

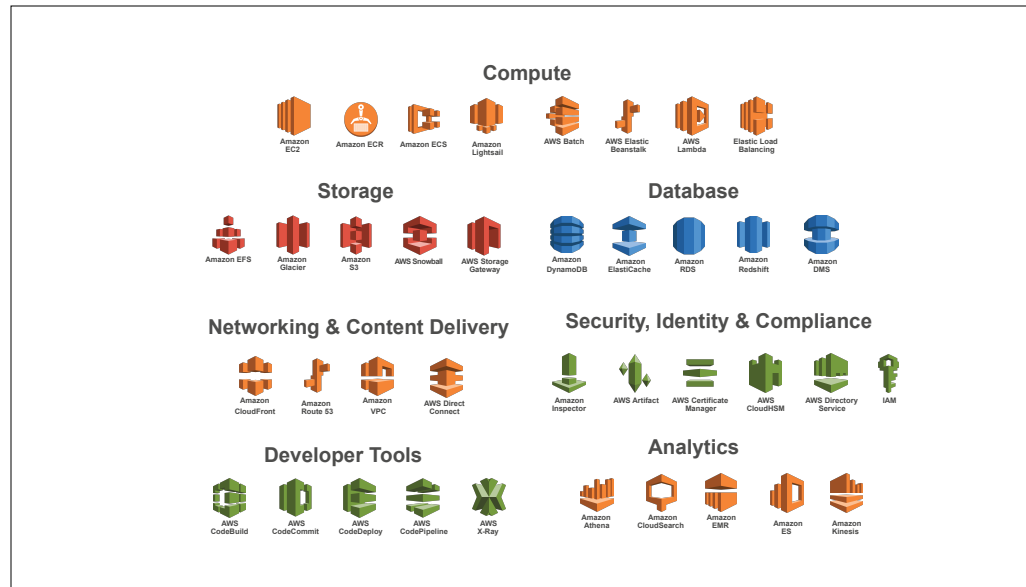
Getting Started With Amazon Web Services

Rich Trouton

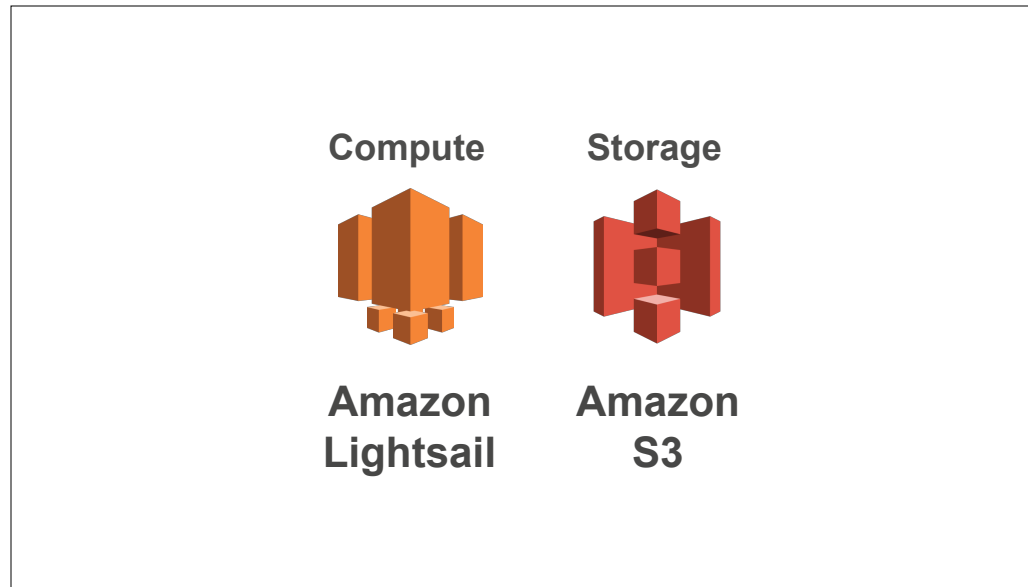
Apple CoE @ 

Before we get started, there's two things I'd like to mention. The first is that, all of the slides, speakers' notes and the demos are available for download and I'll be providing a link at the end of the talk. I tend to be one of those folks who can't keep up with the speaker and take notes at the same time, so for those folks in the same situation, no need to take notes. Everything I'm covering is going to be available for download.

The second is to please hold all questions until the end. If you've got questions, make a note of them and hit me at the end of the talk. With luck, I'll be able to answer most of your questions during the talk itself.



I want to start with doing some expectations management. Amazon has many services available and I'm not going to be talking about a lot of them.



Instead, we're going to focus on two Amazon services which are easy to get started with: Lightsail and S3. To simplify them, S3 is a file storage service and Lightsail is a virtual machine hosting service for Windows and Linux.

Security, Identity & Compliance



Identity & Access Management

However, before we get into those, we need to discuss another service: Identity and Access Management; otherwise known as IAM. The reason is that setting things up properly in IAM is fundamental before you'll be able to work securely with Amazon Web Services.

Security, Identity & Compliance

Three Rules of Thumb

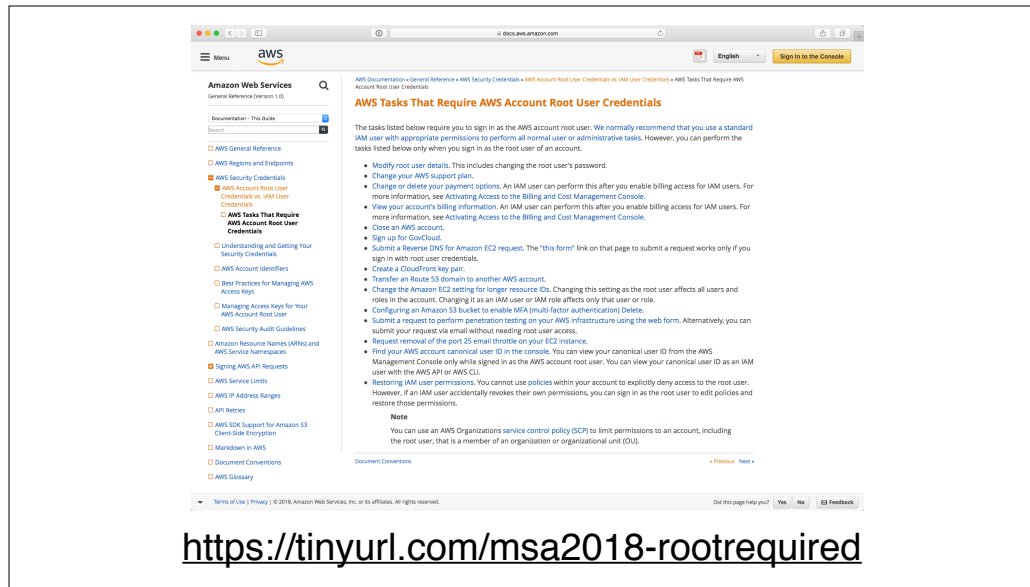
In working with IAM, I've picked up some best practices.

Security, Identity & Compliance

1. Don't log in as root*

***Unless you really need to.**

Just like on Unix-based OSs, you have the choice of logging into your AWS account as root or as a non-root user. Just like with a Unix-based OS, you can get into a lot of trouble if you run as root all the time.



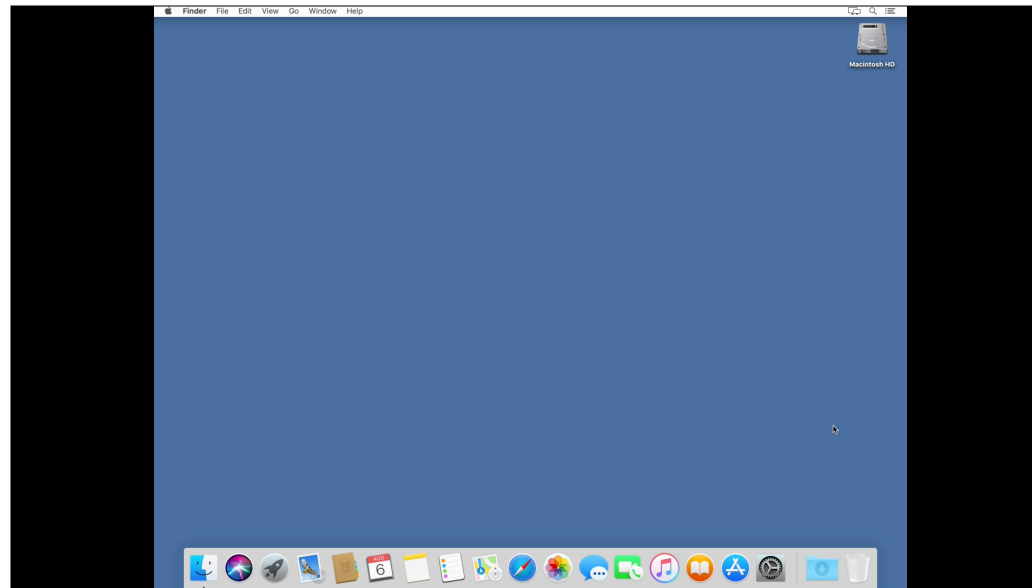
<https://tinyurl.com/msa2018-rootrequired>

That said, there are certain tasks that require root login. AWS provides documentation of when those circumstances apply.

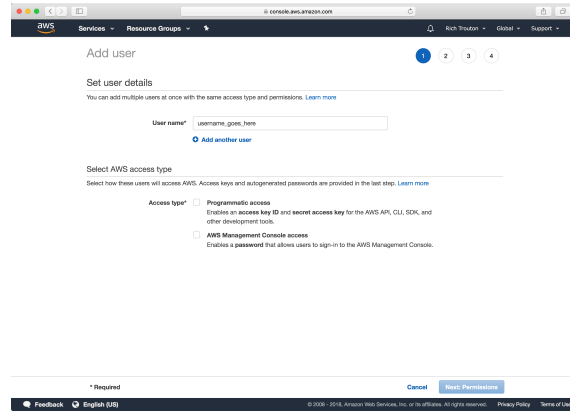
Security, Identity & Compliance

- A. Set a complex password for your root account.
- B. Enable multi-factor authentication (MFA) for your root account.
- C. Only log into your root account if there is no other option available

So what do you do otherwise with your root account? Protect it by setting a complex password and enabling multi-factor authentication, then log into it as seldom as possible.



Creating IAM users



The screenshot shows the AWS IAM console 'Add user' page. The page has a dark header with the AWS logo and navigation links. Below the header, there's a progress bar with four steps: 1 (selected), 2, 3, and 4. The main section is titled 'Set user details' and includes a text input for 'User name' with the value 'usermeme_gives_here'. Below this is a link 'Add another user'. The 'Select AWS access type' section has two radio buttons: 'Programmatic access' (selected) and 'AWS Management Console access'. The footer contains a 'Cancel' button, a 'Next: Permissions' button, and a footer bar with 'Feedback', 'English (GB)', and copyright information.

<https://tinyurl.com/msa2018-createiamuser>

Instead, create IAM users and use those to log into your AWS account. The advantage with using IAM user accounts is that you can make them as privileged or unprivileged as you need to.

IAM Account Types

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type***
- ☐ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
 - ☐ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

There's two general IAM account types, programmatic access and AWS management console.

IAM Account Types

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type*

☐

Programmatic access

Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

☐

AWS Management Console access

Enables a **password** that allows users to sign-in to the AWS Management Console.

The first is programmatic access. These accounts have a username, but don't have a password because they're not meant to log into the AWS web console. Instead, they get an access key ID and secret access key. These keys are used for authentication for the AWS API and other AWS command line and development tools.

IAM Account Types

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

- Access type* ☐ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- ☒ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

The next kind are AWS Management console access accounts. These accounts have a username and a password because they can log into the AWS web console. These accounts can also get an access key ID and secret access key, but that leads to my second rule.

Security, Identity & Compliance

**2. Do not give access
keys to accounts you
use to log into the web
console.**

In general I would not recommend adding keys to an AWS management console account.

IAM Account Types

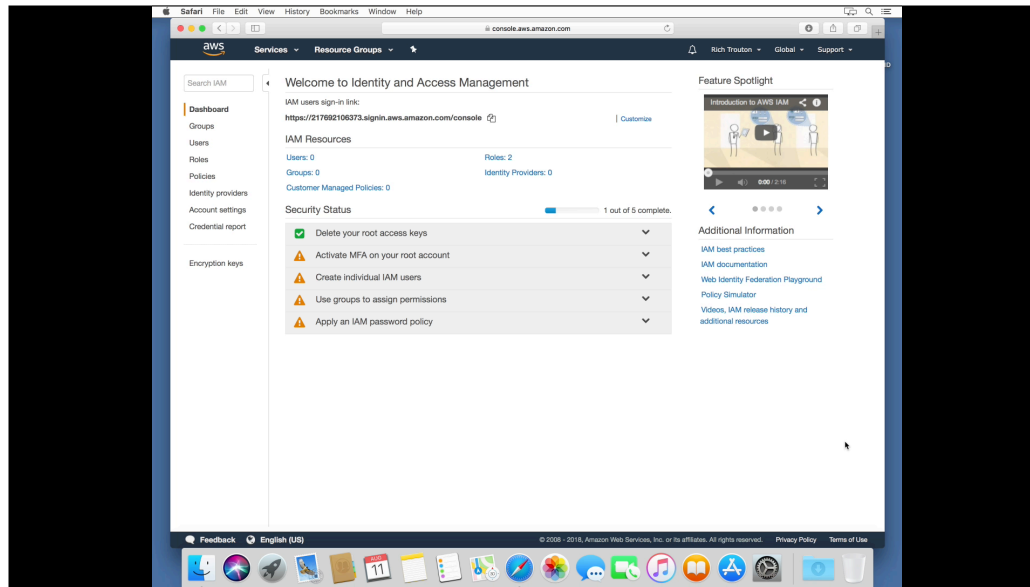
- **AWS Console accounts**
 - User accounts
- **Programmatic accounts**
 - Service accounts

The reason I don't recommend adding access keys is that, in my opinion, you should be treating access key-enabled accounts like you would a service account and give them only the permissions they need to do a specific job.

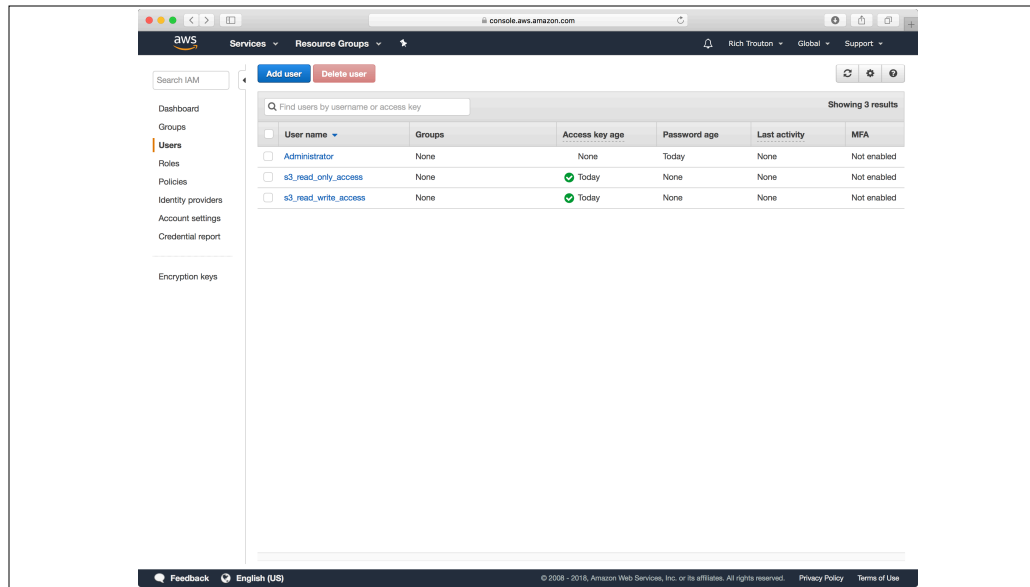
Meanwhile, treat your AWS Console accounts like user accounts which only have rights in the admin console and don't have rights for the AWS API or other command line tools.

In the event that you find that you need to do work with AWS API or other AWS developer tools, set up a separate programmatic account and assign it just the rights which are needed.

Having account separation like this may help keep you out of trouble. For example, you may need administrator permissions in the AWS web console but on the command line you may only need something like read only access to S3. By having two separate accounts, one for console access and one for API work, you can easily accommodate both needs while still following the principle of only having the privileges you need to get your work done.



1. Create administrator management console account
2. Create s3_read_write_access programmatic access account
3. Create s3_read_only_access programmatic access account

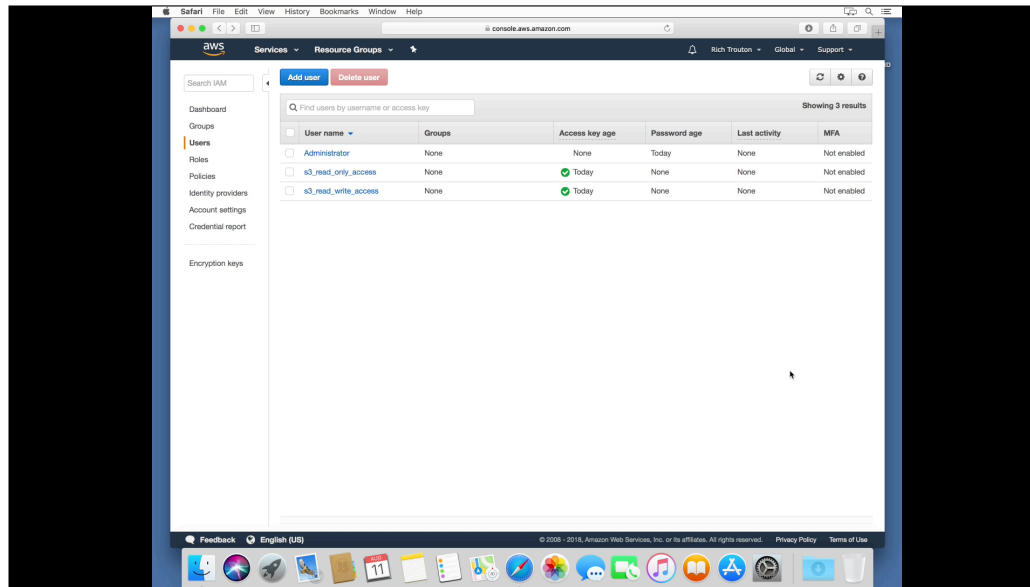


So now I've got three accounts created, but I skipped right past giving them any permissions. They're powerless. Why?

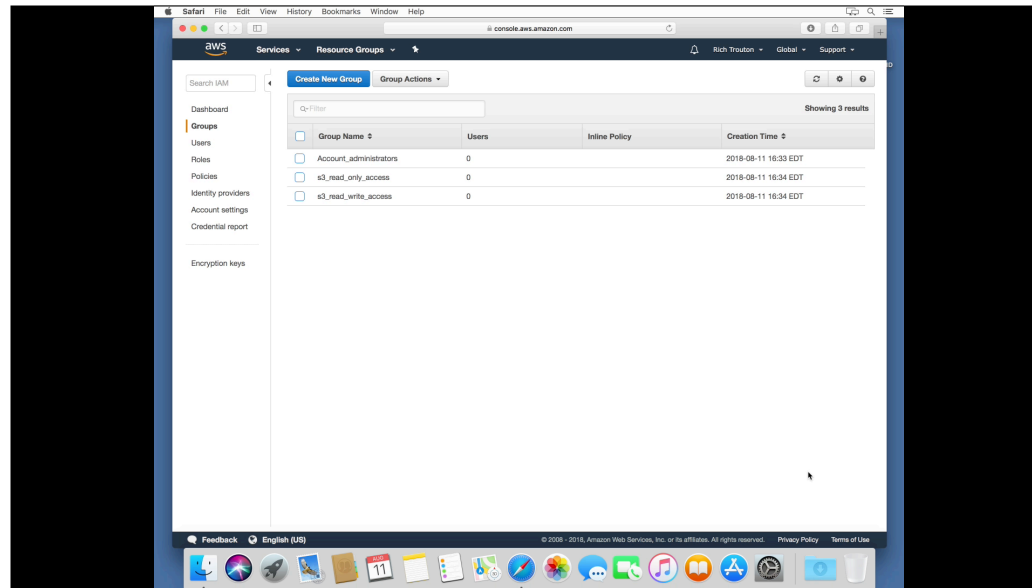
Security, Identity & Compliance

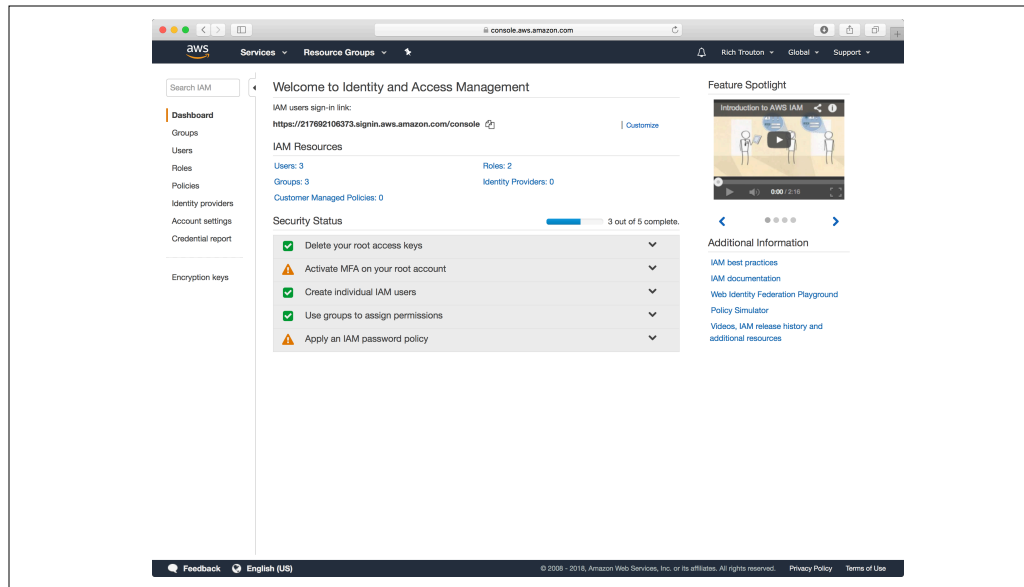
3. Use groups to assign permissions

Last rule, use groups to assign your IAM permissions. The reason why is straightforward: It makes it very simple to manage permissions. Want your new hire to have read-write access to S3? Easy, create their account, add them to a group with the necessary permissions and their account will inherit those permissions. New hire went power mad and trashed the CFO's files? Pull the new hire's account from the group and now they've got just enough permissions to change their password and nothing else.

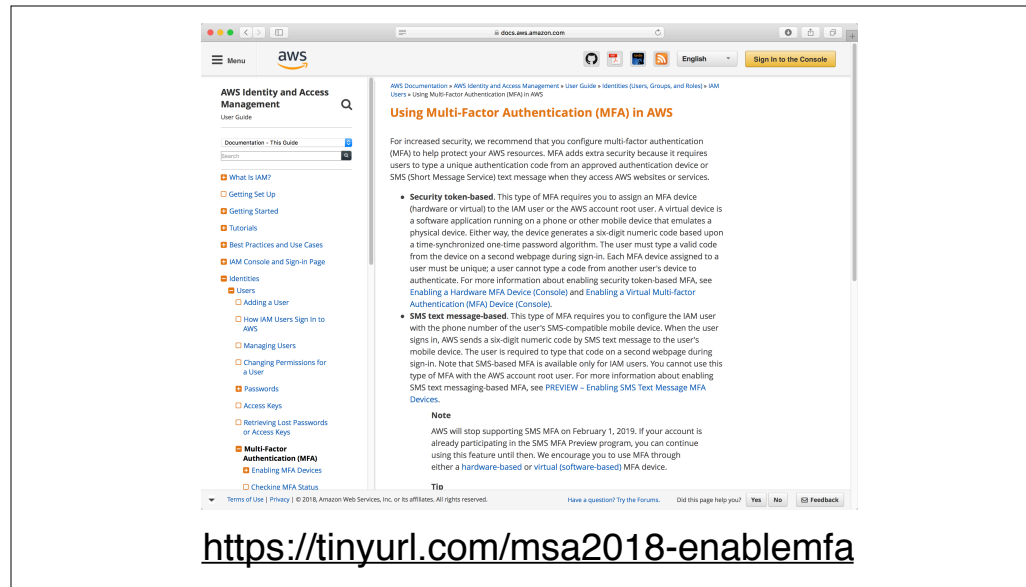


1. Create Account_administrators group
2. Create s3_read_only_access group
3. Create s3_read_write_access group

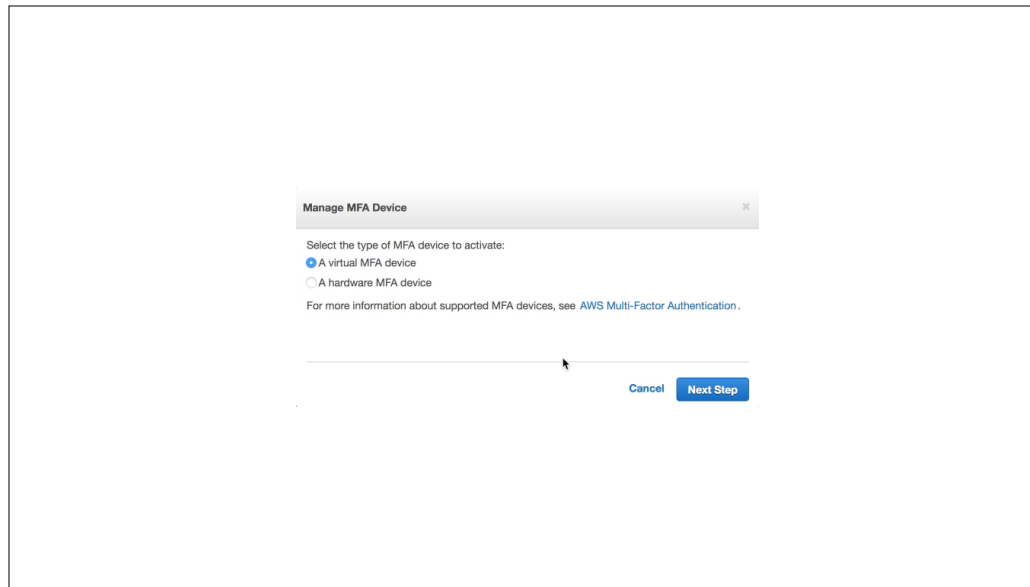




Did I come up with these three rules of thumb on my own? Nope, Amazon's pretty up front with most of them. As you can see, I still need to apply a couple more recommended security settings to my account.



Another good idea is to enable multi-factor authentication for your console accounts, which I'll be referring to as MFA elsewhere in this talk. This will help a lot with securing your accounts, especially those accounts with a lot of administrative rights.

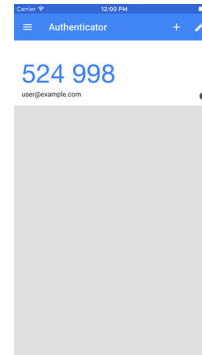


As part of setting up MFA, you'll need to decide to use either a hardware MFA device or a virtual MFA device.

MFA device types



Hardware MFA device



Virtual MFA device

A hardware MFA device usually is a physical device that displays a series of codes. A virtual MFA device does the same, but displays the codes in an app instead.



**Google
Authenticator**

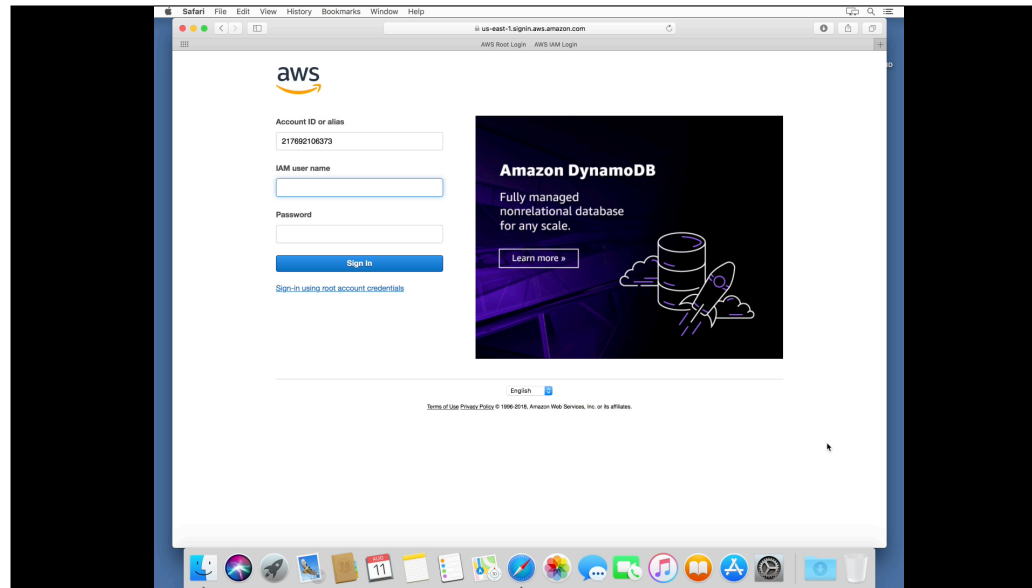


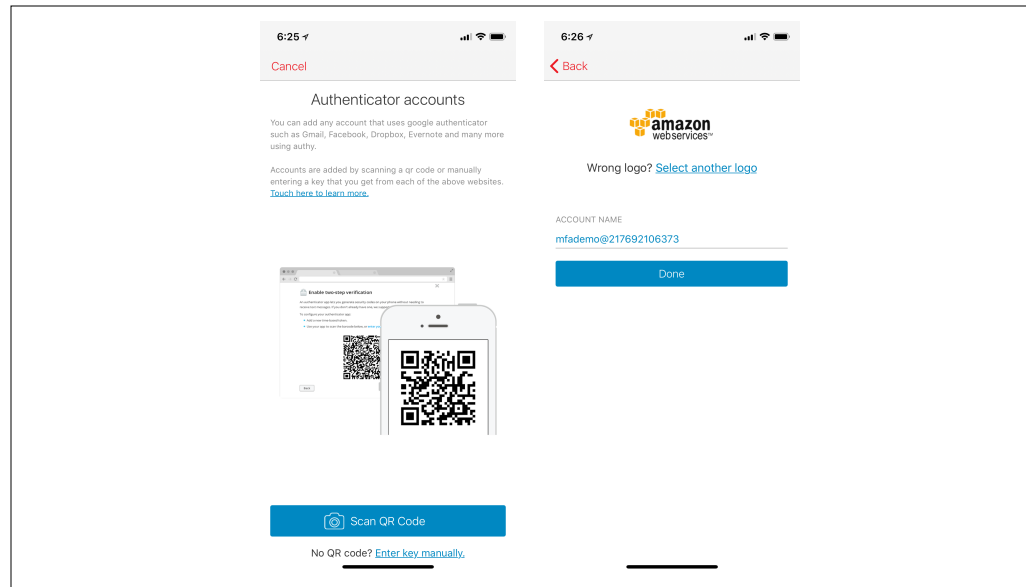
Authy

<https://authy.com/download/>

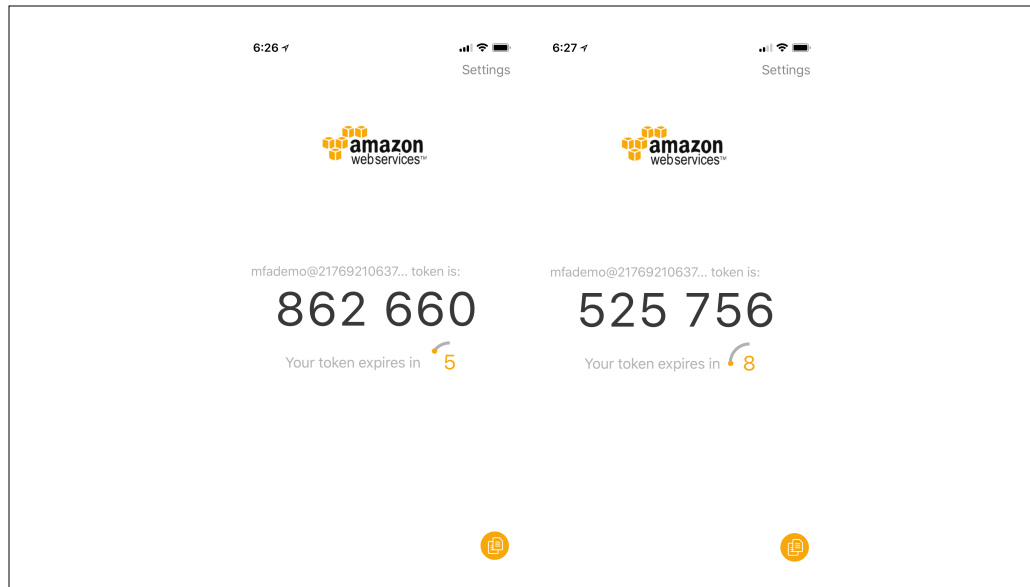
<https://support.google.com/accounts/answer/1066447>

Two iOS apps which can be used with Amazon's MFA are Google Authenticator and Authy. I prefer Authy, so I'll be setting up MFA using that.

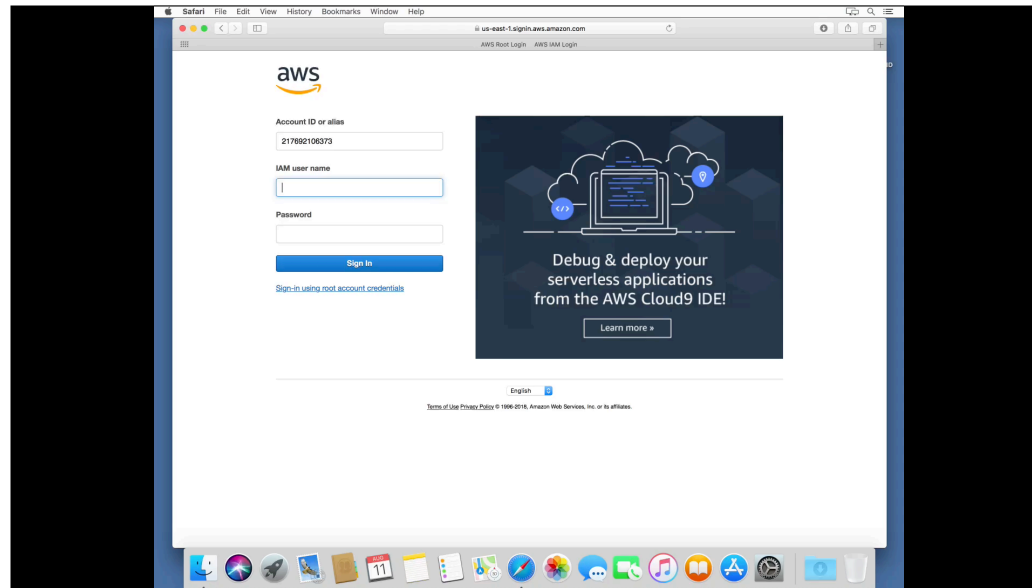


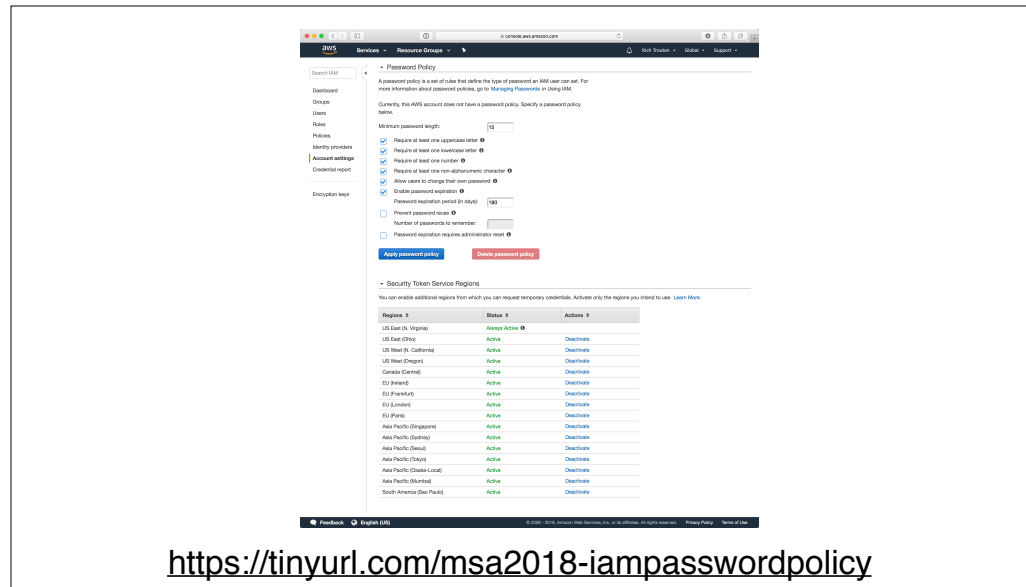


Meanwhile, in Authy, this is what it looks like when I scan the QR code and my AWS account is registered.



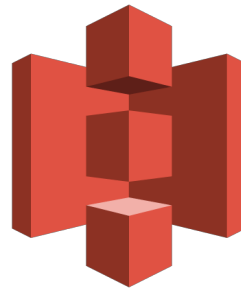
Once my account was set up, Authy began displaying the MFA codes. Codes are good for about thirty seconds each, then they expire and a new code appears.





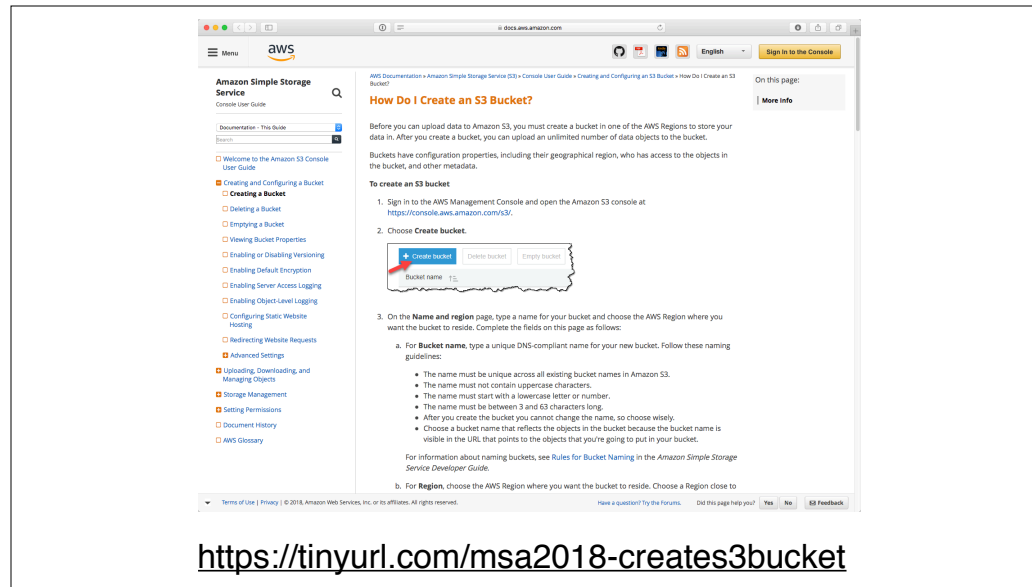
<https://tinyurl.com/msa2018-iampasswordpolicy>

The last security measure on Amazon's recommended list is setting a password policy for your IAM users. It's pretty straightforward for anyone who's had to manage passwords before, but for those folks who want more information about how to set it, please see the link on the screen.

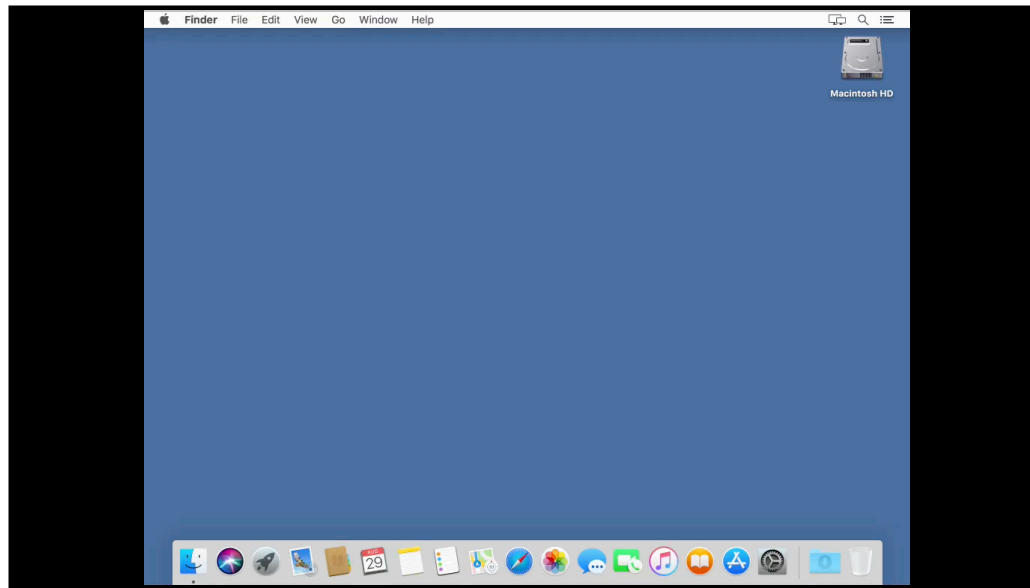


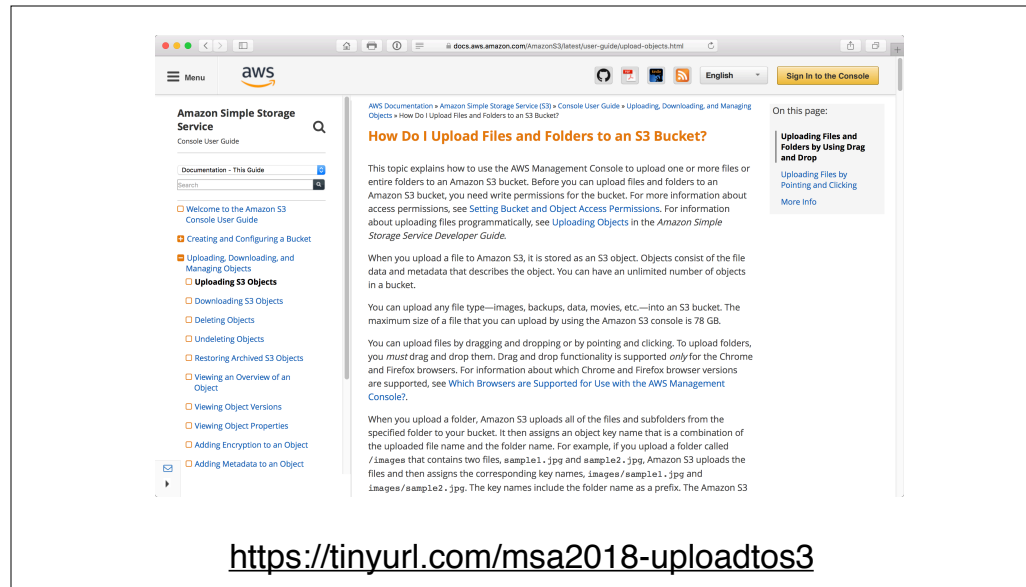
**Amazon
S3**

Now that we've discussed IAM, let's take a look at S3 and some interesting things you can do with it to support your Macs.

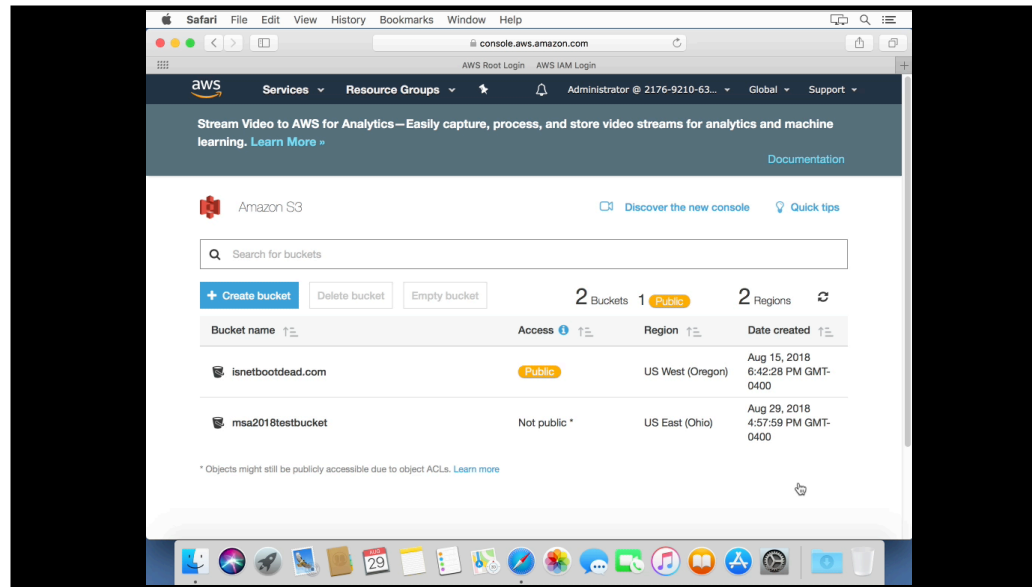


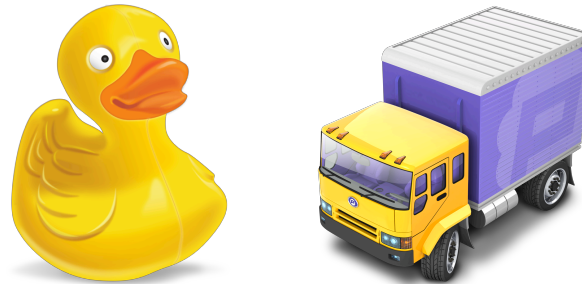
To start from the very beginning, let's create an S3 bucket.





Now that we have our bucket, we should put some stuff in it.

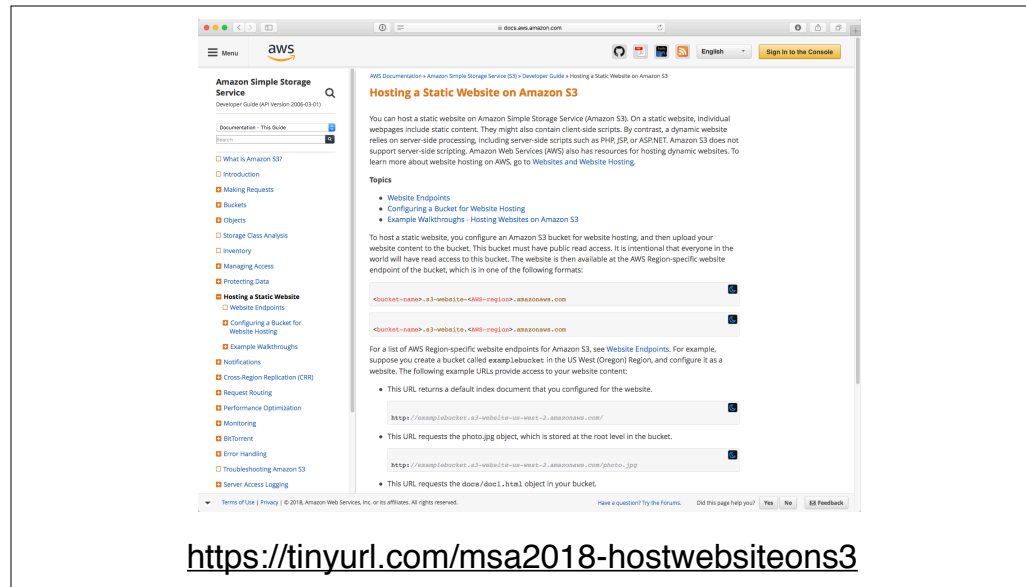




<https://cyberduck.io>

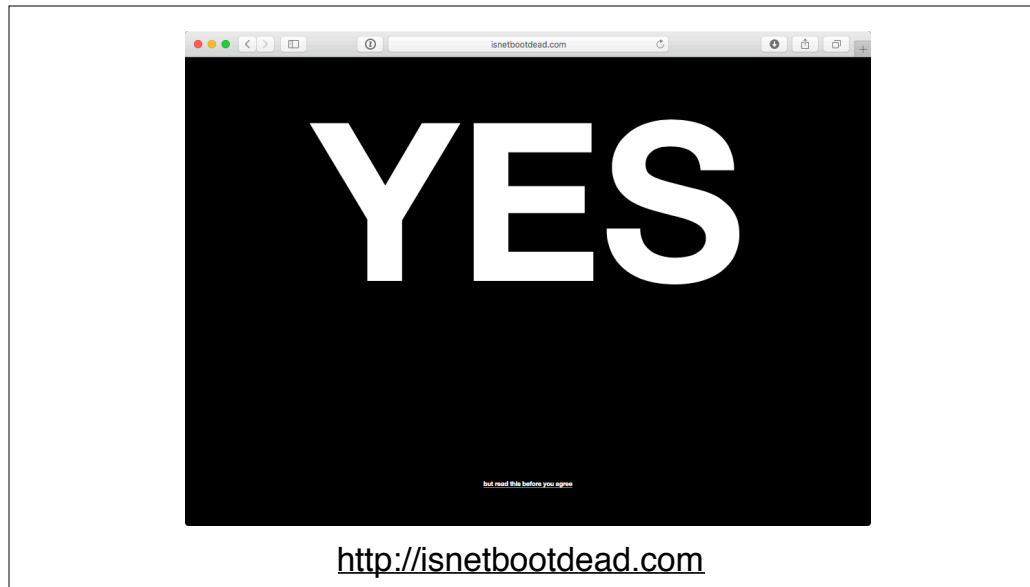
<https://panic.com/transmit/>

If you don't want to use the web console to upload files, there are also a number of S3-compatible file transfer applications available. My usual choice is Cyberduck, a free open-source GUI tool that supports a number of cloud services including S3. Looking at non-free options, Transmit from Panic can work with S3 and there are other shareware options available.

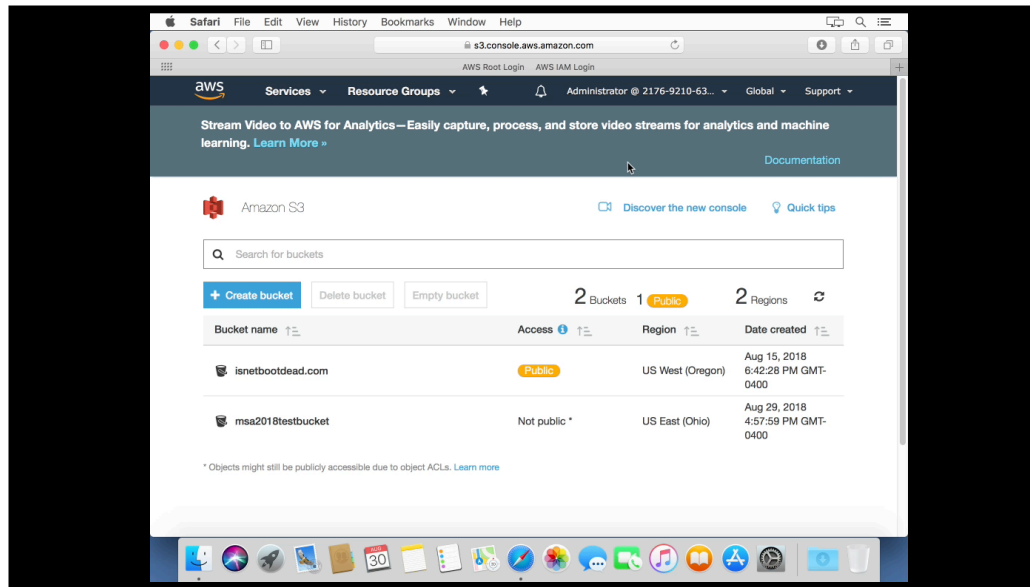


<https://tinyurl.com/msa2018-hostwebsiteons3>

What else? One thing you can do with S3 is host a website. One limitation to be aware of here is that S3 only supports hosting static HTML code, so we're mostly looking at Web 1.0 technology.



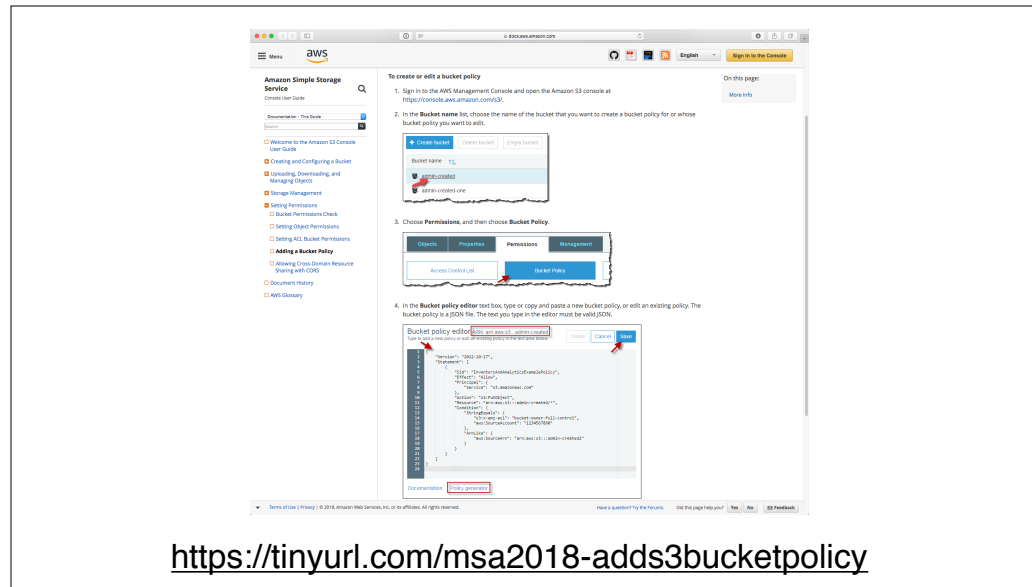
However, that capability may be enough to convey the information you want to provide.



Let's take a look at how this works, starting with setting up a new S3 bucket.

The screenshot shows the 'Static website hosting' configuration window. At the top, the title is 'Static website hosting' with a close button. Below the title, the endpoint is displayed as 'http://example.com.s3-website-us-west-2.amazonaws.com'. There are two radio buttons: 'Use this bucket to host a website' (which is selected) and 'Learn more'. Below this, there are three sections: 'Index document' with a text input field containing 'index.html', 'Error document' with a text input field containing 'error.html', and 'Redirection rules (optional)' with a text area. At the bottom, there are two radio buttons: 'Redirect requests' (with a 'Learn more' link) and 'Disable website hosting'. Finally, there are 'Cancel' and 'Save' buttons at the bottom right.

Now that I've got my files uploaded, I need to do two more things. The first is to set the S3 bucket's properties to allow it to host a static website.



The second is that I need to set a bucket policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::s3_bucket_name_goes_here/*"
    }
  ]
}
```

S3 bucket policies are JSON documents which tell the specified S3 bucket how to behave in certain situations. You will find these kinds of policies used extensively with AWS's various services, with the main differences being which resources are referenced, what permissions are specified and what the actions are.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::s3_bucket_name_goes_here/*"
    }
  ]
}
```

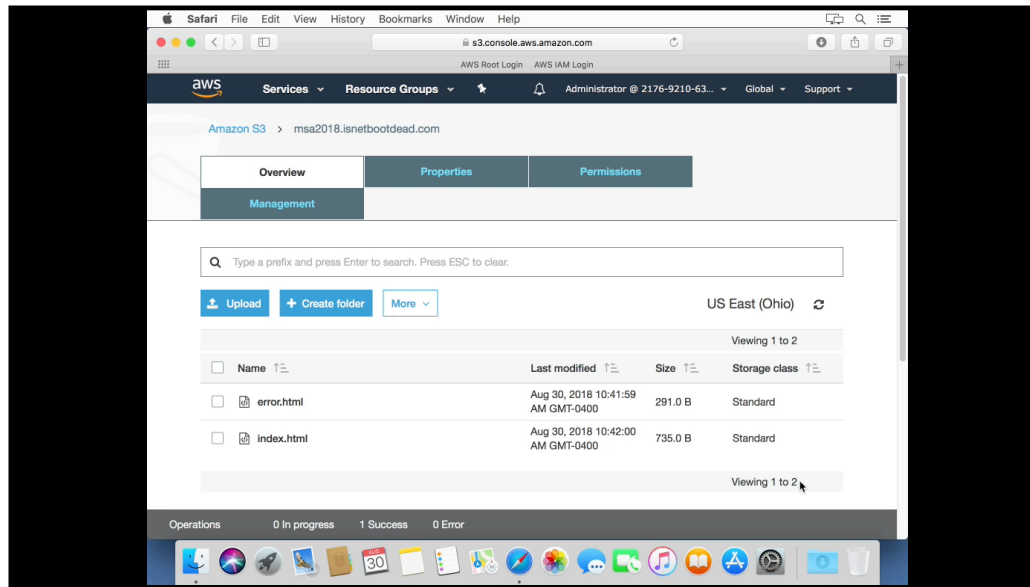
In this case, I'm telling the S3 bucket that I want it to allow anyone to be able to read the objects stored in the S3 bucket. This allows anonymous access via the web.

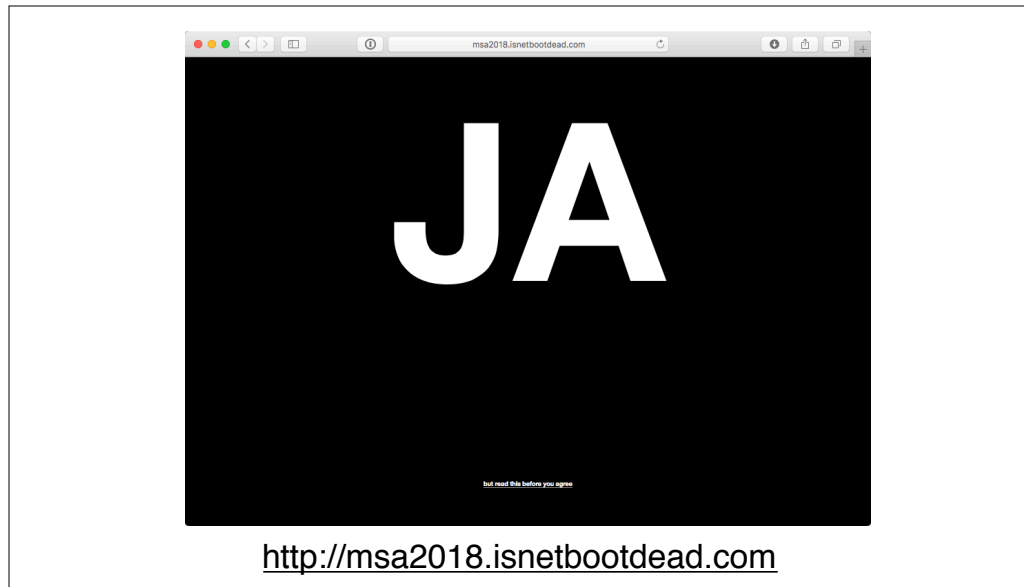
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::s3_bucket_name_goes_here/*"
    }
  ]
}
```

The next part is specifying the S3 bucket in question.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::msa2018.isnetbootdead.com/*"
    }
  ]
}
```

This policy is included with AWS's documentation and designed to be generic, so the only thing you should need to change is putting your own S3 bucket's name in the policy.



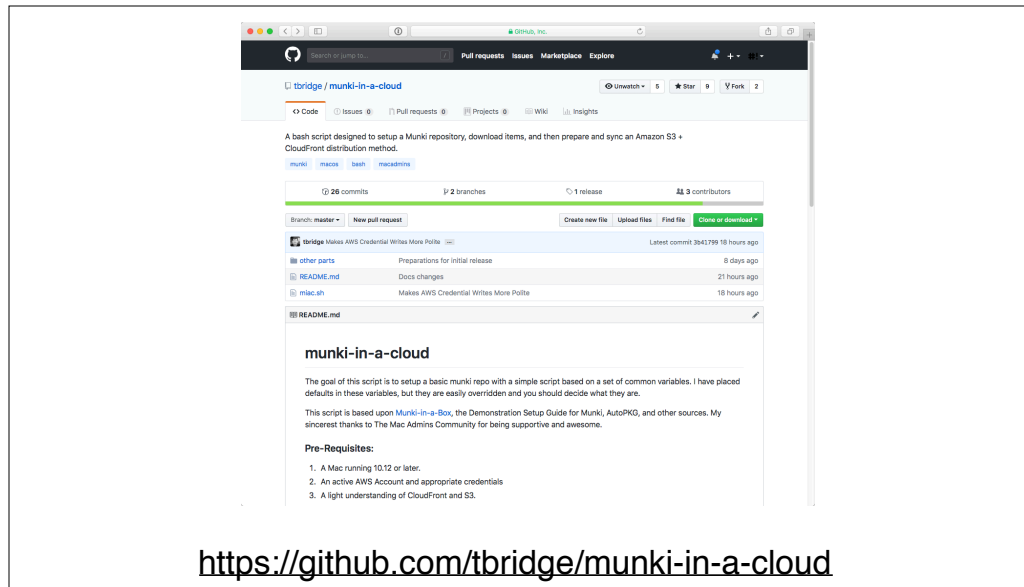


Point a custom DNS CNAME record at your S3 bucket's address and now you have a website for your domain which is backed by AWS's high availability services.

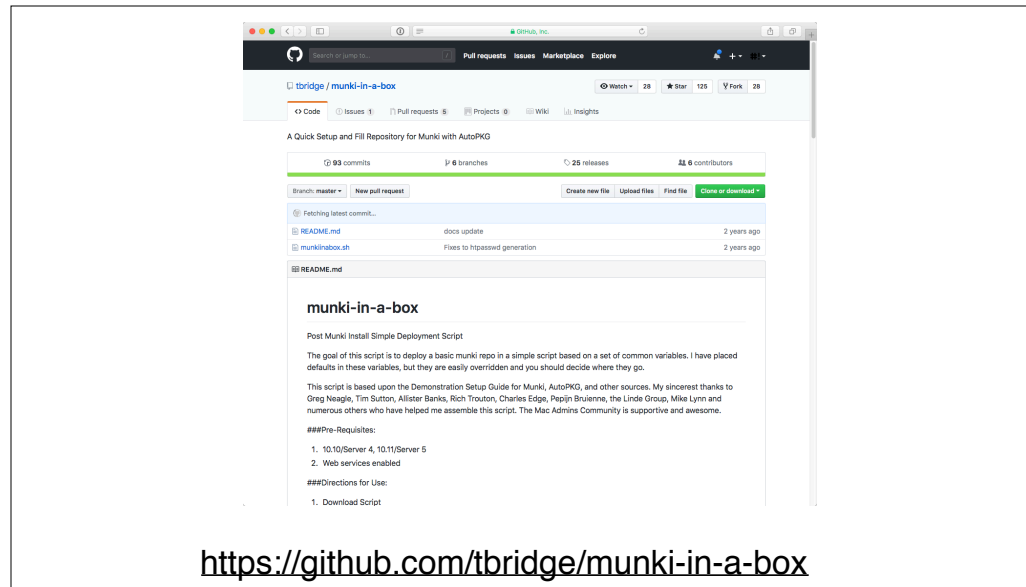
One drawback to hosting a website this way is that the S3 website hosting only uses HTTP and can't by default use HTTPS. There are ways to use AWS's CloudFront service to address this, but I'm not going to cover that as that's getting beyond the scope of an introduction to AWS.



So, big deal right? I can store files on S3 and I can set up a website using HTTP. But what about supporting Macs using S3?



How about using S3 as a Munki repo? My colleague Tom Bridge has a solution for that called Munki in a cloud.

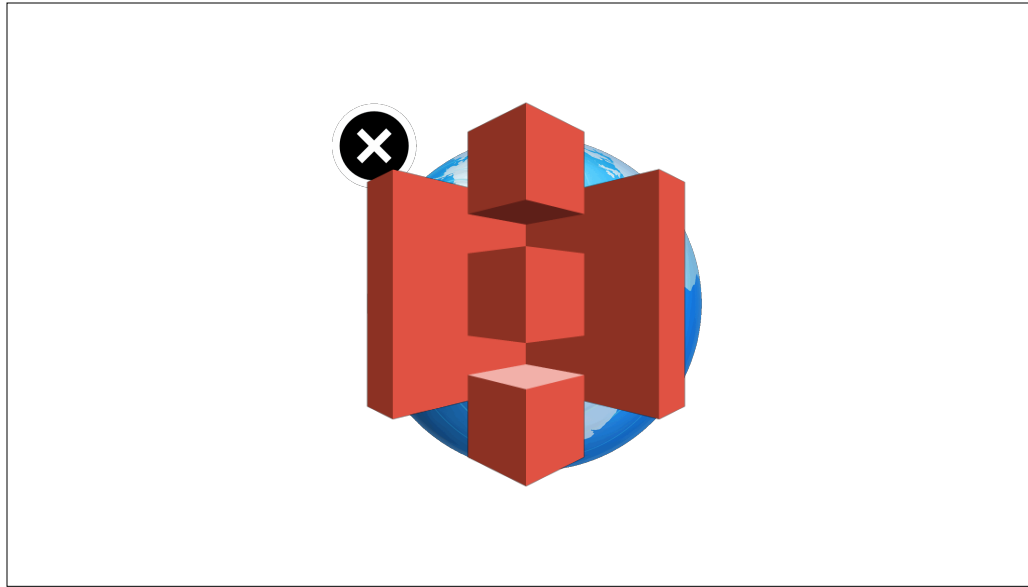


<https://github.com/tbridge/munki-in-a-box>

Munki in a cloud grew out of an earlier solution called Munki in a box. Munki in a box was designed to create a basic Munki repo on macOS along with installing AutoPkg, AutoPkgr, MunkiAdmin, and MunkiReport PHP.



However, Munki in a box has Server.app as a pre-requisite. It uses Server's web services to set up the Munki repo.



As Apple began removing capabilities from Server, including web services, Tom looked around for an alternative and made the decision to use S3.

Pre-requisites

1. A Mac or VM running macOS 10.12.x or later
2. An active AWS account.
3. A programmatic IAM account with:
 - Active access key ID and secret access key
 - Read and write access to S3

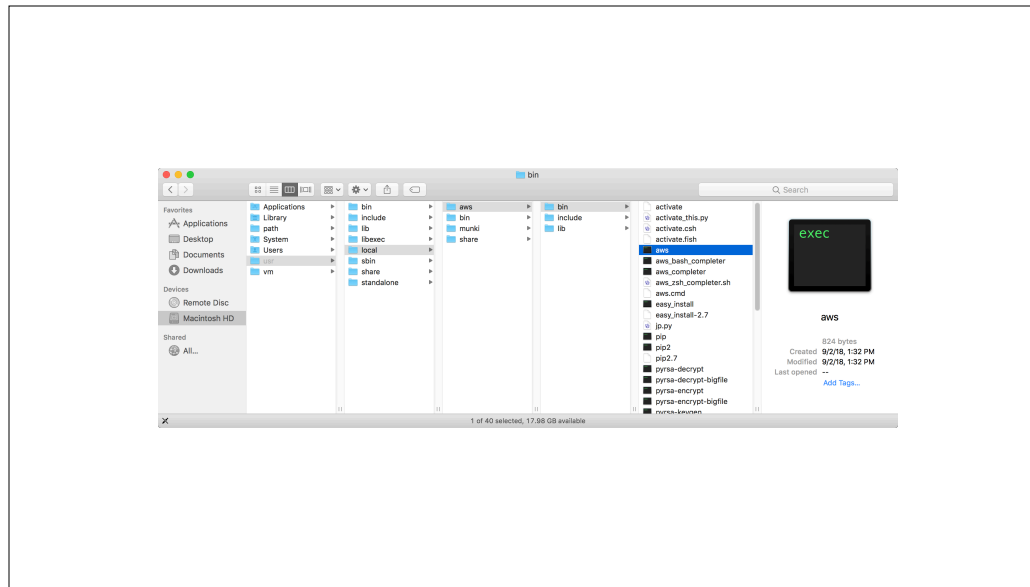
So let's take a look at running Munki in a Cloud. Before we get going, here's the things we'll need.

1. Install **git** if needed.
2. Install the Python **pip** installer tool if needed.
3. Install the PyOpen SSL module if needed.
4. Install AutoPkg if needed.
5. Install Munki if needed.
6. Install the **awscli** tool if needed.
7. Set up a Munki repo and set the logged-in user as the owner.
8. Add specified AutoPkg repos.
9. Run specified AutoPkg recipes to populate the Munki repo.
10. Install AutoPkg.
11. Install Munki Admin.
12. Configure AutoPkg's recipe list.
13. Set up default manifest using the packages added to the Munki repo.
14. Set up new bucket in S3 service.
15. Synchronize Munki repo with S3 bucket.

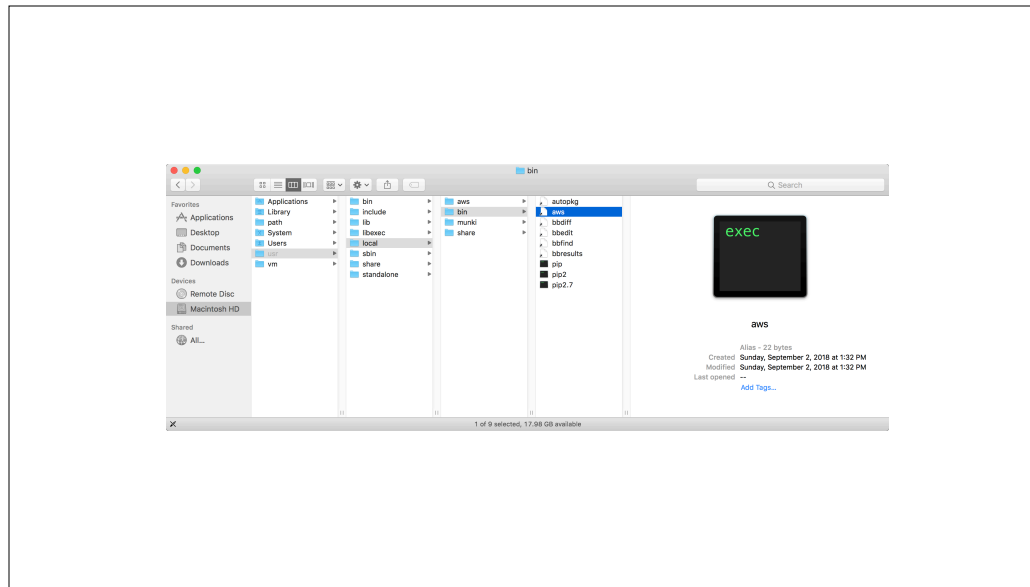
Once we have the prerequisites handled, here's what Munki in a cloud will do.



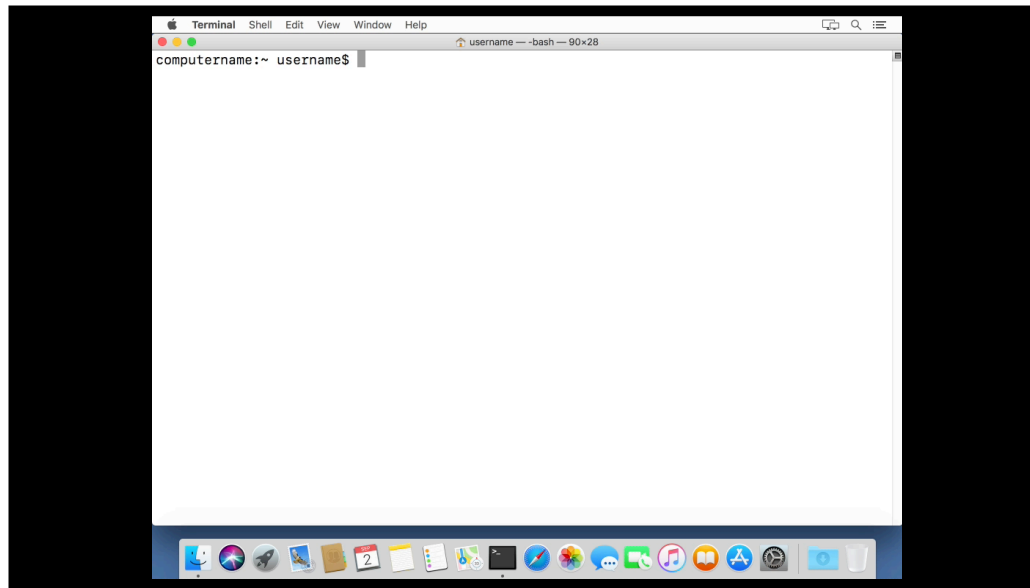
One of the jobs is installing the aws command line tool. This is a Python utility to allow management of AWS services via the command line.

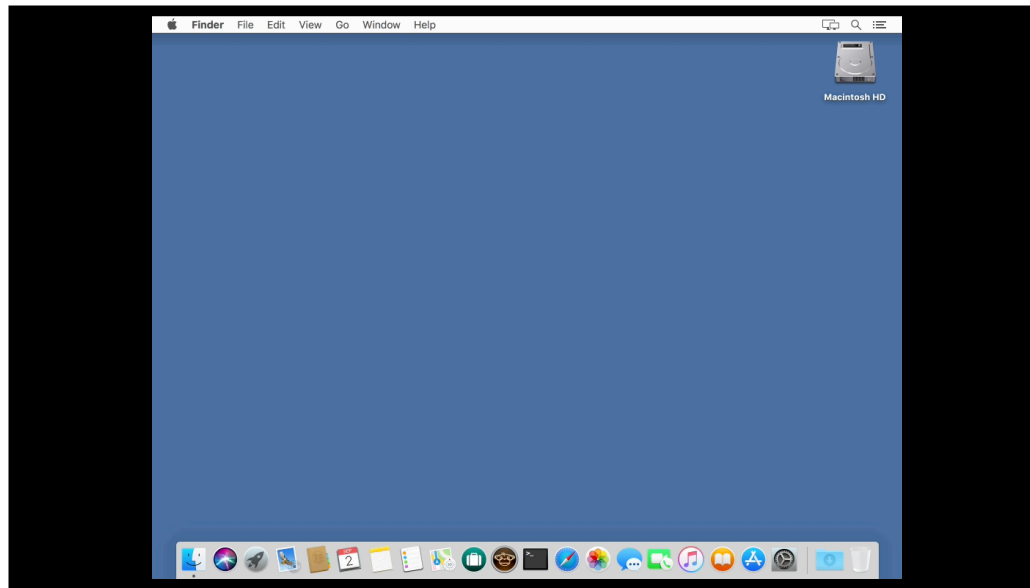


In our case, it's going to be installed to usr local aws bin.



As part of the installation, a symlink will be set up in user local bin so that the default path environment will pick it up.





s3.console.aws.amazon.com

aws

Services

Resource Groups

Administrator @ 2176-9210-63...

Global

Support

Click here to learn how to store and access objects in S3 via NFS Click here »

Documentation

Amazon S3

Discover the new console Quick tips

Search for buckets

Create bucket

Delete bucket


Empty bucket

4 Buckets 2 Public 3 Regions

Bucket name	Access	Region	Date created
d69d9ad9-9f18-4238-bfc6-ebf3af4713d0-miac	Not public *	US East (N. Virginia)	Sep 2, 2018 1:32:28 PM GMT-0400
isnetbootdead.com	Public	US West (Oregon)	Aug 15, 2018 6:42:28 PM GMT-0400
msa2018.isnetbootdead.com	Public	US East (Ohio)	Aug 30, 2018 10:41:29 AM GMT-0400
msa2018testbucket	Not public *	US East (Ohio)	Aug 29, 2018 4:57:59 PM GMT-0400

* Objects might still be publicly accessible due to object ACLs. Learn more

Operations 0 In progress 4 Success 0 Error



A terminal window with a title bar showing 'username -- bash -- 108x5'. The terminal text is as follows:

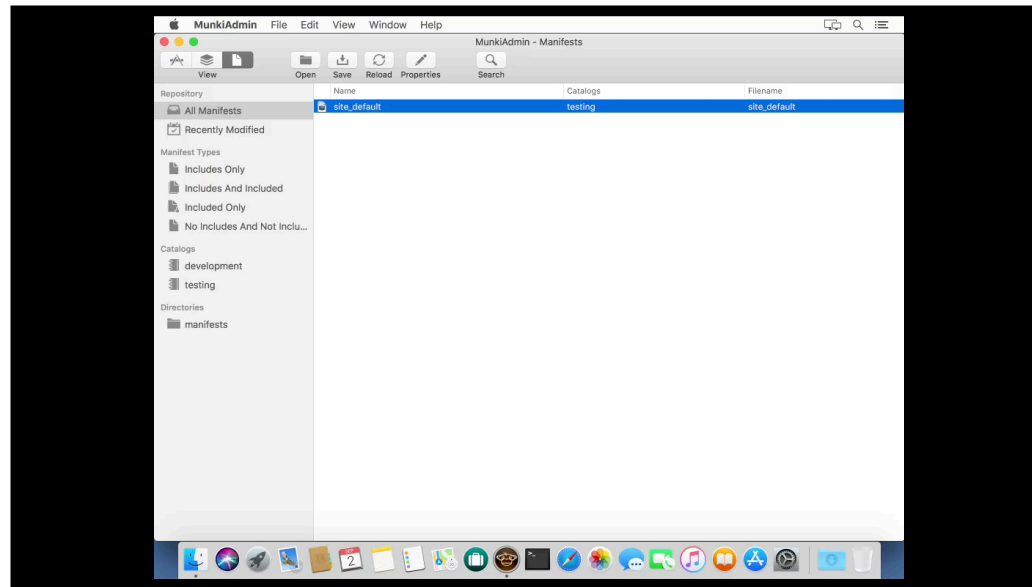
```
computername:~ username$ aws s3 sync /Users/Shared/munki_repo s3://d69d8ad8-9f18-4238-bfc6-ebf3af4713d0-miac  
--exclude '*.git/*' --exclude '.DS_Store' --delete
```

Below the terminal window, the same command is written in a larger font:

```
aws s3 sync /path/to/munki_repo s3://S3_BUCKET_NAME_HERE --exclude '*.git/*' --exclude '.DS_Store' --delete
```

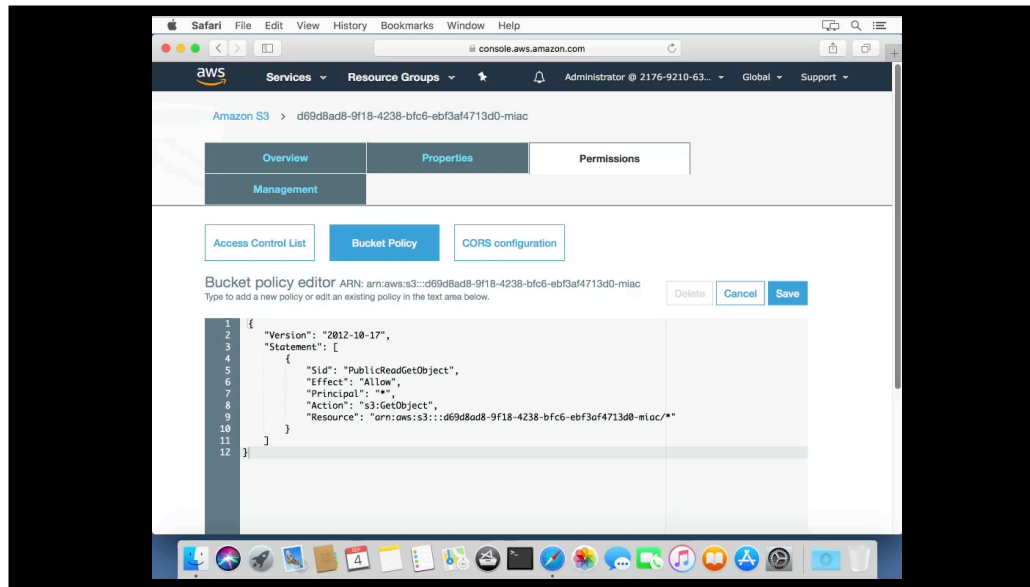
Now that I've got my repo up in S3, how do I update it? Running the command shown on the screen will tell the aws command line tool to update my S3 bucket with the current contents of my Munki repo.

It'll also tell the sync process to ignore certain unwanted files and to delete from the S3 bucket anything that isn't currently in my repo.





Well, this is great. I've got my Munki repo set up in S3 and life is good. How do I tell my Mac to use it?

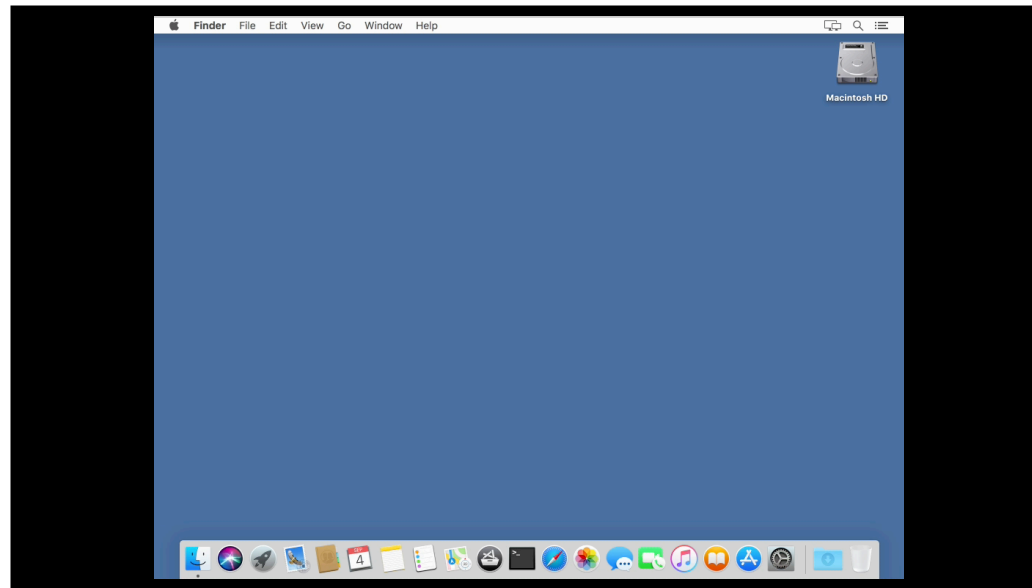


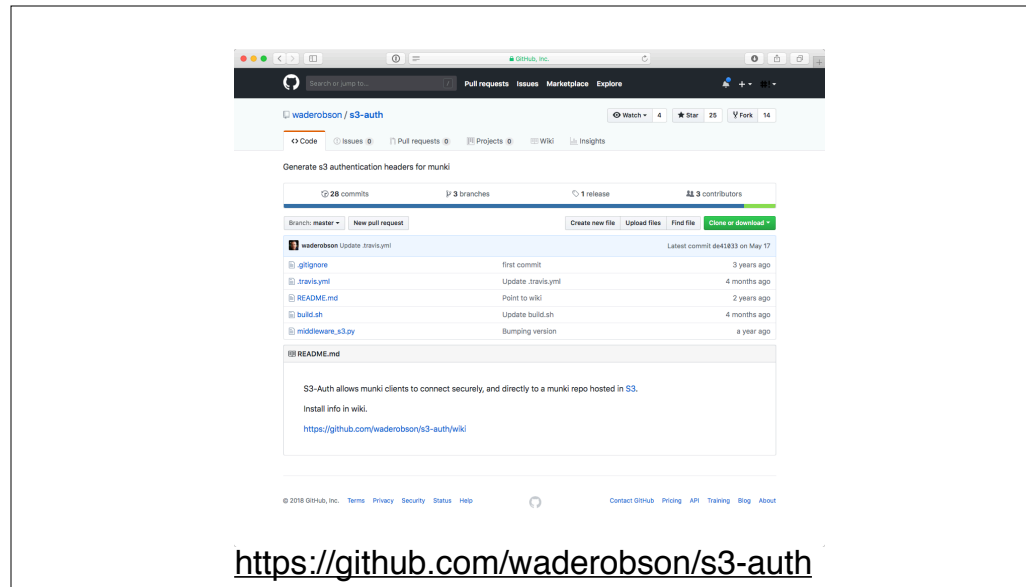
One option is to make your Munki repo public using a bucket policy similar to the one used to set up a website in S3.

Public S3-hosted Munki repo

1. No special authentication needed.
2. Requires S3 bucket be configured to allow anonymous read-only access.
3. Mac can be configured to access repo without additional plug-ins or tools required.
4. Address of repo:
 - <https://s3-bucket-name-here.s3.amazonaws.com>

If the only things you're putting into your Munki repo are items publicly available elsewhere on the Internet, this may be a valid option for you. In this case, you're treating S3 like you would any other webserver which allows anonymous read-only access.

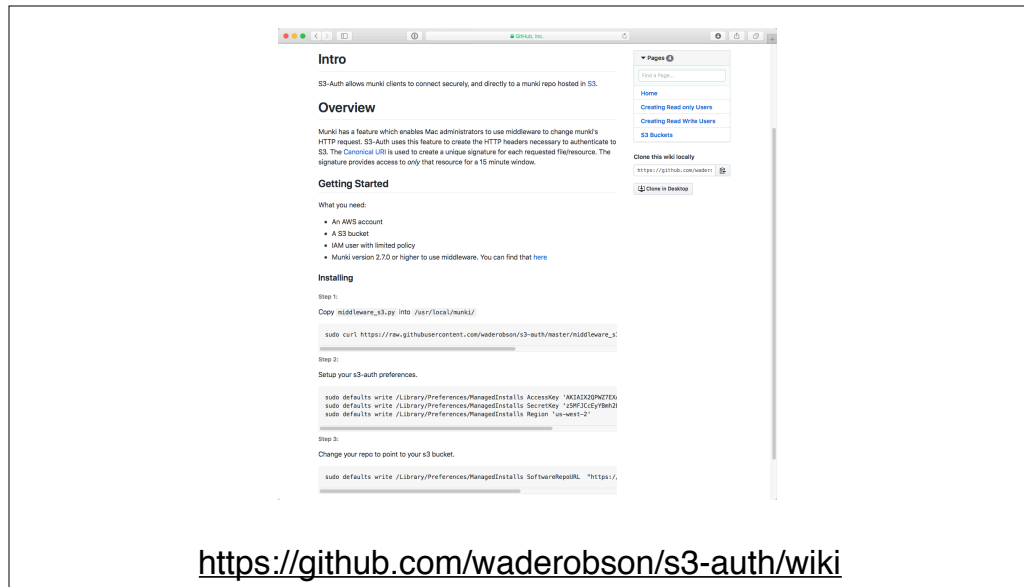




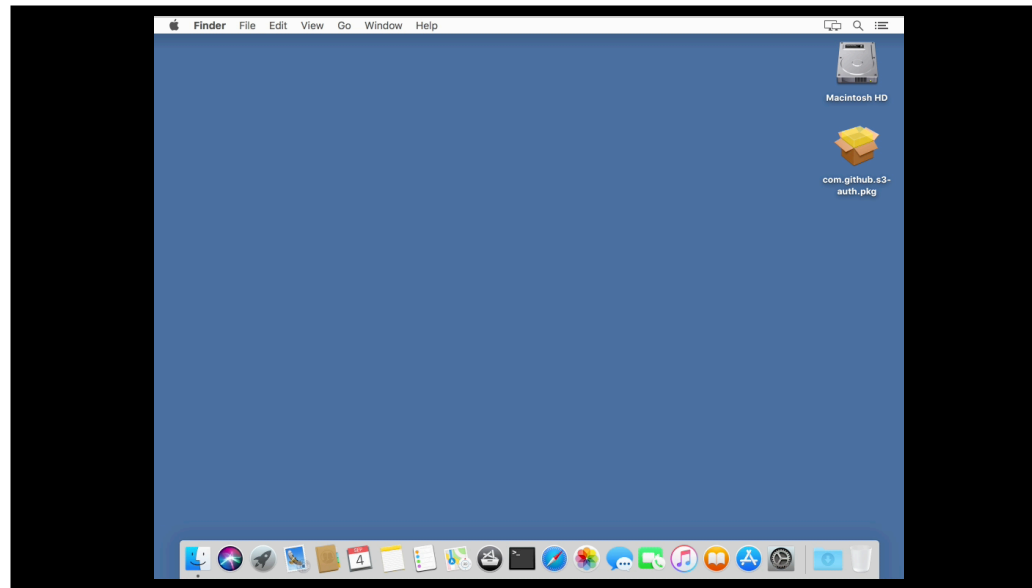
For those folks with security needs that rule out using a publicly accessible S3 bucket, there are other options. One option was developed by Wade Robson, who wrote a Munki middleware solution that allows Munki to connect securely to a S3 bucket which doesn't permit anonymous access.

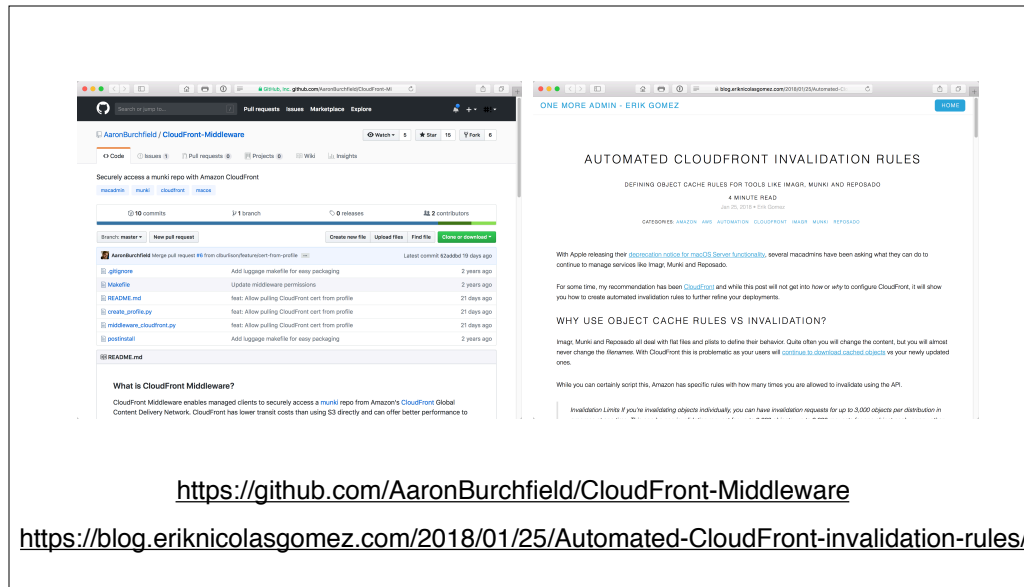
Pre-requisites

1. Munki version 2.7.0 or higher
2. An active AWS account.
3. Munki repo stored in an S3 bucket
4. A programmatic IAM account with:
 - Active access key ID and secret access key
 - Read-only access to S3 bucket



Once the pre-requisites are handled, the setup is straightforward. Munki is configured with the address of the S3 bucket along with the appropriate AWS region and the authentication credentials.





<https://github.com/AaronBurchfield/CloudFront-Middleware>

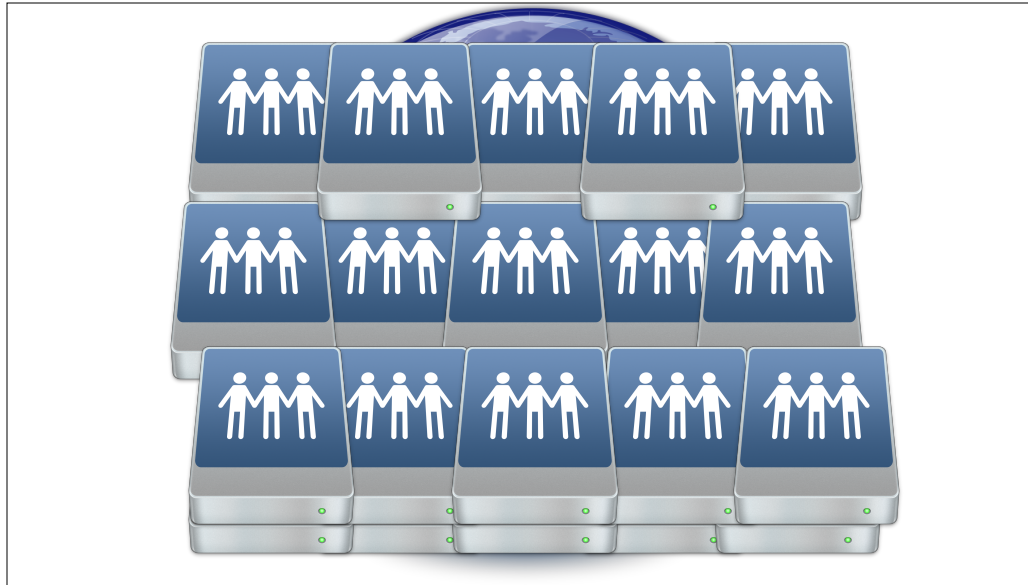
<https://blog.eriknicolasgomez.com/2018/01/25/Automated-CloudFront-invalidation-rules/>

You can also leverage CloudFront with S3 to get expiring URLs for downloads and also lower your overall costs. However, this is a more advanced topic and I'm not going into detail on it. For those interested in exploring those options, I recommend checking out Aaron Burchfield's tools on GitHub and also Erik Gomez's blog post.

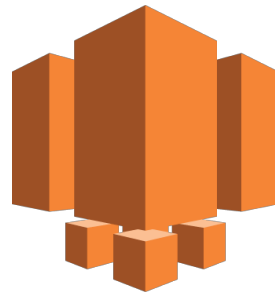


For folks using Jamf Pro, you can also use S3. In my own shop, we're using an S3 cloud distribution point with our Jamf Pro service.

The process of setting up a cloud distribution point on S3 gets into some more advanced areas, so for those interested, I recommend checking out the link on the screen as I've documented the process. As part of setting up the cloud DP, you will need to get a particular encryption key pair which is only accessible by the root user of the account. This is going to be one of the exceptional cases when you need to log in as the root user of the account to get something done.

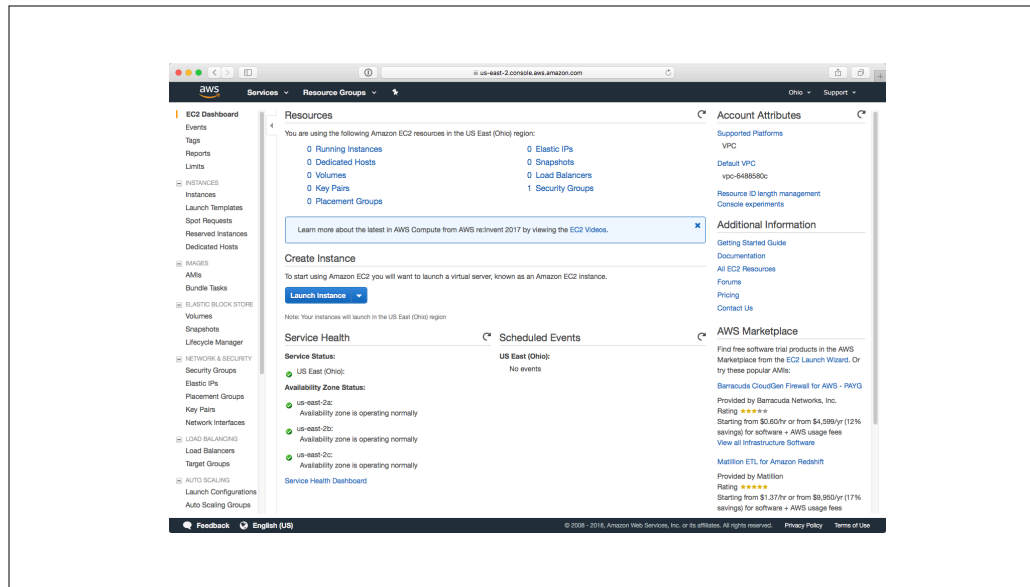


Hosting in S3 allowed us to retire a large number of local distribution points in favor of one global distribution point. Once new software is uploaded to S3, it's instantly available worldwide.

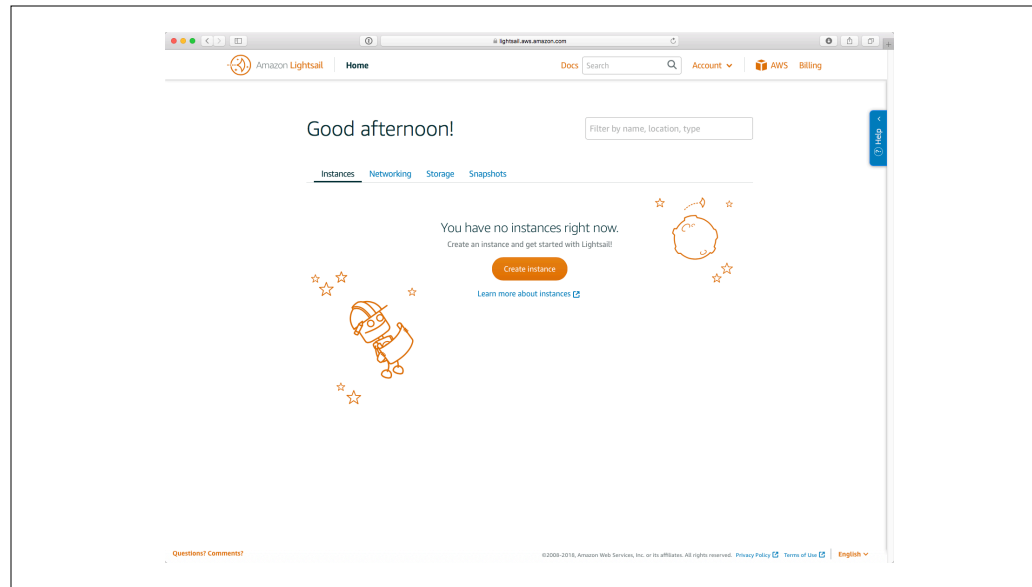


**Amazon
Lightsail**

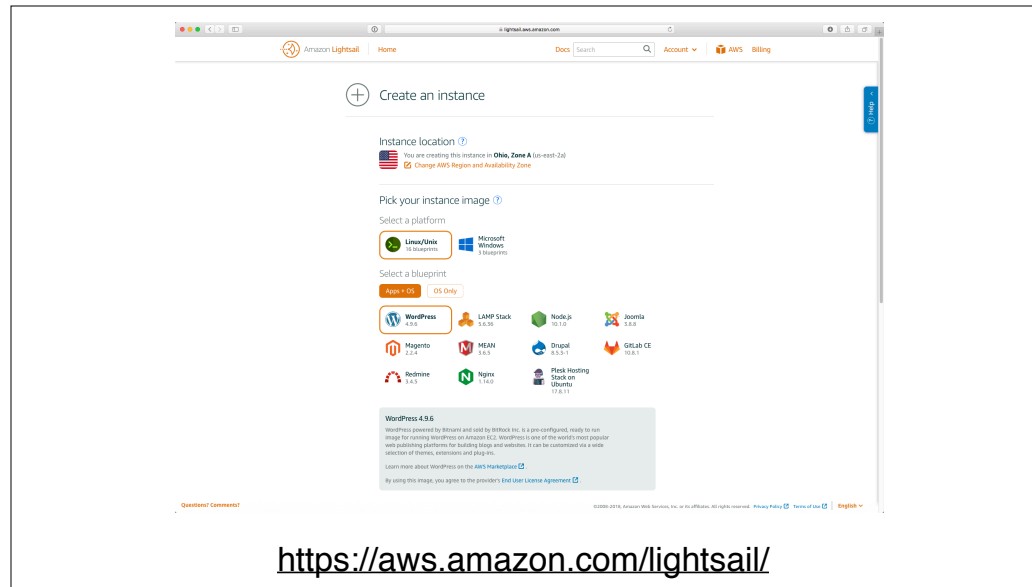
Sometimes you need more than storage. You need a server. This is where AWS services like Lightsail come in. Lightsail is a simplified interface for AWS's EC2 service.



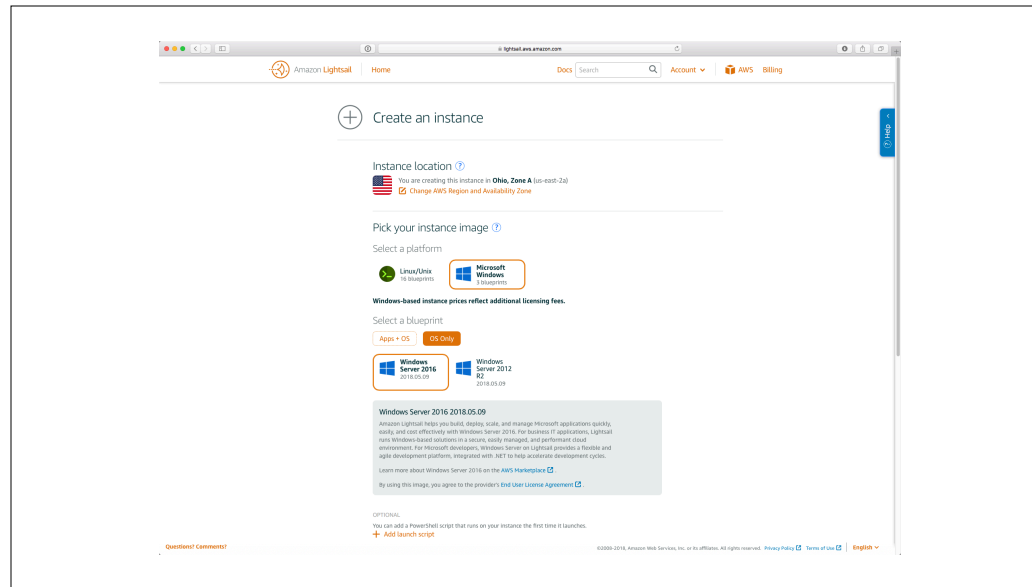
Why use Lightsail over EC2 if you're getting started? One reason is user experience. Here's how EC2 looks if you don't have any instances set up.



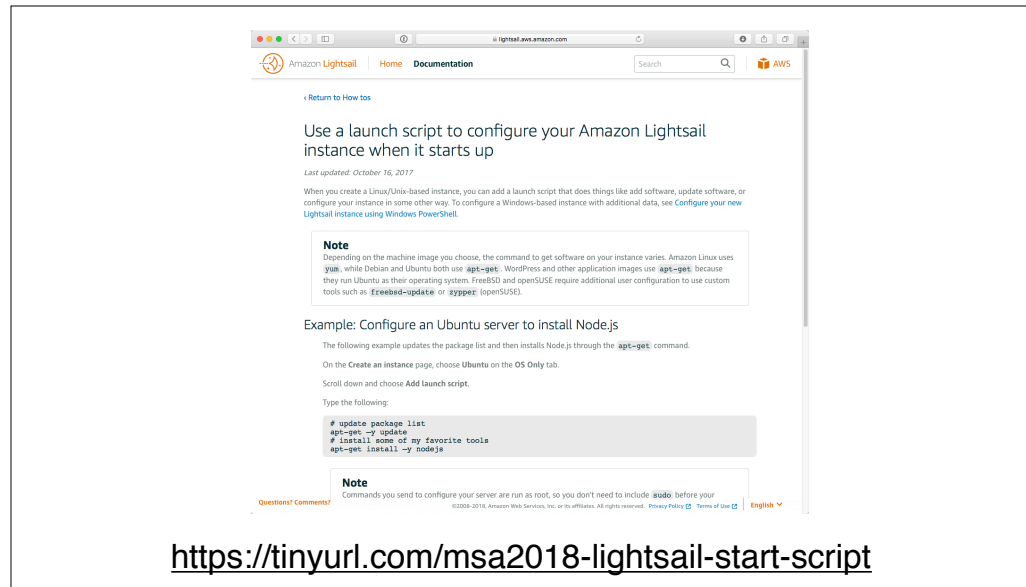
Here's what Lightsail looks like. For a beginner to AWS, it's a much smoother and self-explanatory experience.



Lightsail is designed to get someone quickly set up with a virtual server, using a variety of application templates.

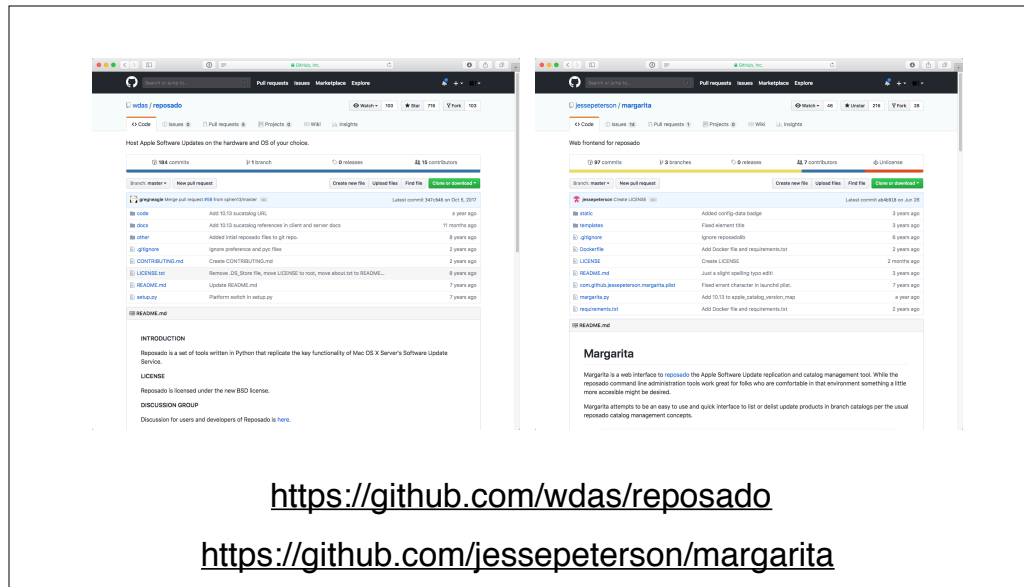


In addition to Linux, Windows Server 2012 and 2016 are supported along with SQL Server.

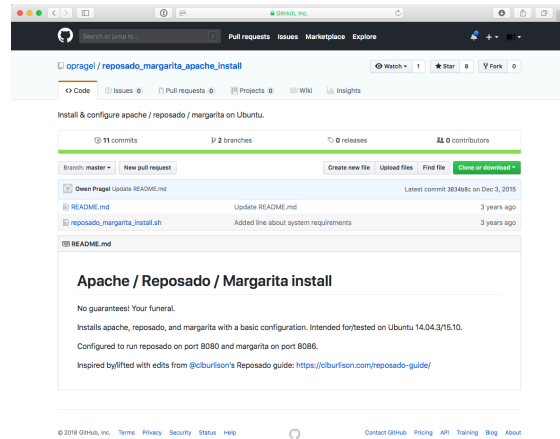


<https://tinyurl.com/msa2018-lightsail-start-script>

One nice feature of Lightsail is that if you're using Linux, you can add a script to configure the instance when it starts up for the first time.

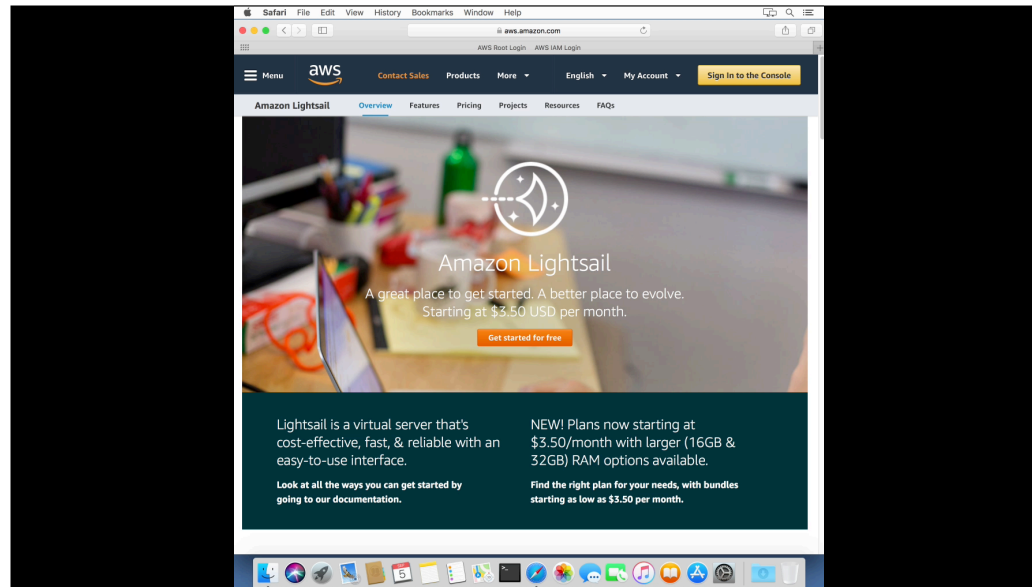


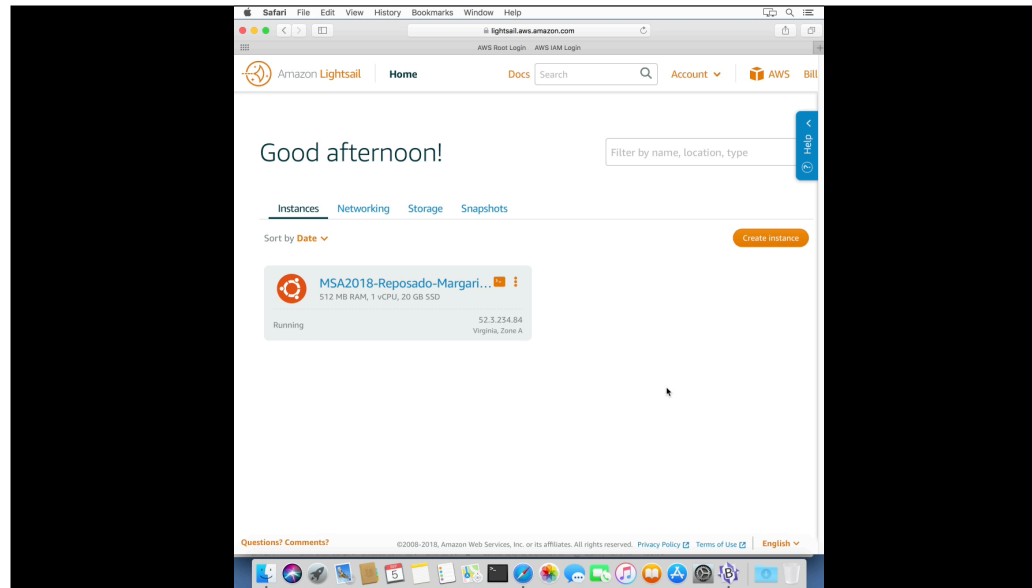
We're going to use that capability to create a Linux instance running Reposado and Margarita. For those not familiar with these tools, Reposado is an open-source utility that allows you to host and distribute Apple software updates. Margarita is a complementary open-source tool for managing Reposado using a web interface.

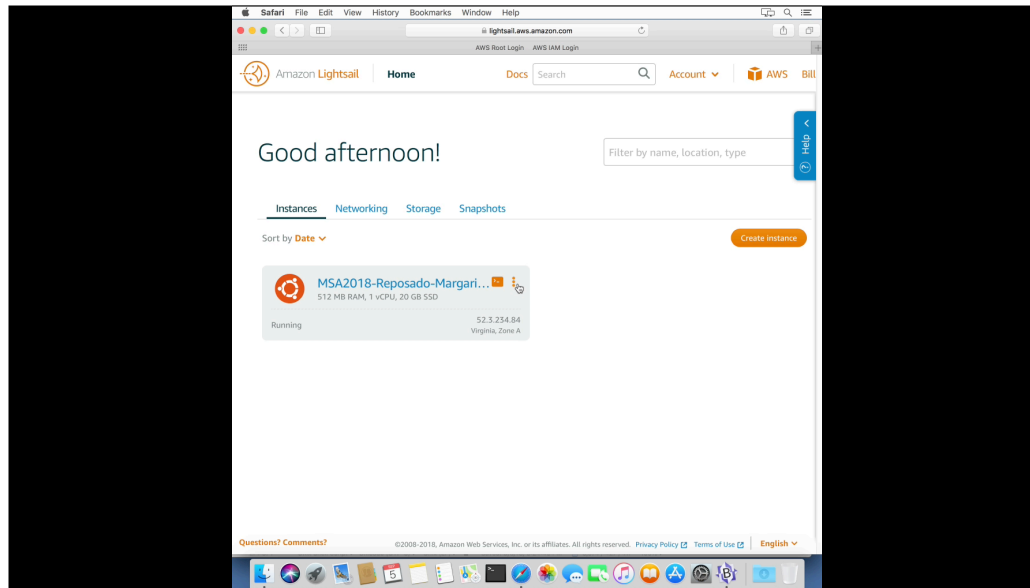


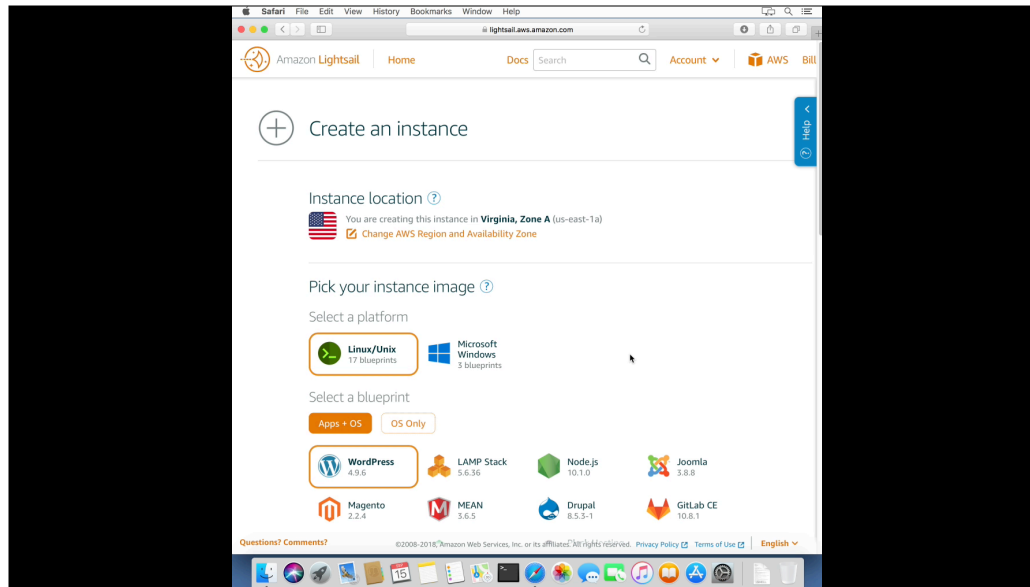
https://github.com/opragel/reposado_margarita_apache_install

For this task, I'm going to use a script written by my colleague Owen Pragel as the setup script.

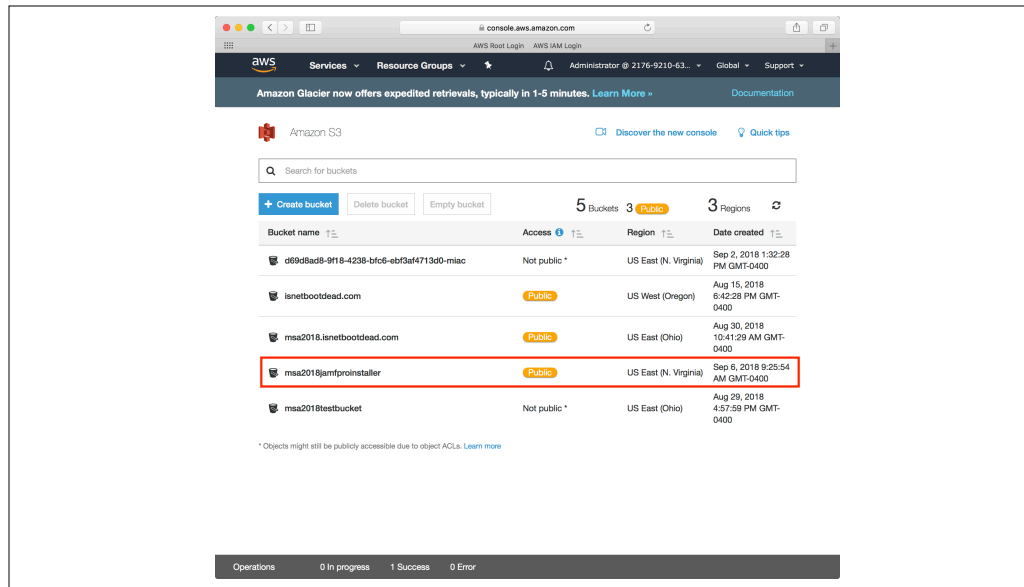




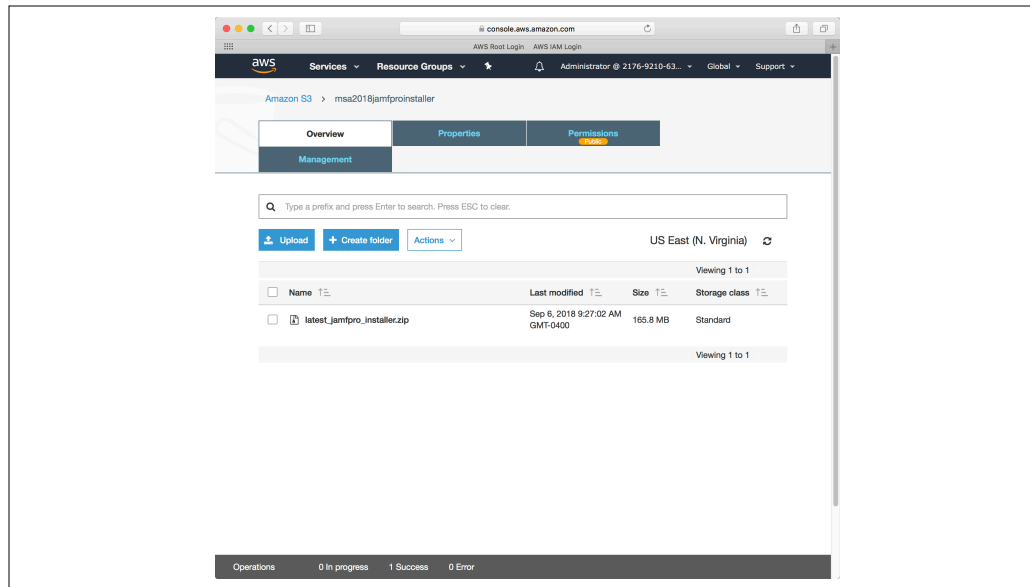




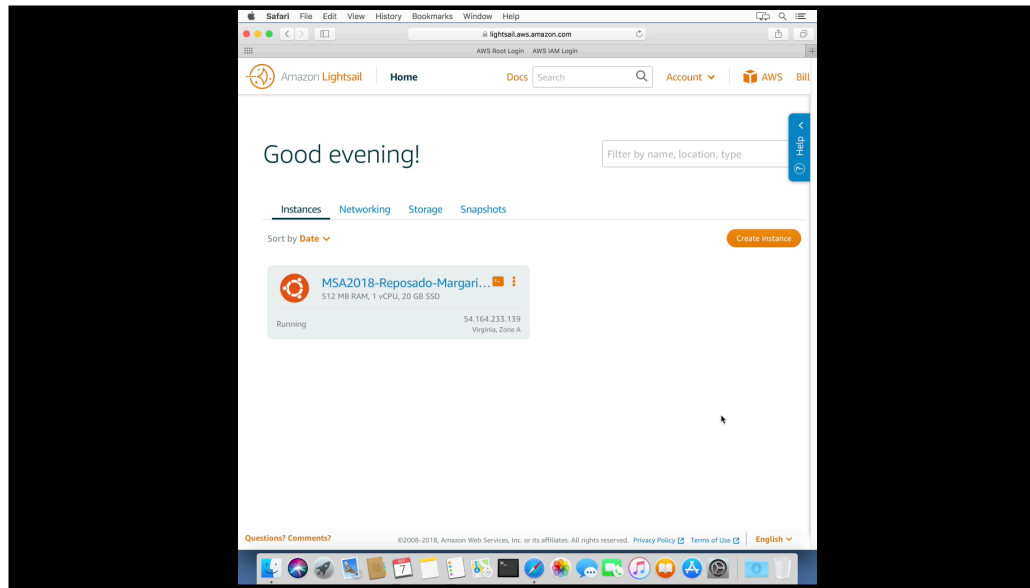
What if you want to use your own SSH keys with a Lightsail instance?

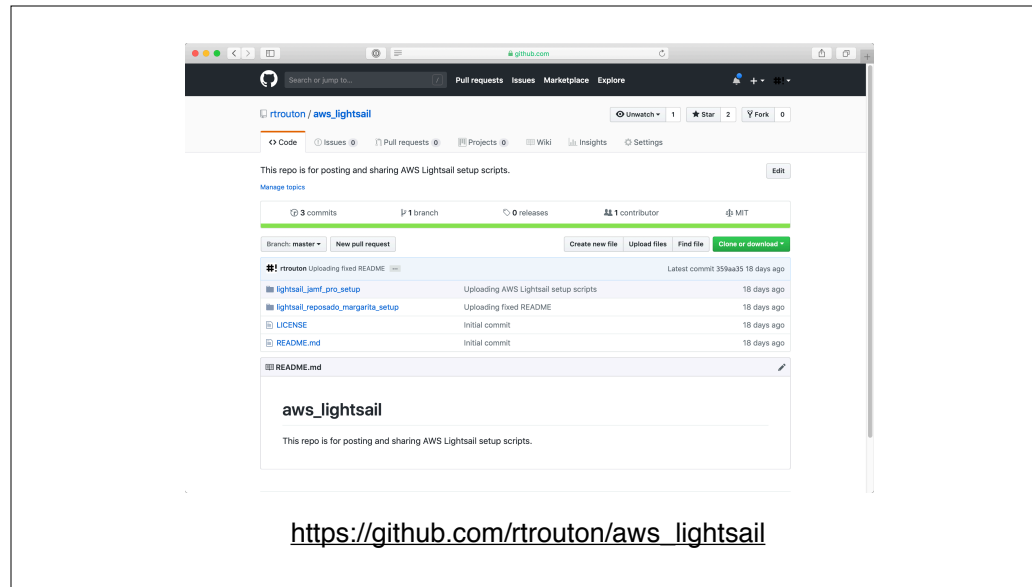


You can also set up a Jamf Pro server on Lightsail. To help me with this process, I'm setting up a new S3 bucket which allows public access to its contents.



Inside that bucket, I'm putting a copy of the latest Linux installer for Jamf Pro.





For those who want a copy of the scripts I used with my Lightsail examples, I've posted them to Github. They're available via the link on the screen.



So for folks who were looking at Server's decline and wondering what's next, hopefully the information I've given you is helpful in getting you started with the services available from Amazon.

Useful Links

AWS Getting Started Resource Center: <https://aws.amazon.com/getting-started/>

AWS 10-Minute Tutorials: <https://aws.amazon.com/getting-started/tutorials/>

Getting Started with IAM: <https://aws.amazon.com/iam/getting-started/>

Using AWS S3 to Store Static Assets and File Uploads:
<https://devcenter.heroku.com/articles/s3>

Useful Links

Getting Started with Lightsail: a Simple VPS Solution from AWS: <https://linuxacademy.com/howtoguides/posts/show/topic/12662-getting-started-with-lightsail-a-simple-vps-solution-from-aws>

Getting started with AWS: <https://medium.com/tfogo/getting-started-with-aws-d7c51133fc92>

How to serve your website on port 80 or 443 using AWS Load Balancers: <https://medium.com/tfogo/how-to-serve-your-website-on-port-80-or-443-using-aws-load-balancers-a3b84781d730>

Downloads

PDF available from the following link:

<http://tinyurl.com/MSA2018AWSPDF>

Keynote slides available from the
following link:

<http://tinyurl.com/MSA2018AWSKeynote>