes are reflected immediately. Additionally, you can roll back and roll forward between different versions of your website.

The main challenge is that content update processes must be “version-aware.” File names must be versioned. Files with references to changed files must also be updated. For example, if an image is updated, the following items must be updated as well:

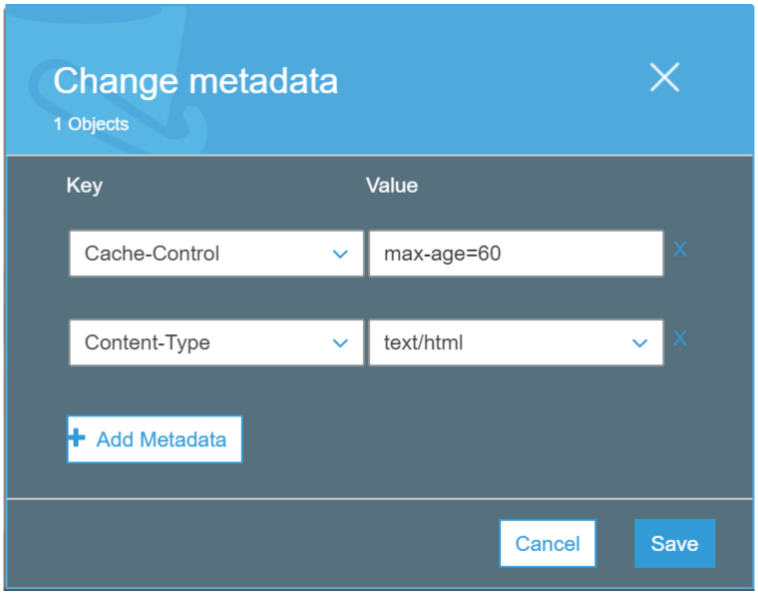
- The image file name
- Content in any HTML, CSS, and JavaScript files referencing the older image file name
- The file names of any referencing files (with the exception of HTML files)

A few static site generator tools can automatically rename file names with version identifiers, but most tools aren’t version-aware. Manually managing version identifiers can be cumbersome and error-prone. If your website would benefit from content versioning, it may be worth investing in a few automation scripts to streamline your content update process.

Specify Cache-Control Headers

You can manage CloudFront expiration behavior by specifying Cache-Control headers for your website content. If you keep the Minimum TTL at the default 0 seconds, then CloudFront honors any `Cache-Control: max-age` HTTP header that is individually set for your content. If an image is configured with a `Cache- Control: max-age=60` header, then CloudFront expires it at the 60 second mark. This gives you more precise control over content expiration for individual files.

You can configure Amazon S3 to return a `Cache-Control HTTP` header with the value of `max-age=<seconds>` when S3 serves up the content. This setting is on a file-by-file basis, and we recommend using different values depending on the file type (HTML, CSS, JavaScript, images, etc.). Since HTML files won’t have version identifiers in their file names, we recommend using smaller `max-age` values for HTML files so that CloudFront will expire the HTML files sooner than other content. You can set this by editing the Amazon S3 object metadata using the AWS Management Console.



Key	Value
Cache-Control	max-age=60
Content-Type	text/html

Figure 11: Setting Cache-Control Values in the console

In practice, you should automate this as part of your Amazon S3 upload process. With AWS CLI, you can alter your deployment scripts like the following example:

```
aws s3 sync /path s3://yourbucket/ --delete --recursive \
                    --cache-control max-age=60
```

Use CloudFront Invalidation Requests

CloudFront invalidation requests are a way to force CloudFront to expire content. Invalidation requests aren't immediate. It takes several minutes from the time you submit one to the time that CloudFront actually expires the content. For the occasional requests, you can submit them using the AWS Management Console. Otherwise, use the AWS CLI or AWS APIs to script the invalidation. In addition, CloudFront lets you specify which content should be invalidated: You can choose to invalidate your entire Amazon S3 bucket, individual files, or just those matching a wildcard pattern. For example, to invalidate only the images directory, issue an invalidation request for:

/images/*.

In summary, the best practice is to understand and use the four approaches together. If possible, implement content versioning. It allows you to immediately review changes and gives you the most precise control over the CloudFront and Amazon S3 experience. Set the Minimum TTL to be 0 seconds and the Maximum TTL to be a relatively low value. Also, use `Cache-Control` headers for individual pieces of content. If your website is infrequently updated, then set a large value for `Cache-Control:max-age=<seconds>` and then issue CloudFront invalidation requests every time your site is updated. If the website is updated more frequently, use a relatively small value for `Cache-Control:max-age=<seconds>` and then issue CloudFront invalidation requests only if the `Cache-Control:max-age=<seconds>` settings exceeds several minutes.

Conclusion

This whitepaper began with a look at traditional (non-AWS) architectures for static websites. We then showed you an AWS Cloud-native architecture based on Amazon S3, Amazon CloudFront, and Amazon Route 53.

The AWS architecture is highly available and scalable, secure, and provides for a responsive user experience at very low cost. By enabling and analyzing the available logs, you can you understand your visitors and how well the website is performing. Fewer moving parts means less maintenance is required. In addition, the architecture costs only a few dollars a month to run.

Contributors

Contributors to this document include:

- Jim Tran, AWS Principal Enterprise Solutions Architect
- Bhushan Nene, Senior Manager, AWS Solutions Architecture
- Jonathan Pan, Senior Product Marketing Manager, AWS
- Brent Nash, Senior Software Development Engineer, AWS

Further Reading

For additional information, see:

- AWS Whitepapers page
- Amazon CloudFront Developer Guide
- Amazon S3 Documentation

Document Revisions

Date	Description
February 2019	Added usage of HTTPS
May 2017	First publication

Notes

[1] Each S3 object can be zero bytes to 5 TB in file size, and there's no limit to the number of Amazon S3 objects you can store.

[2] https://letsencrypt.org/stats/

[3] If your high-availability requirements require that your website must remain available even in the case of a failure of an entire AWS Region, explore the Amazon S3 Cross-Region Replication capability to automatically replicate your S3 data to another S3 bucket in a second AWS Region.

[4] For Microsoft IIS web servers, this is equivalent to "default.html"; for Apache web servers, this is equivalent to "index.html"

[5] For moving very large quantities of data into S3, see https://aws.amazon.com/s3/cloud-data-migration/

[6] "We find that the performance penalty incurred by a web flow due to its TCP handshake is between 10% and 30% of the latency to serve the HTTP request, as we show in detail in Section 2" from https://raghavan.usc.edu/papers/tfo-conext11.pdf

[7] Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on the prices effective at the time of this writing. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

[8] If version control software is not in use at your organization, one alternative approach is to look at the Amazon S3 object versioning feature for versioning your critical files. Note that object versioning introduces storage costs for each version, and requires you to programmatically manage the different versions. For more information, see http://docs.aws.amazon.com/AmazonS3/latest/dev/Versioning.html.

[9] The AWS account is the account that you create when you first sign up for AWS. Each AWS account has root permissions to all AWS resources and services. The best practice is to enable multi-factor authentication (MFA) for your root account, and then

lock away the root credentials so that no person or system uses the root credentials directly for day-to-day operations. Instead, create IAM groups and IAM users for the day-to-day operations.