

Lernetto Blog (/blog)

Get our top 3 programming & design tutorials and expert interviews delivered to you every week

Learn for free

Subscribers also get exclusive discounts on our premium courses.

Tutorial: How to use Amazon S3 and CloudFront CDN to serve images fast and cheap



(/users/samirtalwar) **Samir Talwar** (/users/samirtalwar) 03 May 2016

*This is a guest post by **Samir Talwar** (<https://noodlesandwich.com/>), a software developer based in London. Samir cares deeply about software quality and craftsmanship. You can read more of his writing on his **excellent blog** (<http://monospacedmonologues.com/>) and follow him on **Twitter** (<https://twitter.com/SamirTalwar>).*

In my last tutorial, we figured out **how to host static web pages for free on GitHub pages** (<https://lernetto.com/blog/tutorial-how-to-host-your-websites-for-free-using-github-pages>). However, if you tried to host images or other large assets in the same fashion, you may have noticed that it's pretty slow. GitHub Pages was never designed to handle large files. In this article, we'll explore a much faster, yet extremely cost-effective solution for dealing with non-text data.

My blog, **monospacedmonologues.com** (<http://monospacedmonologues.com/>), is pretty text-heavy, but gets a sprinkling of pictures once in a while. The blog itself is really just a **Tumblr** (<https://www.tumblr.com/>) blog, and so I don't host it anywhere I can store files. But that doesn't matter, because the images can be hosted anywhere.

Monospaced Monologues



Language-Agnostic Test Cases

When pairing with [@sleepyfox](#) on a kata, we decided to write the code in a shell script. Someone snarkily asked how we were going to test-drive our solution. So, after a second of thought, I remembered my test framework, [Smoke](#).

Smoke is a little different from other test frameworks. It was designed to test code written in any language, so you don't write the tests in code. You simply specify the input to be provided to the program via command-line arguments and STDIN, and the expected STDOUT, STDERR and exit statuses. To do this, you just create five text files (though you can leave some out) with the `.args`, `.in`, `.out`, `.err` and `.status` file extensions.

One advantage of this is that it constrains you to test the command-line interface of your program. While not helpful for lower-level testing, it really forces you to think about the output of your command-line application and how it should behave in various edge cases.

Another interesting feature is that if you switch programming languages, your tests can stay the same. We switched languages twice in an hour, from Bash to awk to Python. During the rewrites, our tests stayed exactly the same.

Even the pictures on my blog are code-heavy (<https://learnetto-blog.s3.amazonaws.com/blog/2016-05-04/1462388093679-blog.png>)

Personally, I use **Amazon Web Services** (<https://aws.amazon.com/>) to host my images, but there are lots of other storage providers, such as **Rackspace Cloud** (<https://www.rackspace.com/cloud>), **Google Cloud** (<https://cloud.google.com/>) and **Microsoft Azure** (<https://azure.microsoft.com/>). The important thing is to pick one that isn't going away any time soon, so I'd suggest sticking to the big players.

If you want to follow along, you'll first need an **Amazon Web Services** (<https://aws.amazon.com/>) account. The process is pretty similar for all of the above cloud providers though.

Buckets of assets

Amazon, along with a few of the other cloud providers, use the term "bucket" to refer to a lot of related files. You can pretty much consider it a folder that belongs to you.

Open up the S3 service.

The screenshot shows the AWS console with various service categories. A blue arrow points from the 'Storage & Content Delivery' section to the 'S3' service.

Amazon Web Services

Compute

- EC2: Virtual Servers in the Cloud
- EC2 Container Service: Run and Manage Docker Containers
- Elastic Beanstalk: Run and Manage Web Apps
- Lambda: Run Code in Response to Events

Storage & Content Delivery

- S3: Scalable Storage in the Cloud
- CloudFront: Global Content Delivery Network
- Elastic File System **PREVIEW**: Fully Managed File System for EC2
- Glacier: Archive Storage in the Cloud
- Snowball: Large Scale Data Transport
- Storage Gateway: Hybrid Storage Integration

Database

- RDS: Managed Relational Database Service
- DynamoDB: Managed NoSQL Database
- ElastiCache: In-Memory Cache
- Redshift: Fast, Simple, Cost-Effective Data Warehousing
- DMS: Managed Database Migration Service

Networking

- VPC: Isolated Cloud Resources
- Direct Connect: Dedicated Network Connection to AWS
- Route 53: Scalable DNS and Domain Name Registration

Developer Tools

- CodeCommit: Store Code in Private Git Repositories
- CodeDeploy: Automate Code Deployments
- CodePipeline: Release Software using Continuous Delivery

Management Tools

- CloudWatch: Monitor Resources and Applications
- CloudFormation: Create and Manage Resources with Templates
- CloudTrail: Track User Activity and API Usage
- Config: Track Resource Inventory and Changes
- OpsWorks: Automate Operations with Chef
- Service Catalog: Create and Use Standardized Products
- Trusted Advisor: Optimize Performance and Security

Security & Identity

- Identity & Access Management: Manage User Access and Encryption Keys
- Directory Service: Host and Manage Active Directory
- Inspector: Analyze Application Security
- WAF: Filter Malicious Web Traffic
- Certificate Manager: Provision, Manage, and Deploy SSL/TLS Certificates

Analytics

- EMR: Managed Hadoop Framework
- Data Pipeline: Orchestration for Data-Driven Workflows
- Elasticsearch Service

Internet of Things

- AWS IoT: Connect Devices to the Cloud

Game Development

- GameLift: Deploy and Scale Session-based Multiplayer Games

Mobile Services

- Mobile Hub: Build, Test, and Monitor Mobile Apps
- Cognito: User Identity and App Data Synchronization
- Device Farm: Test Android, iOS, and Web Apps on Real Devices in the Cloud
- Mobile Analytics: Collect, View and Export App Analytics
- SNS: Push Notification Service

Application Services

- API Gateway: Build, Deploy and Manage APIs
- AppStream: Low Latency Application Streaming
- CloudSearch: Managed Search Service
- Elastic Transcoder: Easy-to-Use Scalable Media Transcoding
- SES: Email Sending and Receiving Service
- SQS: Message Queue Service
- SWF: Workflow Service for Coordinating Application Components

Enterprise Applications

- WorkSpaces: Desktops in the Cloud
- WorkDocs: Secure Enterprise Storage and Sharing Service

Resource Groups [Learn more](#)

A resource group is a collection of resources that share one or more tags. Create a group for each project, application, or environment in your account.

[Create a Group](#) [Tag Editor](#)

Additional Resources

[Getting Started](#)

Read our [documentation](#) or view our [training](#) to learn more about AWS.

[AWS Console Mobile App](#)

View your resources on the go with our AWS Console mobile app, available from [Amazon Appstore](#), [Google Play](#), or [iTunes](#).

[AWS Marketplace](#)

Find and buy software, launch with 1-Click and pay by the hour.

[AWS re:Invent Announcements](#)

Explore the next generation of AWS cloud capabilities. [See what's new](#)

Service Health

All services operating normally.

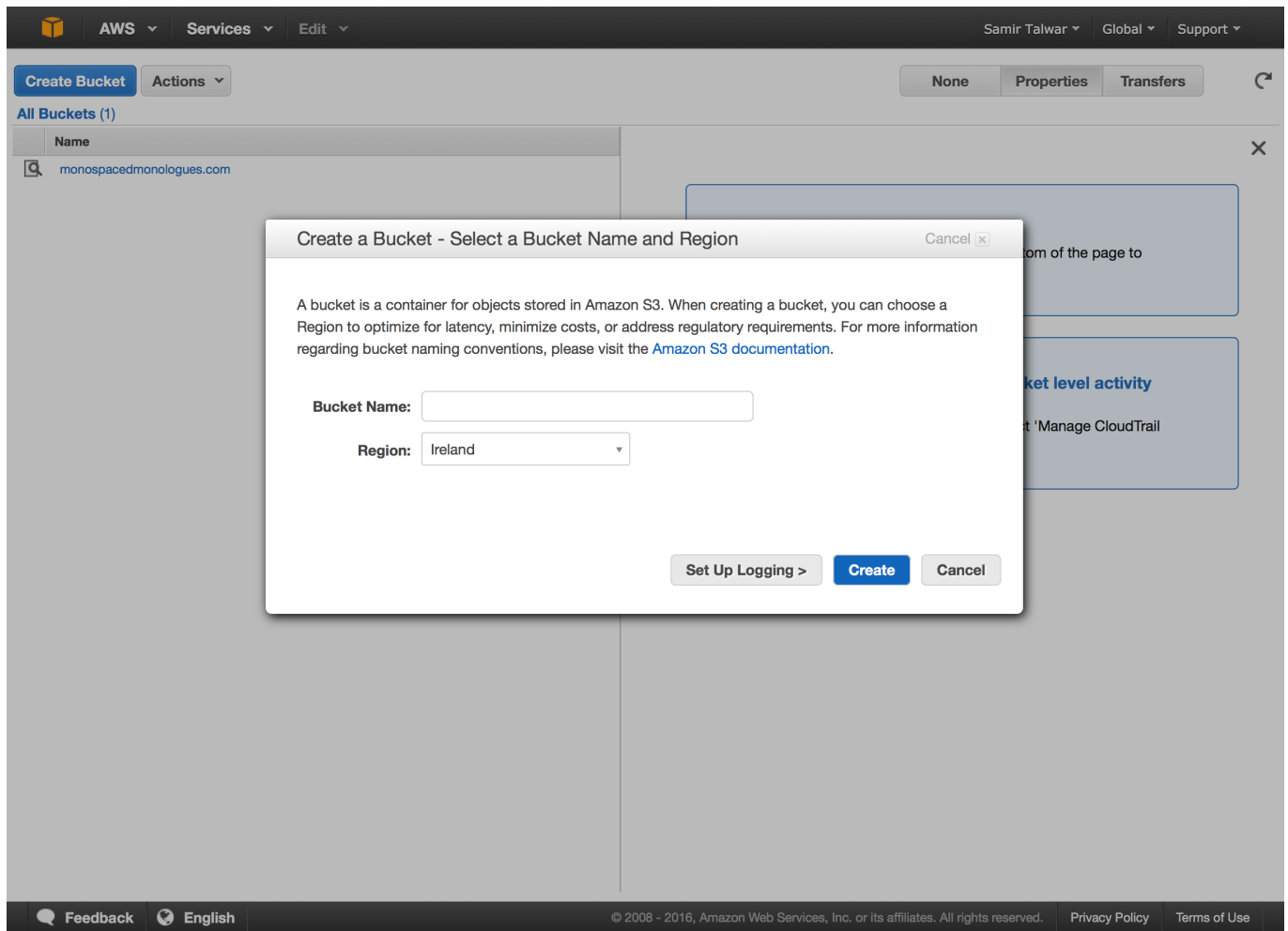
Updated: May 02 2016 20:58:00 GMT+0100

[Service Health Dashboard](#)

The list of services can be a little daunting (<https://learnetto-blog.s3.amazonaws.com/blog/2016-05-04/1462388307514-opening%20S3.png>)

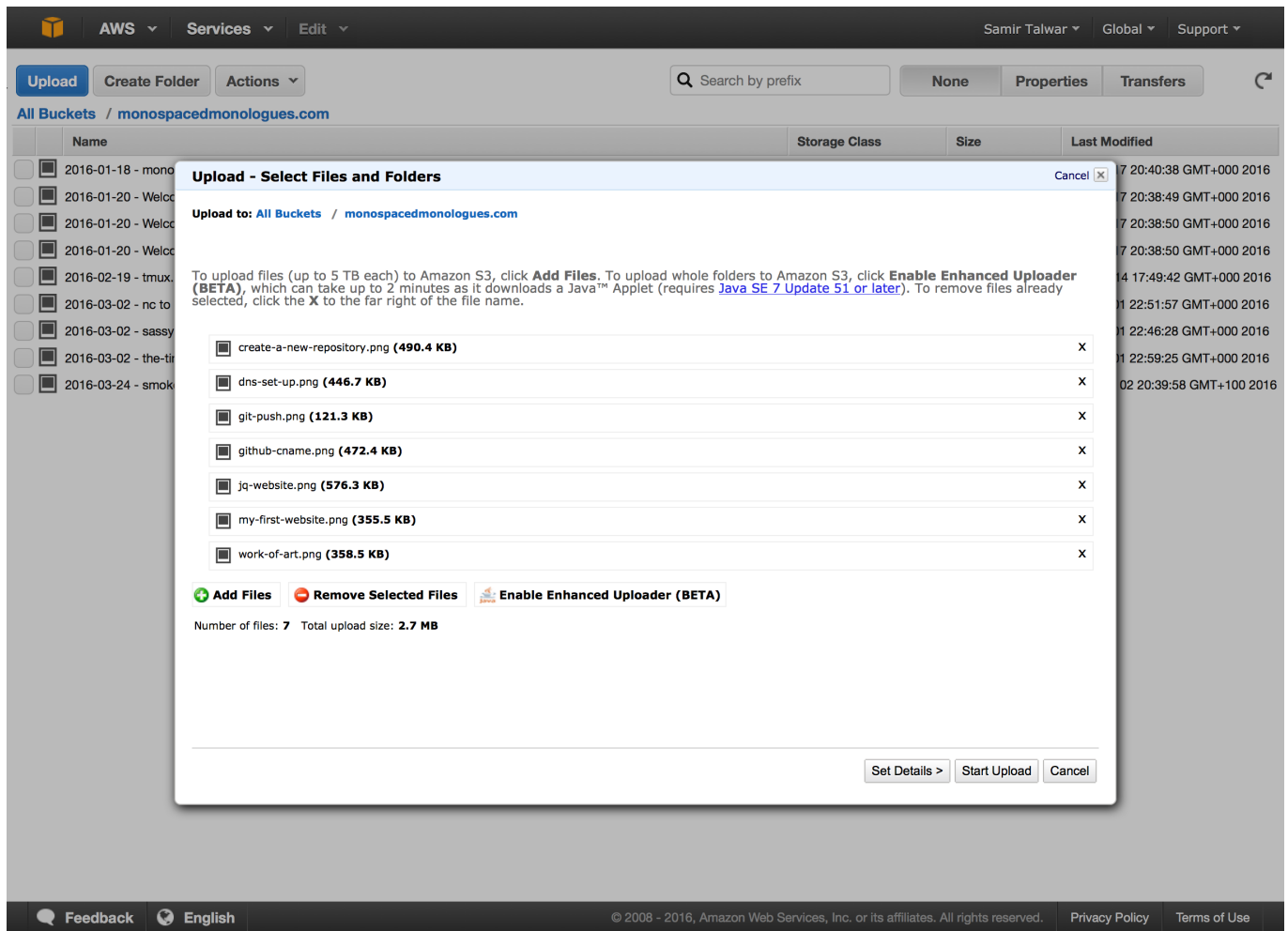
Once we're on the S3 home page, create your bucket. If you have a domain name already, I'd recommend naming it the same as your domain — it'll stop you getting confused later if you end up with multiple buckets for multiple purposes. I named mine "monospacedmonologues.com". Your bucket name needs to be globally unique, so if you don't own a domain name, you may have to be fairly inventive.

As for the location, I picked the one closest to me. If your friends/readers/customers are all in one spot, then you could pick the location closest to them. Don't fret about it too much though. We're going to distribute our files all over the world soon.



Create an S3 bucket (<https://learnetto-blog.s3.amazonaws.com/blog/2016-05-04/1462389182096-create%20an%20S3%20bucket.png>)

Once you have a bucket, you can upload files. Click the big blue *Upload* button, then *Add Files*, and select the files you want to upload. You can rename them later in the S3 interface if you need.



Upload files to S3 (<https://learnnetto-blog.s3.amazonaws.com/blog/2016-05-04/1462389500497-upload%20files%20to%20S3.png>)

Once they're up on Amazon's servers, select one of your newly-uploaded files and click *Properties* in the top-right. You'll see a link to the file. The structure looks something like this:

```
https://s3-<location>.amazonaws.com/<bucket>/<file>
```

You can also access your assets through a nicer-looking link that has this structure:

```
https://<bucket>.s3.amazonaws.com/<file>
```

They're basically the same thing, but having a domain name specific to your bucket has a bunch of advantages we'll see later.

For now, open it up using whichever URL you like. Either way, you'll see an "Access Denied" message. This is because S3 files are private by default, as many people use it to store sensitive data. In order to host your website files there, you'll need to change the permissions of the files so they can be accessed by the outside world.

To do so, first select a file and click *Properties* on the top right. then open the *Permissions* section and add an item. We're going to grant "Everyone" the right to *Open/Download* the file.

The screenshot shows the AWS Management Console interface for an S3 bucket named 'monospacedmonologues.com'. The left sidebar lists various objects, with '2016-03-24 - smoke.png' selected. The main panel displays the details for this object, including its bucket, name, link, size, last modified date, owner, ETag, expiry date, and expiration rule. Below the details, there are sections for 'Permissions' and 'Metadata'. The 'Permissions' section shows that the object is accessible to 'samir' with 'Open/Download' and 'View Permissions' permissions. The 'Metadata' section is currently empty. At the bottom of the console, there is a promotional banner that reads 'Like what you're reading?' and encourages users to get new programming & design tutorials, interviews with experts, and exclusive discounts on courses.

Of course, this could get tedious if we have more than a few files. An alternative is to specify a *policy* for the bucket. These are quite complicated, but we can do it by using the **AWS Policy Generator** (<https://awspolicygen.s3.amazonaws.com/>).

First of all, select "S3 Bucket Policy" as the type of policy. It'll then prompt you for a few pieces of information.

- The *Principal* is the user who will be accessing the object. As we want everyone to access it, enter `*`.
- As for *Actions*, we would like everyone to be able to execute the `GetObject` action and nothing else.
- Just like the example below, the *Amazon Resource Name* should be something like:

```
arn:aws:s3:::<bucket_name>/*
```

Our key name is `*` because we want people to access everything in this bucket.



Like what you're reading?

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an [IAM Policy](#), an [S3 Bucket Policy](#), an [SNS Topic Policy](#) and an [SQS Queue Policy](#).

Select Type of Policy S3 Bucket Policy

Like what you're reading?

A statement is the formal description of a single permission. See [a description of elements](#) that you can use in statements.

Effect

☒ Allow ☐ Deny

Principal

Use a comma to separate multiple values.

AWS Service

Amazon S3

⌵

☐ All Services ('*')

Use multiple statements to add permissions for more than one service.

Actions

1 Action(s) Selected

⌵

☐ All Actions ('*')

Amazon Resource Name (ARN)

ARN should follow the following format: arn:aws:s3:::<bucket_name>/<key_name>
Use a comma to separate multiple values.

Add Conditions (Optional)

Add Statement

Like

You added the following statements. Click the button below to Generate a policy.

Get new programming & design tutorials, interviews with experts and exclusive discounts on courses.

Generate a policy - part 1 (<https://learnnetto-blog.s3.amazonaws.com/blog/2016-05-04/1462389240354>)

Email Address

Get free stuff

POWERED BY DRIP ([HTTP://MBSY.CO/GJTFJ?](http://MBSY.CO/GJTFJ?)

Once you're done, click *Add Statement*, then *Generate Policy*. You'll end up with a policy that looks something like this:

th a policy that looks something like this:

```
{
  "Id": "Policy1462221401547",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1462221025865",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::monospacedmonologues.com/*",
      "Principal": "*"
    }
  ]
}
```

Copy the code you see in the policy generator and Head back to S3. Deselect all files and open the *Properties* view to check out the bucket's properties. Open up the *Permissions* section, click *Add bucket policy* and paste in the policy you generated, then save.

The screenshot shows the AWS Management Console interface. At the top, there's a navigation bar with 'AWS', 'Services', and 'Edit' buttons. Below that, a search bar and tabs for 'None', 'Properties', and 'Transfers' are visible. The main content area displays a list of buckets under the 'monospacedmonologues.com' prefix. A 'Bucket Policy Editor' modal is open, showing a policy for the bucket 'monospacedmonologues.com'. The policy is a JSON document that grants 's3:GetObject' permission to all users. The modal also includes a 'Save' button and a 'Close' button. In the background, there's a sidebar with navigation options like 'Lifecycle', 'Cross-Region Replication', 'Tags', 'Requester Pays', and 'Transfer Acceleration'. A 'Like what you're reading?' banner is also present.

Generate a policy - part 2 (<https://learnetto-blog.s3.amazonaws.com/blog/2016-05-04/1462389268360>)

Finally, click on that link again to view your file. It should be rendering nicely now! You can use those links everywhere. Instead of hosting your assets with your code where they take up valuable bandwidth, let someone else do the heavy lifting.

CDNs, because files are big and not your problem

S3 is a great place to put your files, but a bucket still lives in one place. This means that transferring your assets to someone on the other side of the world will still be slow.

A *Content Delivery Network*, or *CDN*, solves this problem by storing copies of your files all over the planet in lots of data centres. This means that your files are physically close to your customers no matter where they are, increasing transfer speed and improving their experience. We can set up a CDN to copy the contents of our S3 bucket everywhere we need.

Amazon's CDN service is called *CloudFront*. Open up the CloudFront home page and click *Create Distribution*, then select the *Web* distribution. In the *Origin Domain Name*, enter your bucket's domain name in the form `<bucket>.s3.amazonaws.com` (it should auto-complete). Everything else will be filled in for you. Scroll to the bottom and click *Create Distribution*.

Minimum TTL