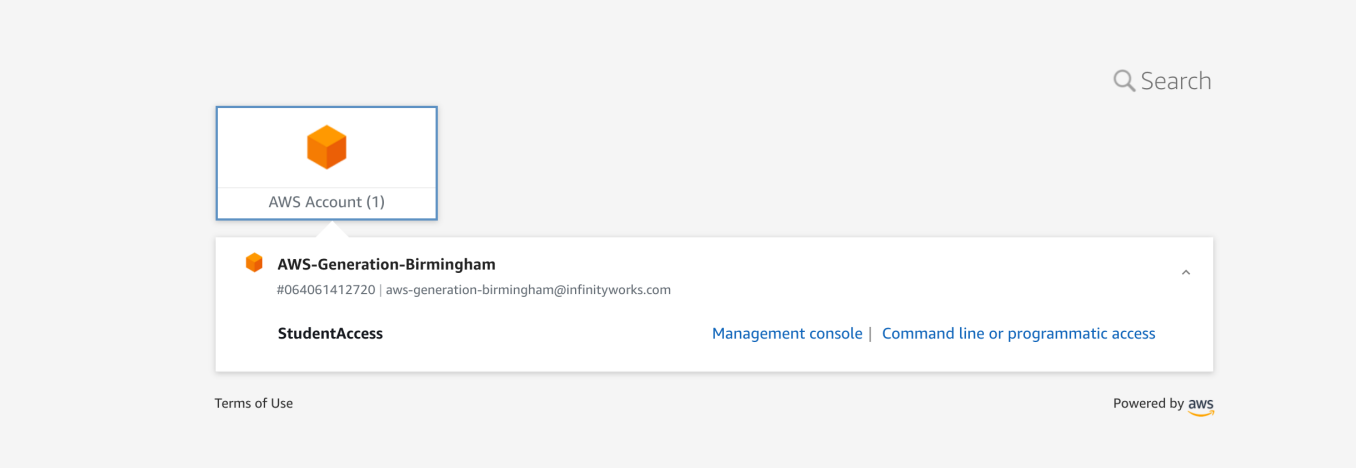
AWS Exercises

AWS Account Setup

1. Open your email client and find the email from no-reply@login.awsapps.com.
2. Click '*Accept Invitation*'.
3. Input a **strong** password which matches the ruleset (use a password manager if you can).
4. You will then need to register an MFA device with your account. Select the **Authenticator app** option and hit next.
5. Follow the steps to complete the MFA sign-up stage and click 'Assign MFA'.
6. You should get a message saying '*Authenticator app registered*'. Click '*Done*'.
7. Once completed, you will be redirected to a new screen. Click on the '*AWS Account (1)*' box so that it expands, then click on the wider box underneath. You should see a screen that looks like the below:



1. Click on '*Management console*' and you will be successfully signed into the AWS console.

AWS CLI Setup

Installation

Windows:

1. Download the latest version.
2. Open the download and follow the installation steps.
3. Verify your installation with the following command:

```
C:\> aws --version
aws-cli/2.1.24 Python/3.7.4 Windows/10 botocore/2.0.0
```

MacOS:

1. Download the latest version.
2. Open the download and follow the installation steps.

3. Verify your installation with the following commands:

```
$ which aws
/usr/local/bin/aws

$ aws --version
aws-cli/2.1.24 Python/3.7.4 Darwin/18.7.0 botocore/2.0.0
```

Linux:

Follow the guide which best matches your setup here.

Configuration

1. On a terminal, run `aws configure sso`.
2. Enter your SSO (*Single Sign-On*) URL (if you don't know it, your instructor will be able to tell you).
3. Enter the SSO Region, which should be `eu-west-1`.
4. A webpage will open asking you to sign into the AWS CLI, click the `*Sign in to AWS CLI*` button.
5. Looking back at your terminal, you will see some text which looks something like this:

   > Using the account ID *xxxxxxxxxxxx* The only role available to you is: StudentAccess Using the role name "StudentAccess"

6. When it asks for *CLI default client Region*, hit enter.
7. When it asks for *CLI default output format*, hit enter.
8. When it asks for *CLI profile name*, enter a reasonable name, such as *learner-profile*.
9. That's all (for now!).

You can log out of your SSO any time by running `aws sso logout`.

You can then login again by running `aws sso login --profile <profile-name>`.

EC2

You're going to setup your own EC2 server, then extend it so that we can host a basic website on it. After that we will look at how we can tighten security of our instances.

Security Group Setup

Before creating your own EC2 instance, you will need to create a security group. Security groups take control of the traffic that is allowed in and out of your instance. You can apply restrictions on port ranges and IP ranges. We will be restricting `SSH` access to your IP, but open `HTTP` to the world. This is bad practice, and so you would normally be much more restrictive in terms of what you allow in and out, but for the sake and simplicity of this exercise, we won't need to worry about that.

1. Go to `EC2` page by using the search bar. On the left-hand side under `Network & Security`, select `Security Groups` and then select `Create security group`.
2. Give your security group a unique name (e.g. `your-name-sg`) and a description (e.g. `My SG`).

3. Under `Inbound rules`, select `Add rule`. a. Rule 1: Select `SSH` for `Type` and `My IP` for `Source`. b. Rule 2: Select `HTTP` for `Type` and input `0.0.0.0/0` in the text field to the right of `Source`.
4. Under `Outbound rules`, select `All TCP` as the type and input `0.0.0.0/0` in the text field to the right of `Destination`.
5. Select `Create security group` to finish.

EC2 Instance Setup

Now let's set an instance up.

1. Go to EC2 and select `Launch Instance`.
2. Step 1 - Select `Amazon Linux 2 AMI (HVM), SSD Volume Type`.
3. Step 2 - Nothing to do, hit next.
4. Step 3 - a. For `Subnet`, select the option which contains the string `PublicSubnet1A`. b. For `Auto-assign Public IP`, select `Enable`.
5. Step 4 - Nothing to do, hit next.
6. Step 5 - Nothing to do, hit next.
7. Step 6 - a. Choose `Select an existing security group` and select the security group you created. b. Select `Review and Launch` and then `Launch`.
8. In the pop-up, select `Create a new key pair` and give it a name (something like `my_name_key`.
9. Download the key and launch the instance.
10. Navigate to `Instances` and select the `Instance ID` value of your instance.
11. Wait for your instance to have an instance state of `Running` before moving on. This should only take about 15-30 seconds.

Accessing the Instance

Your instance has now been spun up and is ready to be accessed. Let's see how we can go about getting inside it.

1. On your instance summary page, select `Connect` in the top-right of the webpage.
2. Select the `SSH Client` tab and copy the long `ssh` command under `Example:`.

Now follow the below steps on your terminal (use `git bash` if on Windows).

1. In the folder your downloaded key file is in, run: `chmod 400 {name-of-key}.pem`.
2. Paste the `ssh` command you copied and hit enter.
3. You will be asked `Are you sure you want to continue connecting (yes/no/[fingerprint])?`, type yes and hit enter.
4. You should now be logged in!

Setting up the website

1. Elevate your privileges: `sudo su`.
2. Update all of the packages on the instance: `yum update -y`.
3. Install an apache webserver: `yum install httpd -y`.
4. Change directory to /var/www/html with `cd /var/www/html`.
5. Run `nano index.html`, copy/paste the contents of the `index.html` handout and save the file. To copy the contents you need to open the index.html file with a text editor to get the html from it.

6. Start the webserver: `service httpd start`.
7. Configure the web server to restart if it gets stopped: `chkconfig on`.
8. Copy the IP address of your instance, you can find it under `Public IPv4 address` on the instance page in AWS.
9. Paste the address into your browser and watch the magic happen. Hope you like it. ☺

Note: If you can't hit it and you are using Chrome (or similar), it will try to default to `https:xxx`. This won't work, so change the URL to `http:xxx` instead.

Extending our security

The problem with our current setup is that we're relying on having a key file on our machine. Think about the below:

- What if we lose the key?
- What if the key is leaked online?
- What if someone at a company leaves, how do we safely transfer the key?
- What if multiple people want to access the EC2 instance, how would they safely distribute the key?

Is there a way we can login to our instance without needing to worry about keys? We can with a tool called `SSM Agent`.

`SSM Agent` is Amazon software that can be installed and configured on an EC2 instance. This will remove the need for us to use a key to access our instance. This works by configuring and ensuring the correct people are accessing it with IAM policies and roles. It also means we can close off port 22 inbound access so we increase security even further.

1. SSH into your instance if you exited from it.
2. Enable `SSM Agent` with `sudo systemctl enable amazon-ssm-agent`.
3. Start `SSM Agent` with `sudo systemctl start amazon-ssm-agent`.
4. You can confirm it is running by running `sudo systemctl status amazon-ssm-agent`.
5. Exit your instance by running `exit`

If you want to use the AWS CLI to start and end sessions that connect you to your managed instances, you must first install the Session Manager plugin on your local machine.

1. Follow the installation guide for Windows or MacOS.
2. Navigate back to the security group you created on the AWS console and select `Edit inbound rules`.
3. Delete the rule for SSH (port 22) and save rules.
4. Navigate to the EC2 instances page on the AWS console and tick the checkbox for your instance.
5. Select `Actions --> Security --> Modify IAM role` in the top right corner.
6. Click on the dropdown and select `ssm-ec2-role` as the role and select `Save`. This is a role that gives SSM certain permissions to perform actions for the instance on its behalf.

Now let's try and access our instance again, but in a different way this time.

1. Copy your EC2 instance ID.
2. Run the command `aws ssm start-session --target [instance_id] --profile [name_of_profile] --region eu-west-1`

You should now be logged back into your instance, but this time we let AWS do all the legwork for applying security restrictions as opposed to putting that on the user. We also no longer need to use an SSH key to access it.

Wrapping up

When you are done with this part of the exercise, please delete the following:

1. Any EC2 instances you created.
2. Any security groups you created.
3. The `.pem` file you downloaded.

AWS S3

S3 has a wide range of features, it isn't *just* for storing objects.

Part - 1 Dealing with files

Use the AWS CLI to perform the following actions:

1. Create two S3 buckets (bucket names are **globally** unique, so if someone in the world already created a bucket called `Test123`, you will not be able to use it).
2. Upload a handful of files of your choosing to the first one (make sure there is no personal information in them).
3. *List* your buckets.
4. *List* the files in your first bucket.
5. *Copy* a file from the first to second bucket.
6. *Move* a file from the first to second bucket.
7. *Delete* a file from either bucket.

Hint: Here is a link to the S3 CLI docs.

Part 2 - Setup a website in S3

In this exercise we will be leveraging S3 in a slightly more unusual way. S3 is a fantastic tool for hosting static websites. On a static website, individual webpages include static content. They might also contain client-side scripts.

A lot of websites are becoming static websites which means they run zero server side code and consist of only HTML, CSS and JavaScript. With no server side code to run, there is no reason to host them on a traditional server.

By using the static website hosting feature on an S3 bucket, we can host static websites for one to two dollars a month and scale to handle millions of users. So let's try it out!

Setting up the bucket

1. Navigate to the S3 console and click `Create bucket`.
2. Enter a name for the bucket (bucket names are globally unique, so if someone has taken your name, tough!).

3. Scroll to the bottom and hit `Create bucket` again.
4. Navigate to `Permissions --> Block public access (bucket settings) --> Edit`.
5. Untick `Block all public access` and save.
6. Navigate to the `Properties` tab.
7. Scroll to the bottom of the page until you see `Static website hosting` and click `Edit`.
8. Select `Enable` for `Static website hosting`. Input `index.html` for `Index document` and `error.html` for `Error document`.
9. Select `Save changes`.

Setting up the static website

Now we need to upload both `index.html` and `error.html` to our bucket. These files will have been provided to you.

1. On the `Objects` tab, select `Upload`.
2. Select `Add files` and select the two files.
3. Under `Additional upload options --> Access control list (ACL)`, select `Read` for `Everyone`. You will get a warning. Tick off the checkbox which says `I understand the effects of these changes on the specified objects.`.
4. Select `Upload`.

Everything should be set up at this point, so navigate to your static website URL (this can be found under `Properties --> Static website hosting --> Bucket website endpoint`). You should be able to both access the page and see the HTML content rendered on the page.

Showing the error page

Sometimes we want to display a custom error page when something goes wrong, follow these steps to see it:

1. In your bucket, click on the `index.html` file and select the `Permissions tab`.
2. Select the `Edit` button and untick the `Read` checkbox for `Everyone`.
3. You can now see your error page!

Simply revert the above action to display your normal webpage.

Important: This mini-exercise was to demonstrate some of the features S3 has to offer. However, you would normally never *just* use S3 to host a website as there are a plethora are security vulnerabilities. You can use another service that sits in front of your bucket, called Cloudfront, which provides services such as a CDN, caching, DNS and traffic distribution, including security protections on the bucket itself.

AWS Lambda

Setup

1. Navigate to `Lambda` on AWS.
2. Create a new function and select `Author from scratch`.
3. Give your function a unique name and select `Python 3.8` as the runtime and create the function.

4. Expand `Change default execution role` and select `Use an existing role`. Select `lambda-execution-role`.
5. Select `Create function`. You will be directed to your new lambda.

Executing some code

1. Navigate to the `Code source` window on the page. Double-click on the `lambda_function.py` file to open the file for editing.
2. Insert the below code and update the string to include your name. Watch out for formatting when you paste it. Click the deploy button.

```
import json

def lambda_handler(event, context):
  return {
    'body': json.dumps('Hello [insert name]!')
  }
```

1. Click the `Test` button, this will display a pop-up. Select the `hello-world` template and give the event a name. Click `Create`.
2. Click `Test` again to run the lambda, you should see a response with your name in the result window below!

```
{
  "body": "\"Hello [insert name]!\""
}
```

Interacting with other AWS services

We've managed to create a basic lambda function. Let's try something more exciting to show the capabilities of lambda. We'll update the code we just deployed. We will use the lambda to upload a file we've created locally to the S3 bucket you created earlier.

1. Go back to your lambda and paste in the code below. Make sure to update the bucket name to one you've created.
2. Deploy the lambda and test it.
3. Navigate back to your S3 bucket, you should now see a file called `hello.txt` in there!

```
import json
import boto3 # library used to access AWS API
import os

def lambda_handler(event, context):
  os.chdir('/tmp')
  bucket = '[insert-bucket-name-here]'
  filename = 'hello.txt'
```

```
    client = boto3.client('s3')

    response = client.put_object(
      Body="Hello [insert name]!",
      Bucket=bucket,
      Key=filename
    )
```

What happened? We created a lambda with basic permissions. The permissions applied came bundled with the role we applied to it. You can view the policy attached to the role to get an idea of what is happening. The policy has an action of `s3:PutObject` which means the lambda has permission to call the `PutObject` operation for `S3`. This allows the lambda to upload objects to an S3 bucket. The policy is very broad as it allows the lambda to put an object into *any* bucket in our account. We would normally restrict access to a specific bucket, or set of buckets. For this exercise though, it won't be a problem.

If we removed that permission, we would get an error when the lambda attempts to put the object when calling the AWS API via `boto3`.

As lambdas are essentially containers under the hood (think back to Docker). It comes with an underlying file system where we can perform operations such as `os.chdir('/tmp')`. The `tmp` folder allows us to create temporary files and hold them before the lambda terminates. Once terminated, the file(s) will be lost.

We created an AWS client with the `boto3` library, which can be used for virtually any AWS service. In our case, we used S3. The S3 API contains a multitude of functions that can be called to interact with S3.

Wrapping Up

- Delete any Lambda(s) you created for this session.
- Delete all objects inside the S3 bucket(s) you created, then delete the bucket(s).