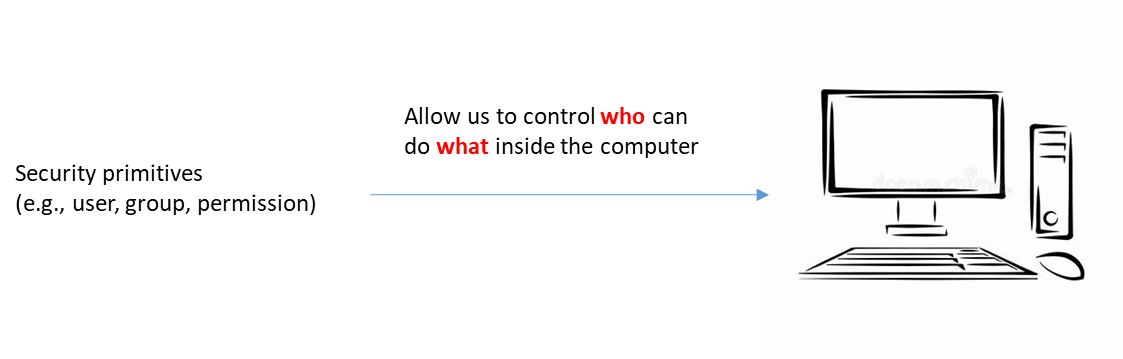
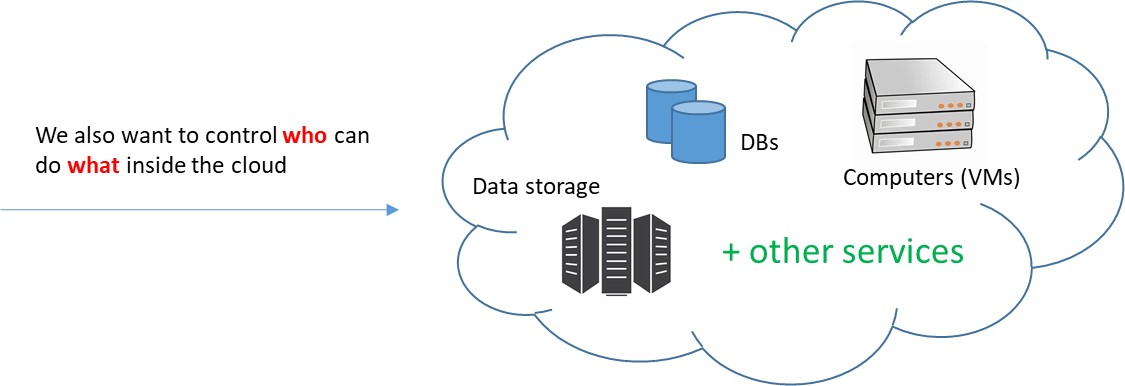
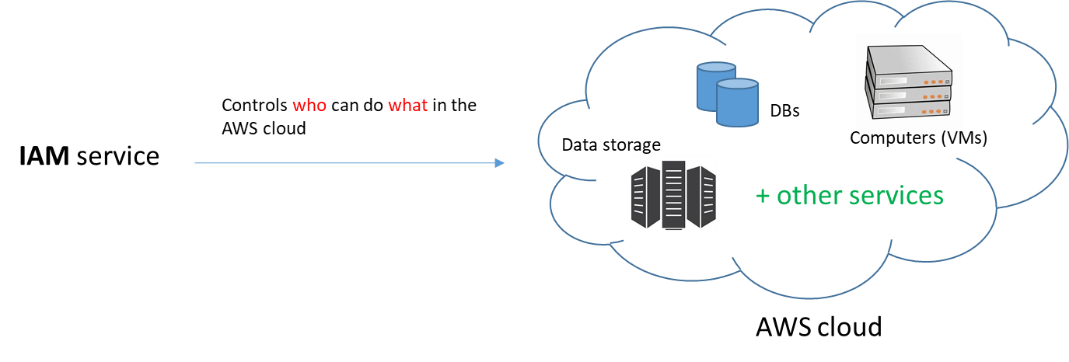
# Module 4 Identity and Access Management

## Overview

A regular computer comes with built-in security primitives that allow managing access to the computer (Figure 1). For example, in Windows or Linux we can think of these security primitives as things like user, group, and permission. A user is identified by a username/password. And permissions control access to files (e.g., read, write, execute). Groups are named collections of users.   
  
  
**Figure 1**: A collection of security primitives are used to control *who* can do *what* on a personal computer.

We can think of the cloud as a giant computer with many things typically found in a personal computer and much more (e.g., file storage to store files, computing capability to run programs, databases to save and search data, etc.). The cloud is a collection of resources (e.g., computers (VMs), databases, queues, storage, etc.), and we also want to control who can do what in the cloud (Figure 2).  
  
  
  
**Figure 2**: The cloud is a collection of computing resources. Just like a personal PC, we also want to control *who* can do *what* in the cloud.

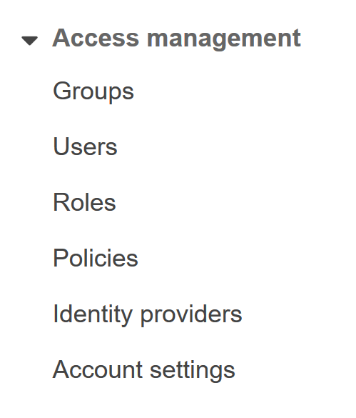
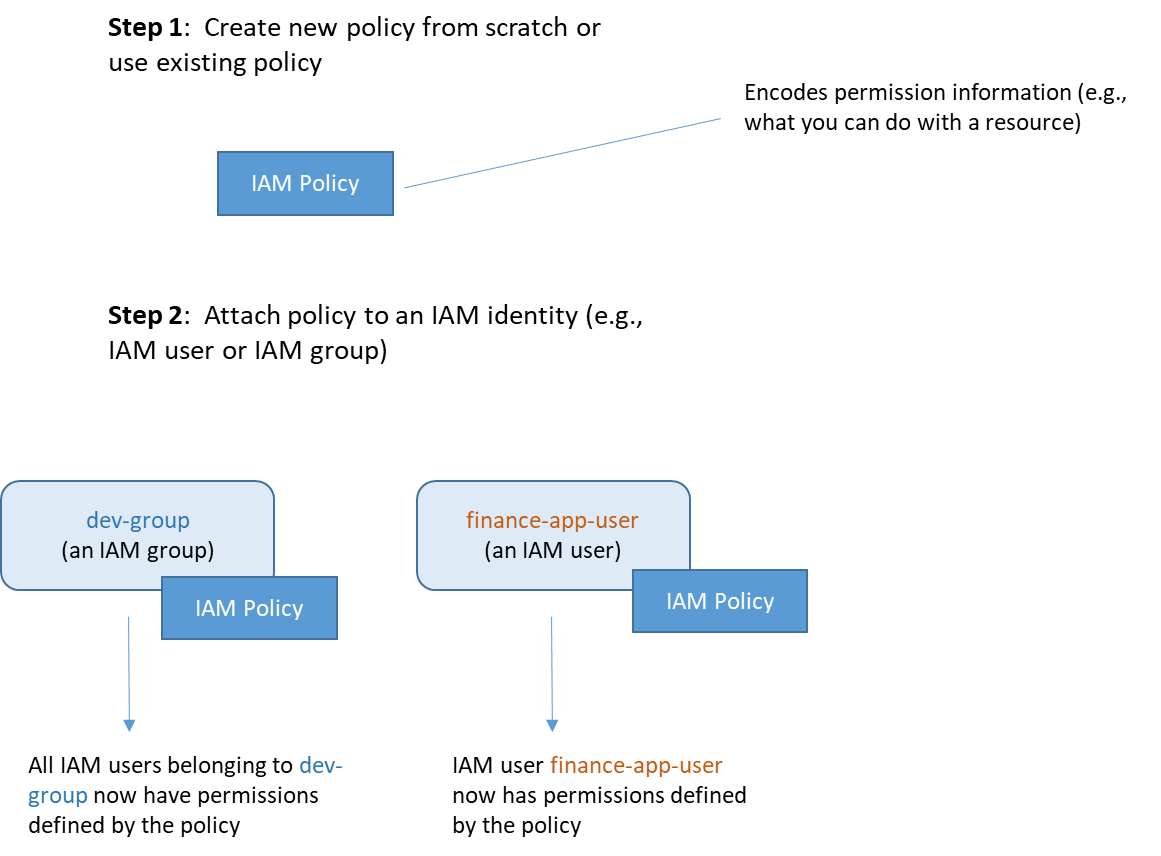
The AWS service that provides security primitives to control access to the AWS cloud is called **IAM** (**Identity and Access Management**). We can refine the image of Figure 2 to make it look like Figure 3:  
  
  
**Figure 3**: The IAM service is a collection of security primitives used to control who can do what in the AWS cloud.

One important characteristic of the cloud that does not have an equivalent in a personal computer is the concept of **billing**. On a personal computer, the user doesn’t incur extra charges for additional uses of the PC resources. That is, there is no extra charge for storing more files on a hard disk or simultaneously running 20 programs instead of just 5. The cloud differs in this characteristic; because the cloud operates like a utility-based service, additional uses of its resources translates to additional charges the cloud provider charges its customers.

When an individual or company first creates an account with AWS (not you since you are an IAM user which we will discuss shortly), that account is known as a **root account**. A root account has complete and unrestricted access to all resources in AWS. It has no restrictions and its privileges cannot be reduced. Furthermore, the root account is tied to billing – usually a credit card. If a malicious user gets hold of this account, he/she can spin thousands of VM machines and say use them to mine bitcoins, charging your card tens of thousands of dollars a day. Because this root account is so powerful (and somewhat dangerous), it is not recommended that it be used for day-to-day activities. We will see shortly how an IAM user solves this problem.  
  
IAM comes with security primitives to allow centralized and granular control over the AWS cloud resources. Many of the IAM security primitives are already familiar to you (user, group, policy, role, etc.). We will look at these in a moment. But first, let’s quickly take a glance at IAM in the AWS console.

* Login to the AWS Academy canvas site, and go to the “AWS Academy Learner Lab - Foundation Services” class.
* Click the **Modules** link.
* Click **Learner Lab – Foundational Services**.
* Start the lab. After the lab starts, click the AWS link to navigate to the AWS Console.  
    
  (The steps above might be slightly different than what I listed above – I login as an educator and what I see is slightly different from your view).

This AWS Console page is the central web page you use to navigate to many AWS services, look at your account, etc.

* Type IAM in the Search Textbox at the top. IAM should come into view.
* Click it. This takes you to the IAM service page.  
    
  You should now be on the IAM service page. Notice in the bar on the left side some of the security primitives that you can use to control access to AWS resources. Another name often used for any of these is **identity** (that’s where Identity comes in the service name). So an IAM user is an identity, an IAM group is also an identy, so is an IAM Role, etc.  
    
     
  Let’s use a fictitious company scenario to illustrate the meaning of the above primitives and how we can use them. Consider a company called SoftwareFactory with many types of employees: software developers, software QA engineers, and software managers. We know the following:  
    
  a) - All charges resulting from the use of the AWS cloud (by any type of employee or application at SoftwareFactory) are paid by SoftwareFactory. In other words, we desire that all charges be centralized under one account, owned by SoftwareFactory. This account is the root account of the SoftwareFactory company.  
    
  b) - Different employee types at SoftwareFactory (e.g., developers, QA engineers, and managers) play different roles and, therefore, require different access privileges to cloud resources. For example, a software developer may need to create a cloud database as part of an application she is working on. The manager, on the other hand, does not need to create a database because she is not doing any programming work.  
    
  c) - SoftwareFactory is hosting a finance application on an Amazon VM. And this finance application needs access to S3 storage (e.g., it needs to download files from S3). For now don’t worry about what S3 is – just think of it as a cloud file storage drive.  
    
  In such a scenario, one way we can organize access control by SoftwareFactory to the AWS cloud is by imagining a hierarchy of primitives that looks like Figure 4 (where the item in black is the root user, items in blue are IAM groups, and items in orange are IAM users):  
    
    
    
    
    
  softwarefactory-root-user   
   dev-group   
   john   
   jennifer   
   sally   
   qa-group  
   tom  
   matt  
   managers-group  
   dawn  
   bill  
    
  **Figure 4**: Hierarchy of IAM groups and IAM users under the root account softwarefactory-root-user of SoftwareFactory.  
    
  By having a root user (softwarefactory-root-user) as the root account and owner of all IAM users, we automatically satisfy condition (a) above – all charges of SoftwareFactory users and applications are consolidated under one account. In this way we won’t need for example to have Jennifer or Tom create their own root accounts and give their credit cards to AWS. All charges incurred by Jennifer, Tom, and all others, will be handled by softwarefactory-root-user (tied to a single credit card owned by SoftwareFactory). This way billing got really simplified - If SoftwareFactory wants to know how much it is spending on AWS per month, it can look at its credit card statement. No need to ask each user on their spending and sum the total.   
    
  IAM Groups and IAM Users  
    
  The concepts of **IAM group** and **IAM user** are identical to the group/user in an operating system: groups are logical groupings of users. It simplifies management because we can associate privileges with a group and have all users in that group automatically inherit those privileges. This is more straightforward than having to assign the same privileges for every single developer at SoftwareFactory. If a new developer, Peter, is hired, we simply add Peter to the IAM dev-group group (see Figure 4). Now Peter automatically gets all the privileges that other developers in dev-group have. Furthermore, the privileges assigned to a group are almost always more restrictive than those of the root user. This ensures that people or application have just enough security needed for their work – and nothing more.   
    
  So far every IAM user under the 3 IAM groups in Figure 4 is a real person. However, an IAM user does not necessarily need to map to a real human. We don’t have an example above that illustrates this but it is perfectly fine to have an IAM user that does not map to a person.   
    
  An IAM user, just like the root user, can login to the AWS console or interact with AWS using the AWS SDKs.   
    
    
  IAM Policies  
    
  Now that we outlined the hierarchy of the SoftwareFactory groups and users we need to give them permissions to do things in the AWS cloud. In AWS, the security primitive used to give permissions to identities such IAM users, IAM groups, and IAM roles to access resources in the AWS cloud is known as an **IAM policy**. Note that we haven’t discussed IAM roles yet - we will do that shortly.  
    
  You create an IAM policy and you attach that policy to an IAM identity such as IAM user, an IAM group, or an IAM role (Figure 5). These types of policies are known as **Identity-based policies**.   
    
    
    
  **Figure 5**: Using an IAM policy to give IAM identities permissions over AWS resources. The policy grant permissions to an identity.  
    
    
  In AWS, an IAM policy is stored as a JSON document. For example, below is a policy copied frAWS documentation:  
   

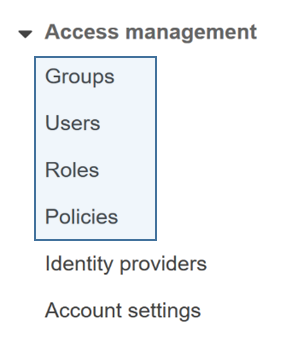
Policies can also be applied directly to a resource (not to an identity). An example resource is an S3 bucket or a database. These types of policies are known as **Resource-based policies**.  
Let’s try to understand the third statement above (the one with sid “ThirdStatement”. Let’s first look at the “Action” key, which is a JSON array with two items: s3.List\* and s3.Get\*. These represent any S3 action with a name that starts with List or Get (the \* is a wild card).  
  
Go to [this page](https://docs.aws.amazon.com/AmazonS3/latest/API/API_Operations.html) that lists S3 actions. Skim through the action names and notice that many start with Get or List. So s3.List\* means any S3 action that starts with the word List.  
  
The “Effect” key has the value “Allow”, which is self-explanatory. It means “allow all s3.Get\* and s3.List\* actions).

The “Resource” key is a JSON array with two items. It lists the resources that “Effect” and “Action” apply to. In this case the names of two S3 buckets. Notice that these names start with **arn** (acronym for Amazon Resource Name or ARN). An ARN is a unique identifier of an AWS resource. Every resource you create in AWS has a globally unique ARN that identifies it. For example, if you create a Lambda function, it will have its globally unique ARN, etc. ARNs follow a specific naming format that you can read about [here](https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html#arns-syntax). ARNs are created by AWS (not you the user). However, it can sometimes be beneficial to decipher a few characteristics of a resource from its naming parts.

**Should I invest time and energy mastering the syntax of writing IAM policies?**  
  
The answer is NO for 3 reasons: (1) Policies are things you don’t typically need to create on a daily basis. (2) There is a visual editor in the AWS Console that allows you to construct a policy. (3) You can investigate the needed syntax on as-needed-basis.   
  
**Is it essential to understand what a policy means when I read it?**  
  
The answer here is YES. The good news is that it is easy to do so as we demonstrated above when we analyzed the “ThirdStatement” statement.

If we now consider again the phrase “who can do what”, we can say: identities such as IAM User and IAM Group represent the “who”, and IAM Policy represents the “what”.

IAM Roles

We discussed the IAM user and IAM group identities. An IAM role is also an identity and you can attach a policy to it (in the same way we attach a policy to an IAM user or IAM group – look at Figure 5 again). But what is an IAM role and what is it useful for?   
  
An IAM role is somewhat similar to an IAM user in the sense that you attach policies to it which controls what it can and cannot do in AWS. However, one major difference is that a role does not have long term credentials (password or access keys) associated with it. What conceptually this means is that a role is not associated with a particular person. Instead, it is something that anyone can assume. For example, an IAM user can assume a role to temporarily take on different permissions so it can accomplish a specific task.  
  
Let’s use requirement (c) above to illustrate the use of an IAM role: “SoftwareFactory is hosting a finance application on an Amazon VM that accesses S3”. One possible way to handle this is that we can create an IAM user (a non-human IAM user) and attach to it a policy that gives permissions to S3. We can then embed this IAM user’s credentials in the application. This is not recommended. A better way is to create an IAM role that you attach to the VM to give applications running on this VM access to S3. Read this [article](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html) for more information.  
  
When you go to IAM, you should now have a good understanding of the primitives (users, groups, roles, and policies) used to control access to AWS cloud resources:  
  


Two fundamental advantages of IAM are:  
  
a) It is centralized: You manage users, groups, roles, policies in one place. You do not have to go to each service (e.g., EC2, S3) to manage users that use these services.  
  
b) It is fine-grained: For example, in the “ThirdStatement” statement we analyzed above, the policy allow s3:Get\* and s3:List\* actions only even though s3 has much more actions than that (this is what we mean by fine-grained).

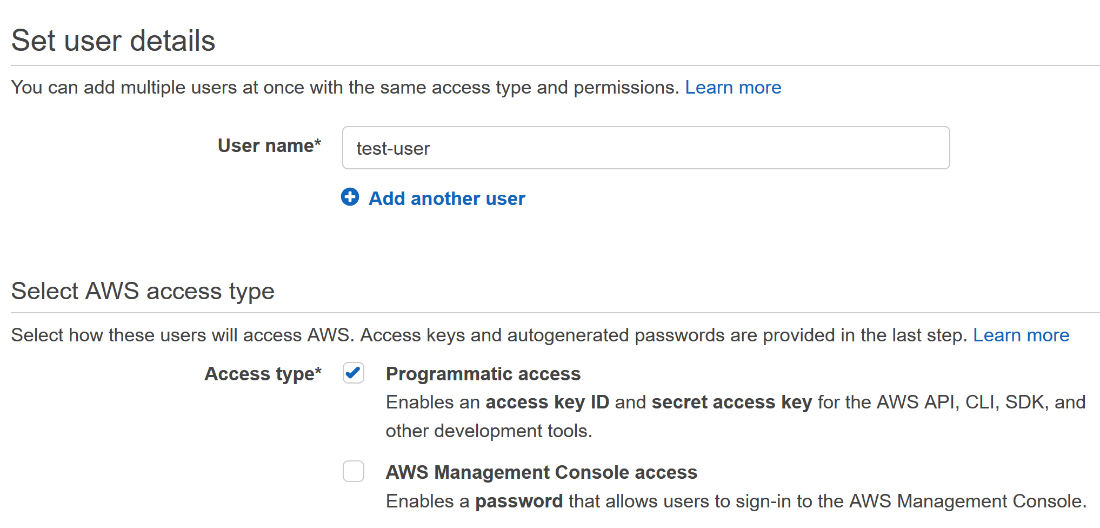
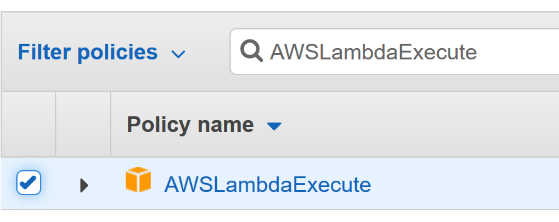
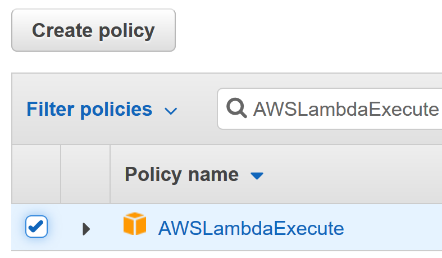
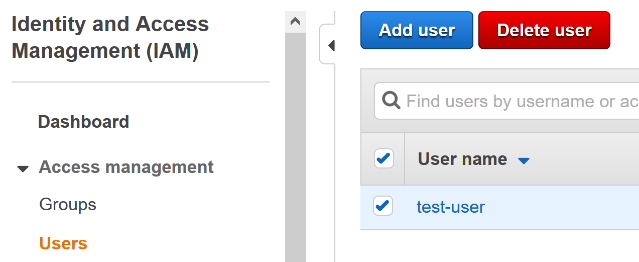
Note that you do not get charged for using IAM. IAM is free and you can create many users, groups, roles, policies, etc. There is no charge for doing that.

NOTE: this applies to this module and future modules.  
  
Modules are not intended to cover every minute detail of a concept. There isn’t enough time in a quarter to cover all services and everything of every service (e.g., IAM’s User Guide by itself is 500+ pages long).  
  
The idea is to introduce you to the basic concepts of a service. You need to do your own research if you need to use other advanced features. This is part of the needed research and investigation needed to accomplish any project.

**Principle of Least Privilege**  
  
AWS follows and recommends following the principle of [**Least Privilege**](https://en.wikipedia.org/wiki/Principle_of_least_privilege) in security. The principle of Least Privilege means you give access to the minimum needed to accomplish something and no more.

## 

## Exercises to complete on Your Own

1. Login to the AWS console and go to the IAM service.  
     
   If you see an error toward the top it is most likely you don’t have the permissions to do something the UI is trying to display (ignore it).
2. Most likely you don’t have any users and groups (and you should see 0s as shown below):  
     
   
3. On the left navigation column, click on **Users**.
4. Click the **Add User** button and fill in the form (something similar to the below):  
     
     
     
   (Note how in Access type there are 2 checkboxes. Remember that AWS can be used not only from the Management Console but also programmatically (via CLI and API – we will learn how to do that in future modules). So here I am saying that I want this “test-user” IAM user to be able to access resources via the API and CLI (and not via the AWS Console).
5. Click the **Next: Permissions** button.
6. Click the **Attach existing policies directly** button. This should show you some existing policies.
7. In the Filter policies Textbox enter some policy you want to give this user. For example enter AWSLambdaExecute.
8. AWSLambdaExecute should come into view. Check the checkbox to its left.  
     
   
9. Notice the button Create policy. If you don’t want to use an existing policy you can create your own policy by clicking this button. We won’t create a policy now. So don’t click it.  
     
   
10. Click the **Next: Tags** button.
11. Leave the Add tags screen empty (this allows you to enter some metadata about the user – can be useful when programmatically manipulating users). For now we don’t need any metadata. Click the **Next: Review** button.
12. Click the **Create user** button. This create the user – you might see a warning/error. Ignore it for now (the IAM User that AWS Educate gave you does not have a policy associated with it to do a specific task).
13. Click the Close button to dismiss the Success screen
14. Now you should be back to the main dashboard of IAM and you should see that you have an IAM user called test-user.  
      
    
15. You can now Sign out of the AWS Management console.
16. Answer the following questions. You need to submit these at the end of the module.

**What to Submit:  
  
Nothing to submit for this module and it doesn’t have a quiz associated with it.**