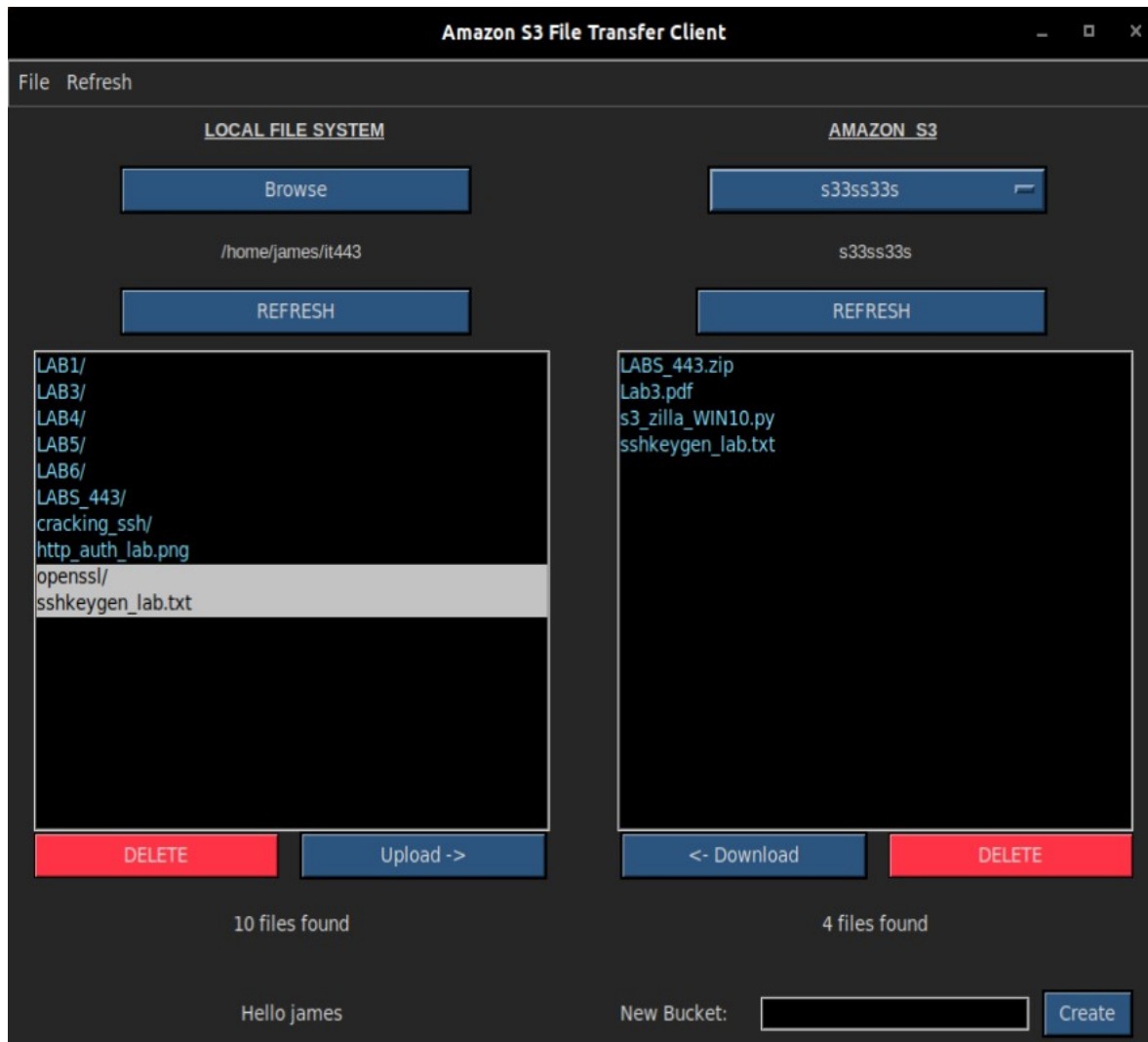


## S3Zilla – An Amazon S3 File Transfer Client for Linux and Windows



## DESIGN

The following S3 methods are explicitly used within the code.

### Amazon S3 API Methods:

**s3.create\_bucket(bucket\_name)** - for creating a new bucket

**s3.Bucket(bucket\_name)** - accepts a bucket name to “use” for uploading/downloading etc.

**s3c.download\_file(file, bucket, file\_name)** for downloading files from a bucket

**s3c.upload\_file(file, bucket, file\_name)** for uploading a file to a given S3 bucket

**s3.buckets.all()** for retrieving all available bucket names

**s3.bucket.objects.all()** to retrieve the contents of a particular S3 bucket (with it's accompanying method variable available to instances of the class named s3\_file.key)

**delete()** method for deleting files from a given bucket.

## CLASS S3Zilla() METHODS:

**\_\_init\_\_(self, master)** – This is the entire program's constructor. All buttons and widgets are initialized here. Self.master is the parent widget and basically used to subclass the parent Tkinter Frame Widget. Most of the instances created in the constructor have an accompanying grid() method that sets placement of the widget.

**quit()** - This method quits the program. It's only called by the cascading menu set in the constructor on line 53.

**get\_local\_sel()** - This method consists of one simple but very important line of code. It's called in several places and uses two built in Tkinter methods, *get()* and *curselection()*. These two methods are used to retrieve anything that the user highlights within a listbox. The instance ex\_loc is the name that I gave to the listbox instance that acts as the "local file explorer" . It displays the contents of any chosen directory on the local file system.

**get\_s3\_sel()** - This method mirrors the above *get\_local\_sel()* method. The key difference being that this is used to get the selection of the second file explorer named ex\_s3 (an abbreviation for explorer\_s3). The ex\_s3 listbox displays the contents of any chosen S3 bucket. This method returns any chosen/highlighted files inside that listbox.

**set\_drop\_val()** - This is an interesting little method that is only called once inside the constructor when the program initially loads. It works with an instance of the *OptionMenu()* class - a newer edition to Tkinter that works slightly differently than the rest of the widgets. This widget is very complex to set up because of this. While the option menu object is created on line 99, it actually calls the *set\_drop\_val()* method, and discretely passes combined arguments via the \*self.dropdown\_data argument. The self.dropdown variable is set to *StringVar()*. And self.dropdown data is really just a list of all buckets that the user has available in their S3 account. The *OptionMenu()* needs *StringVar()* as it's an addition to Tkinter. This is because StringVar provides functions for "helping" the TCL interpreter (Tkinter-Python interface to Tcl/Tk, 1). The TCL interpreter is the shell that runs Tkinter GUI applications (1).

**delete\_local\_records()** - This method is called by the “action” on line 217 inside the instance of the delete local *Button()* widget. When the delete button for the local file system is pressed, this method is called as a result. It calls the *get\_local\_sel()* method to see what file(s) were selected in the local file explorer. If there is a valid selection, the *del\_local()* method is called; this is the method that does the actual deleting, and it accepts a list of filenames to delete. Otherwise the status label is updated with a message informing the user that no valid selection has been made.

**del\_local(files\_remaining)** – This method accepts a list of file names passed from the above *delete\_local\_records()*. The method knows the right location/directory that contains the files because the *self.dir* variable is preset with the location before the method is called. This is a recursive function as it calls itself, popping each file name off of the stack and deleting with each iteration. It also sets the status label notifying the user that the files are deleted. The *os.remove()* function and the *shutil.rmtree()* function were used to delete the files and directories respectively. The recursion depth stops as soon as the list of files that was passed in (*files\_empty*) actually becomes empty.

**del\_s3\_records()** - This is the S3 version of the delete method. It is not recursive, but it does utilize the *s3.Bucket()*, *delete()*, and the *bucket.objects.all()* methods from the S3 API. The method looks for the chosen files inside a preset *self.drp\_sel* variable, iterates through the list, and simply deletes any that match. The status label is updated accordingly depending on a successful delete.

**load\_dir()** - This is a small method that uses a Tkinter method named *askdirectory()*. It “magically” opens a file explorer reflecting the current system settings (ie: Windows file explorer vs a Linux file explorer) where the user may pick/highlight a directory and either press ‘OK’ or ‘Cancel’. If a selection is made, the selection/file directory name is saved into the variable *self.dir* and the status label is updated with the chosen directory.

**refresh\_local()** - This method is very important and is required to run in some instances because it sets status labels depending on Boolean variables, often after pressing the “refresh button”: If files were just deleted, it knows to print “FINISHED DELETING”. It also displays the number of files found in the selected directory inside of a status label. It checks the given directory to see what files are there, and displays them in the local file explorer (listbox). In some cases (such as after deleting a file), this method is called automatically without the “refresh button” being pressed. That gives an automatic and dynamic feel to the file explorer as it can update itself within the code simply by calling this method. It also severs to check for errors. If a user does not select a file or directory, it displays a message saying so with a label. It also says “hello” to the user upon start up by calling the *getpass.getuser()* method and updating the label accordingly from within the constructor.

**refresh\_s3()** - This method accomplishes basically the same tasks as the above method and is equally as important. It ensures a bucket is selected that needs to be refreshed. Similar to the above method, it uses the built-in Tkinter methods *delete()* and *insert()* which reference the the file explorer only, not the files themselves. It refreshes the S3 Explorer by reflecting what’s actually found inside the S3 bucket. This method is also automatically called by other methods as well as with the “refresh” button on the right-hand/S3 side of the application.

**finished(incoming\_message)** - This is a small function that isn’t all that important, but it’s more for “user experience”. It’s responsible for actually printing the central status label after a file has been uploaded or downloaded for either of file explorers. It accepts a string, and it prints the string onto the label (similar to a print function for alerting the coder or end-user of certain events or errors). As an extra feature, there is a .1 second delay in printing. For example, it will print “example” like so:

```
e
ex
exa
exam
examp
example
```

The slight delay in between iterations gives a fun marquis or banner effect when the message “FINISHED DOWNLOADING” or “FINISHED UPLOADING” is printed.

**upload()** - Upload is a very import method: It accomplishes the actual uploading of files. It does some error checking to ensure that the user has selected a local directory and a file to upload as well as an S3 bucket for a destination. If the user has forgotten to select anything, they get notified accordingly. Then the method checks to see if the selection is a file or a directory. If the selection is a directory, it's converted to .zip format via the handy built-in *shutil.make\_archive()* method. After uploading, if a directory has been compressed, the newly-made compressed version is removed after uploading a copy (clean-up is performed). The S3 API method *upload()* is used to upload the chosen file.

**download()** - This method downloads a file from a given S3 bucket to a selected local directory destination after some error checking. It uses the S3 API method *s3c.download\_file()* to do the heavy lifting. This method as well as the above *upload()* method both call the *finished()* method to print a message. Both methods also call the previously mentioned *refresh\_local()* or *refresh\_s3()* method as well to automatically update the file explorers.

**create\_bucket()** - This method is called when the "create" button is pressed. There must be a file name specified in the associated Text widget, *self.create\_bucket\_name*. Since a bucket name cannot already be in use, and a bucket name must only contain valid chars that adhere to domain naming conventions, a try/except statement is used when creating the bucket. If an error is caught, it's printed to the status label. Otherwise, the built-in *os.exec()* method is used to restart the application which in turn restarts the application and correctly displays the existing bucket names along with the newly created bucket name within the S3 bucket drop-down menu.

**get\_bucket\_contents()** - This method returns a list of all available file-names for a given bucket argument.

**populate\_dropdown()** - This method returns the entire list of buckets that are linked to the user's Amazon S3 account().

```
set_local_browse_label(incoming_string)
set_s3_bucket_label(incoming_string)
set_status_label(incoming_string)           ← set various labels found throughout the application
set_found_local_label(incoming_string)
set_found_s3_label(incoming_string)
```

## IMPLEMENTATION

### CONFIGURING YOUR MACHINE:

There are two Python files containing the entire code for the project. One is used for Linux, and the other is used for Windows.

#### Windows:

s3\_zilla\_WIN10.py – Windows 10 Version

Configuring the computer to run this code is fairly simple on Windows 10. The default Python 3 installation that comes with Windows also happens to come with Tkinter. The only other requirement is to install the Amazon S3 Boto3 module in order to use their API (The AWS SDK for Python, 1):

```
pip install boto3
```

#### Linux:

s3\_zilla.py – Linux Version (may also work with Mac Operating Systems – not thoroughly tested)

In order to get Tkinter running on Linux, the following command must be run:

```
sudo apt-get install python3-tk
```

Then the following libraries also must be installed for OpenSSL:

```
sudo apt-get install libssl-dev
```

Now similarly to the Windows installation, the Amazon S3 API module can be installed into the Python 3 interpreter via the Pypi manager:

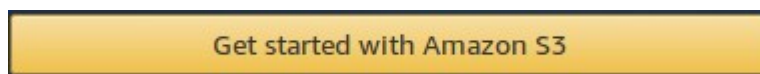
```
pip install boto3 (or pip3 depending on your configuration)
```

## CONFIGURING AMAZON S3 IAM USER:

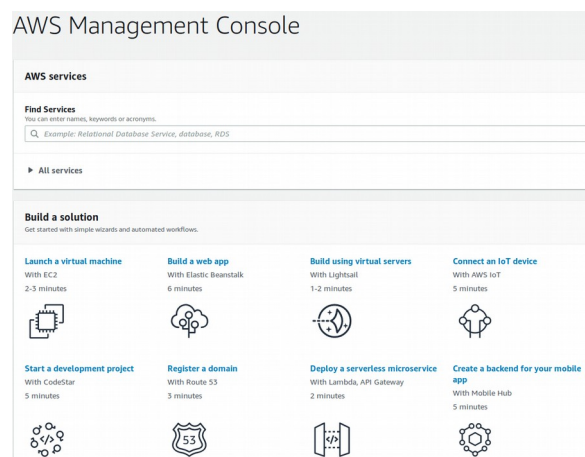
In order to get the code running, a valid Amazon S3 account must be set up. One of the best benefits of Amazon's EC2 and available cloud-computing infrastructure is that just about all of it can be tested for free. It costs nothing to set up an account. However your credit/debit card is needed to create the account. The "free tier" for Amazon S3 includes 5 Gigabytes of storage, 15 gigabytes of data transfer, up to 20,000 HTTP GET requests to your buckets, and up to 2,000 put requests to your buckets (Amazon S3 Pricing , 1).

This is actually a decent deal considering that it's on a monthly basis (1). This also allows plenty of room for a developer to practice with the technologies, learn the APIs, learn the management consoles, and develop the code without having to pay a cent.

There are several steps that are needed to complete the account set up and API-key configuration. First visit the URL <https://aws.amazon.com/s3/> and look for the "Get started with Amazon S3" button.



This will lead you through the typical online account setup process that is similar to other Amazon account initialization processes. A valid email address, phone number, address, and credit or debit card will be needed during this process. Once it's complete, a user will be greeted with a screen like this:





This is the basic console for Amazon Web Services. The options can be intimidating, and finding what you are looking for may be awkward at first. The “Find Services” text input box is extremely useful and will find any service that they have by keyword. If a user starts visiting this console along with some of the Amazon services on a regular basis, the services that are visited most often will be displayed and highlighted on the dashboard so that the user can find those particular services quickly and efficiently.

For this project, the only service that is needed is Amazon S3 as well as one other service named “IAM”. If you were to visit Amazon S3 at this point by finding it in the search box and clicking on it, several warnings and errors will be shown that may prevent you from using the Amazon S3 service.

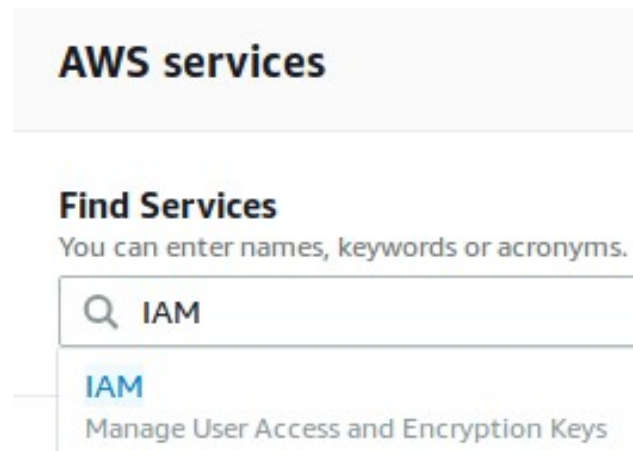
Amazon S3 will automatically issue a key to the root user once, and then automatically delete it during another setup process. While it’s okay to login as the root user, the root user should not have his or her credential’s stored anywhere. They should also not be used to make API requests from code at all. Their only purpose is related to the previously mentioned “IAM” service and it is to create new users with certain accesses.

The IAM service makes it quick to securely create and issue keys that may be used to access that particular S3 account’s resources. Permissions can be granted for various roles and tasks, and it’s managed centrally from the dashboard/console in a very convenient fashion.

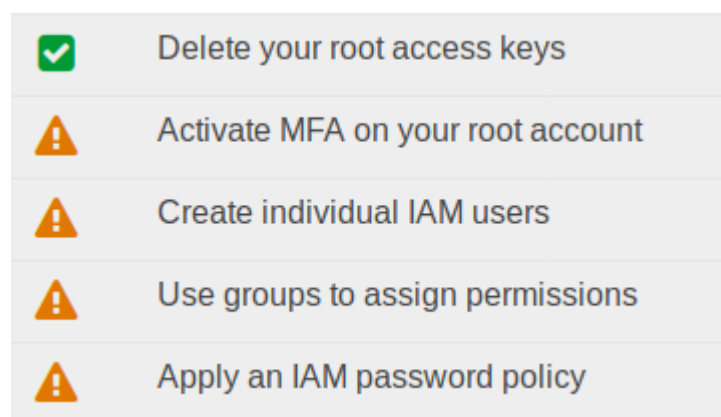
This is the main reason that some of the original code written for `s3_zilla.py` that encrypted the API credentials on a client machine was eventually dropped from later versions of the Project.

The IAM service can instantly revoke any API key that may be suspected of misuse, no longer has a purpose, or no longer belongs to a user (possibly the user left the organization etc). I felt strongly that the additional encryption code would obscure the overall project idea, especially because of additional software requirements that particular end-users may not want to install.

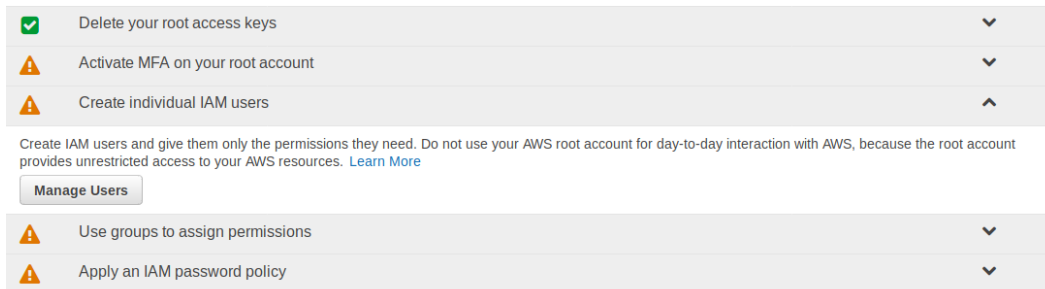
To set up an IAM user and start using the API, look for the IAM service in the Find Services search box:



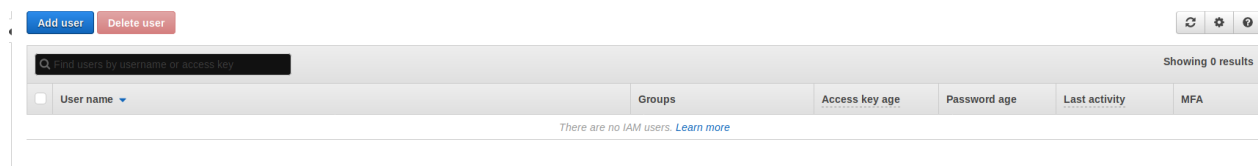
Upon opening the IAM console, several warning will be presented. Activate Multi-Factor Authentication isn't necessary to get the project and Python code working. The most important item is "Create individual IAM users". This is the section that allows new users to be added as well as permissions to govern which user can do what regarding the S3 account.



When the button is clicked, it reveals another button that should be clicked, “Manage Users”.



Once “Manage Users” is clicked, another console will appear. This is the main console that is used to create users and issue API keys/credentials.



Now the button “Add user” button can be pressed to bring up a form. The username must be entered and Access Type must be checked. Multiple users can be added under a similar key to control members as a group. Programmatic access for the Access type is necessary for users that wish to use the Amazon S3 AP I keys within their code. In this case, this is what is most important for the s3\_zilla.py to run. However in a real situation, the given users(s) may also be granted access to the actual AWS Management Console with a password. This central authority makes it easy to revoke and add at will.

### Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name\* jamescollely


[+ Add another user](#)

### 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.

After completing this, the next part is the “Use groups to assign permissions” section on the Security Status list of drop-downs.

 Use groups to assign permissions











Create group

Create a group and select the policies to be attached to the group. Using groups is a best-practice way to manage users' permissions by job functions, AWS service access, or your custom permissions. [Learn more](#)

Group name

Create policy Refresh

Filter policies  Showing 438 results

	Policy name	Type	Used as	Description
<input checked="" type="checkbox"/>	 AdministratorAccess	Job function	None	Provides full access to AWS services and resources.
<input type="checkbox"/>	 AlexaForBusinessDeviceSetup	AWS managed	None	Provide device setup access to AlexaForBusiness services
<input type="checkbox"/>	 AlexaForBusinessFullAccess	AWS managed	None	Grants full access to AlexaForBusiness resources and access to related AWS Services
<input type="checkbox"/>	 AlexaForBusinessGatewayExecution	AWS managed	None	Provide gateway execution access to AlexaForBusiness services
<input type="checkbox"/>	 AlexaForBusinessReadOnlyAccess	AWS managed	None	Provide read only access to AlexaForBusiness services
<input type="checkbox"/>	 AmazonAPIGatewayAdministrator	AWS managed	None	Provides full access to create/edit/delete APIs in Amazon API Gateway via the AWS Management Console.
<input type="checkbox"/>	 AmazonAPIGatewayInvokeFullAccess	AWS managed	None	Provides full access to invoke APIs in Amazon API Gateway.
<input type="checkbox"/>	 AmazonAPIGatewayPushToCloudWatchLogs	AWS managed	None	Allows API Gateway to push logs to user's account.
<input type="checkbox"/>	 AmazonAppStreamFullAccess	AWS managed	None	Provides full access to Amazon AppStream via the AWS Management Console.
<input type="checkbox"/>	 AmazonAppStreamReadOnlyAccess	AWS managed	None	Provides read only access to Amazon AppStream via the AWS Management Console.

Cancel Create group

This section is important because it allows access to many different groups, one of the most notable being “AdministratorAccess”. For the sake of demonstration, “AdministrativeAccess” is checked off. This may be dangerous as this group has access across ALL of the available web services in the dropdown list. Every service available is listed here, from ElasticBeanstalkFullAccess to DirectoryServiceFullAccess and RekognitionFullAccess. Many different roles exist for each; all are listed in the extensive dropdown menu. The best move is actually to search for s3 next to “Filter policies”. Several are shown but AmazonS3FullAccess is needed. This access along with the Administrative access are the only groups that can GET and POST/PUT to an account’s S3 buckets via the API keys:





Create group

Create a group and select the policies to be attached to the group. Using groups is a best-practice way to manage users' permissions by job functions, AWS service access, or your custom permissions. [Learn more](#)

Group name

Create policy Refresh

Filter policies  Showing 4 results

	Policy name	Type	Used as	Description
<input type="checkbox"/>	 AmazonDMSRedshiftS3Role	AWS managed	None	Provides access to manage S3 settings for Redshift endpoints for DMS.
<input checked="" type="checkbox"/>	 AmazonS3FullAccess	AWS managed	None	Provides full access to all buckets via the AWS Management Console.
<input type="checkbox"/>	 AmazonS3ReadOnlyAccess	AWS managed	None	Provides read only access to all buckets via the AWS Management Console.
<input type="checkbox"/>	 QuickSightAccessForS3StorageManagementAnalytics...	AWS managed	None	Policy used by QuickSight team to access customer data produced by S3 Storage Management Analytics.

Note that if a free account is being used, any user that performs and HTTP GET or POST/PUT counts towards the 20,000/2,000 free tier limits (I personally checked).

A confirmation that the user’s information is correct will be presented:

Add user

1

:

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name

jamescolley

AWS access type

Programmatic access - with an access key

Permissions boundary

Permissions boundary is not set

Permissions summary

The user shown above will be added to the following groups.


Type	Name
Group	<a href="#">admins</a>

Tags

No tags were added.


## CONFIGURING AMAZON S3 CREDENTIALS


Now the Amazon S3 Access Key ID and Secret Access Key must be configured on the user's machine. Upon successfully creating a user, the following screen is shown with several very important items:

 **Success**

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://090934745984.signin.aws.amazon.com/console>

 **Download .csv**

	User	Access key ID	Secret access key
▶	 jamescolley	AKIARKLBCDOAPBZSXTVN	***** <a href="#">Show</a>

Now when the user's name (jamescolley) is clicked on inside the IAM console, all of the user's permissions, groups, and security credentials can be managed (and made inactive as well). Complete control is offered from one central console:

Summary

User ARNarn:aws:iam::090934745984:user/jamescolley

Path /

Creation time2019-05-04 04:04 EDT

PermissionsGroups (1)TagsSecurity credentialsAccess Advisor

▼ Permissions policies (1 policy applied)

Add permissions

Policy name

Attached from group

▶ AdministratorAccess

▶ Permissions boundary (not set)

Summary

User ARNarn:aws:iam::090934745984:user/jamescolley

Path /

Creation time2019-05-04 04:04 EDT

PermissionsGroups (1)TagsSecurity credentialsAccess Advisor

Add user to groups

Group name

Attached permissions

adminsAdministratorAccess

User ARNarn:aws:iam::090934745984:user/jamescolley

Path /

Creation time2019-05-04 04:04 EDT

PermissionsGroups (1)TagsSecurity credentialsAccess Advisor

Sign-in credentials

Summary

• User does not have console management access

Console passwordDisabled | Manage

Assigned MFA deviceNot assigned | Manage

Signing certificatesNone

Access keys

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs. For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation. [Learn more](#)

Create access key

Access key ID	Created	Last used	Status
AKIARKLBCDOAPBZSXTVN	2019-05-04 04:04 EDT	N/A	Active   <a href="#">Make inactive</a>

14

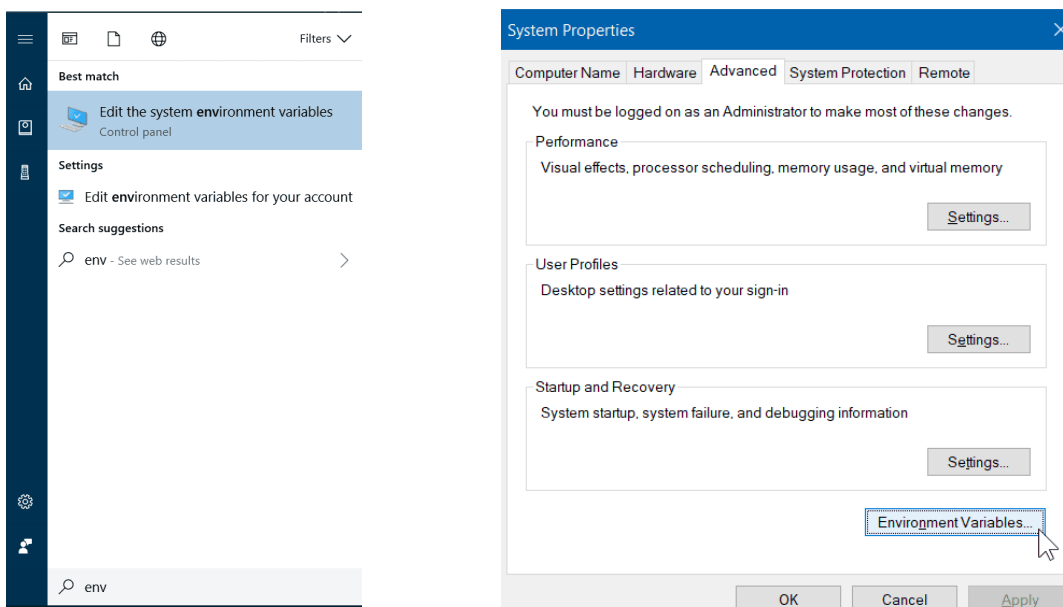
In the green colored “Success” page shown above, the Access key ID is shown for the newly created user “jamescolley”. The Access key ID is the equivalent of a username. While this example will use the environment variable method, here is the order that Amazon S3 looks for API credentials:

1. **Command line options** – You can specify `--region`, `--output`, and `--profile` as parameters on the command line.
2. **Environment variables** – You can store values in the environment variables: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`. If they are present, they are used.
3. **The CLI credentials file** – This is one of the files that is updated when you run the command `aws configure`. The file is located at `~/.aws/credentials` on Linux, macOS, or Unix, or at `C:\Users\USERNAME\.aws\credentials` on Windows. This file can contain the credential details for the default profile and any named profiles.
4. **The CLI configuration file** – This is another file that is updated when you run the command `aws configure`. The file is located at `~/.aws/config` on Linux, macOS, or Unix, or at `C:\Users\USERNAME\.aws\config` on Windows. This file contains the configuration settings for the default profile and any named profiles.
5. **Container credentials** – You can associate an IAM role with each of your Amazon Elastic Container Service (Amazon ECS) task definitions. Temporary credentials for that role are then available to that task's containers. For more information see [IAM Roles for Tasks](#) in the *Amazon Elastic Container Service Developer Guide*.
6. **Instance profile credentials** – You can associate an IAM role with each of your Amazon Elastic Compute Cloud (Amazon EC2) instances. Temporary credentials for that role are then available to code running in the instance. The credentials are delivered through the Amazon EC2 metadata service. For more information, see [IAM Roles for Amazon EC2](#) in the *Amazon EC2 User Guide for Linux Instances* and [Using Instance Profiles](#) in the *IAM User Guide*.

Using any text editor, the credentials should be added as environment variables. In this screenshot, they have been added to the `.bashrc` file located in the home directory of most users on a Linux machine. Ensure that the API keys are named **AWS\_ACCESS\_KEY\_ID** and **AWS\_SECRET\_ACCESS\_KEY**.

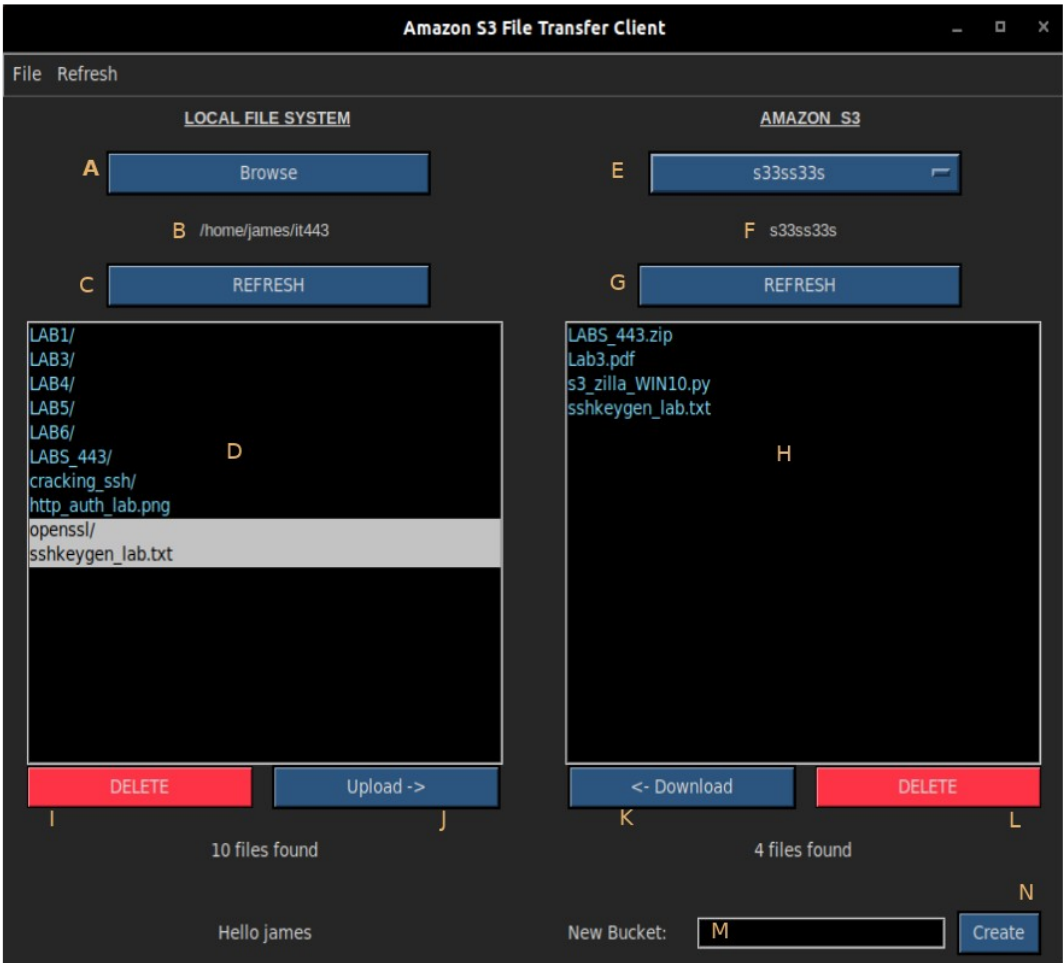
```
james@rootVIII:~$ cat .bashrc | grep -i aws
export AWS_ACCESS_KEY_ID=AKIARKLBCDOAPBZSXTV
export AWS_SECRET_ACCESS_KEY=icB0zPkfMDTqahv
```

If on a windows machine, use the search bar to search for “environment variables” and follow the instructions to add the two API keys.



# OPERATION

## GUI DIAGRAM



A	Browse Button for the local file system, select a directory	H	File explorer for the chosen S3 bucket, displays contents
B	Label that displays the local system's current working directory	I	Delete button for local file system, delete file or directory
C	Refresh button for local file system and working directory	J	Upload button-uploads to the selected S3 bucket
D	File explorer for the local file system, displays chosen directory	K	Download button-downloads to chosen working directory
E	Browse button for the S3 account, select an existing S3 bucket	L	Delete button for S3 bucket, deletes a chosen file from S3
F	Label that displays the S3 account's current working bucket	M	Text box for a new, unique bucket name
G	Refresh button for S3, refreshes chosen bucket's contents	N	Create button for creating a new Amazon S3 bucket



## S3Zilla OPERATION

- The “left half” of the application is intended to represent the local file system, while the “right half” of the GUI is intended to represent the Amazon S3 account.
- The **refresh buttons** are used to update the local file explorer or the chosen Amazon S3 bucket. Use the appropriate one after changing the working directory or working S3 bucket.
- The **local browse button** is used to select a working directory from local file system. Press the corresponding refresh button after selecting a working directory.
- The Amazon **S3 drop-down menu** is used to select an S3 bucket. Press the right-hand refresh button after selecting a new bucket. If no buckets are available, a note will display within the drop-down menu notifying the user that no buckets are available.
- The **upload button** uploads to the chosen Amazon S3 bucket while the **download button** downloads to the chosen local directory.
- **Delete buttons** may be used to delete local files and directories from a chosen directory or to delete files from a chosen S3 bucket. The left delete button works with the local file system and the right delete button deletes from the S3 buckets. **THIS ACTUALLY DELETES YOUR FILES/DIRECTORIES AND IS NOT REVERSIBLE!**
- One or multiple files may be selected within either file explorer for upload or download. Click on any file name and it will automatically highlight. Click again to deselect it.
- The file explorers will automatically update after uploading or downloading a file.
- A ‘/’ after a filename in the explorer denotes a directory.
- If a directory is selected within the local file explorer, it will automatically be converted to zip format prior to uploading to the chosen S3 bucket.
- The **create button** is used to create a new Amazon S3 bucket.
- A new bucket **MUST** have a unique name: If another user on the other side of the world has a bucket named “testbucket”, that name is no longer available for any other user (Amazon S3 Bucket Naming Requirements 1). The name must also conform to a similar naming convention as with hostnames and DNS. An S3 bucket may have between 3 and 63 characters, must start with a number or lowercase letter, and may not be formatted as an IP address (1). “The bucket name cannot contain underscores, end with a dash, have consecutive periods, or use dashes adjacent to periods” (1). Therefore when creating a new bucket, several attempts may be necessary.