Lecture 02

Building Modern App on AWS

Part 01 - AWS CLIs and APIs

SENG 4620 - Practical Cloud Computing

TRU

Sina Keshvadi

Introduction to Building Modern Applications

Part 01 - AWS CLIs and APIs

Note:

Building brand new applications on Cloud is a different task than lifting and shifting existing applications into Cloud.

We are going to, from scratch, build
a serverless backend using AWS cloud-native tools and services.

In modern cloud-native application development,
it's oftentimes the goal to build out serverless architectures
that are scalable, are highly available, and are fully managed.

This means less operational overhead for you and your business,
and more focusing on the applications and business-specific projects
that differentiate you in your marketplace.

This means we will **not** be covering Amazon EC2 or AWS Container services here (we will cover them later).

Instead we are going to, from scratch,
build a serverless backend
using AWS cloud-native tools and services.

It's commonplace to choose to build API-driven applications.

We first explore how to build an API-driven application using Amazon API Gateway for serverless API hosting.

Then we will add authentication to the API using Amazon Cognito.

From there, we will add a Lambda backend
that will be triggered by API Gateway.

AWS Lambda for serverless compute

Amazon Cognito for serverless authentication.

Some of the features of our API
will require multiple Lambda functions
to execute in a specific order, like a workflow.

And we will be using AWS Step Functions to create a serverless workflow.

As you can see, we aren't going to be talking
about front-end application development much in this course.

We are mostly focusing on the backend aspect of this application,
where we'll be standing up an API that a client or front-end can then consume.

In the first 6 lectures, we'll be covering how to build a modern greenfield serverless backend on AWS.

All right, let's dive in.

Main Application

Demo Code

The demo code you will need for this lecture can be found in Moodle alongside this lecture.

Please download it now as you will refer to it later in the lecture.

Throughout this course,
we're going to be building a backend API
to demonstrate how to use a variety of AWS services
and show how they integrate and work together.

Dragons.json

```
{
        "description_str":"From the northern
fire tribe, Atlas  was born from the ashes
of his fallen father in combat. He is
fearless and does not fear battle.",
        "dragon_name_str":"Atlas",
        "family_str":"red",
        "location_city_str":"anchorage",
        "location_country_str":"usa",
        "location_neighborhood_str":"w
fireweed ln",
         "location_state_str":"alaska"
      }
```

So we have this data in JSON
and people want to query the data and consume it.

We are going to design a system to make that possible.

Now, for various security and design reasons
you generally don't want to expose your data directly to your consumers.

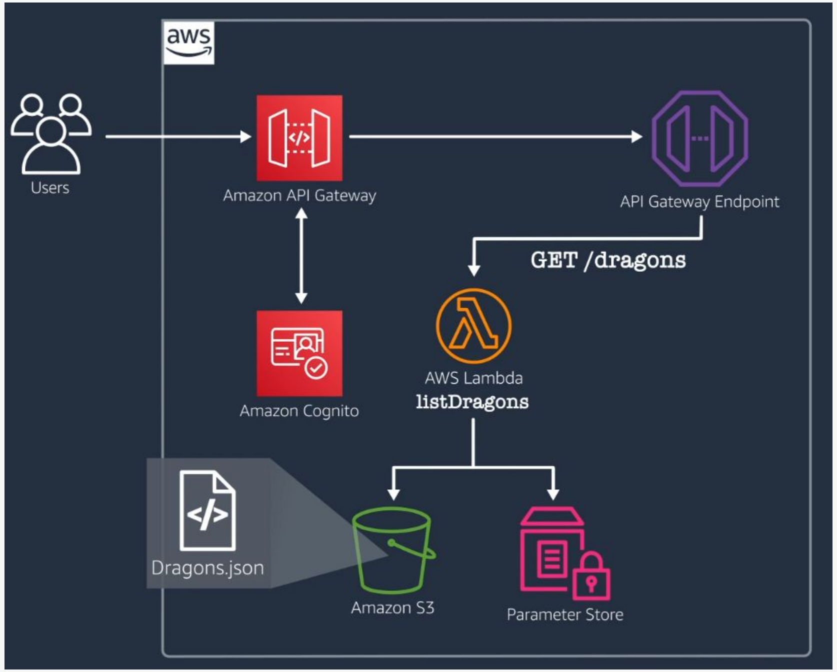
Instead, you want to expose your data to your consumers via
an API, or application programming interface.

We are going to build out an API using

Amazon API Gateway to expose that data securely.

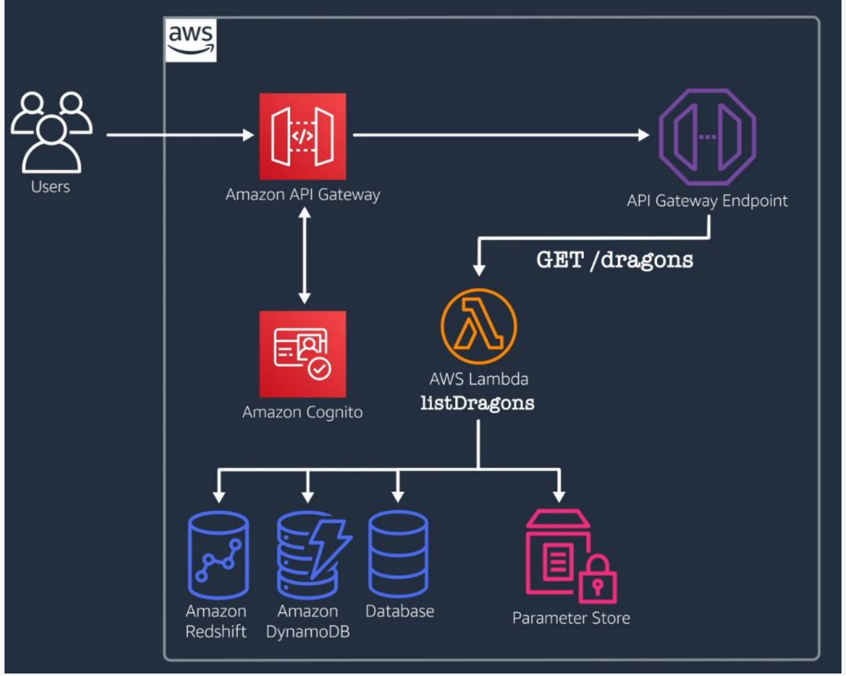
Knowing that our data is being stored in S3
and we will be using API Gateway as a front door to our backend,
we will create an end point for these `/dragons` resource in API Gateway.

You will be able to get a list of all dragons
using the GET HTTP method on the dragons resource.



API Gateway will accept the requests,
perform authorization using **Amazon Cognito**, validate the payload,
and if all that passes, it will invoke the backend to query the dragon data.

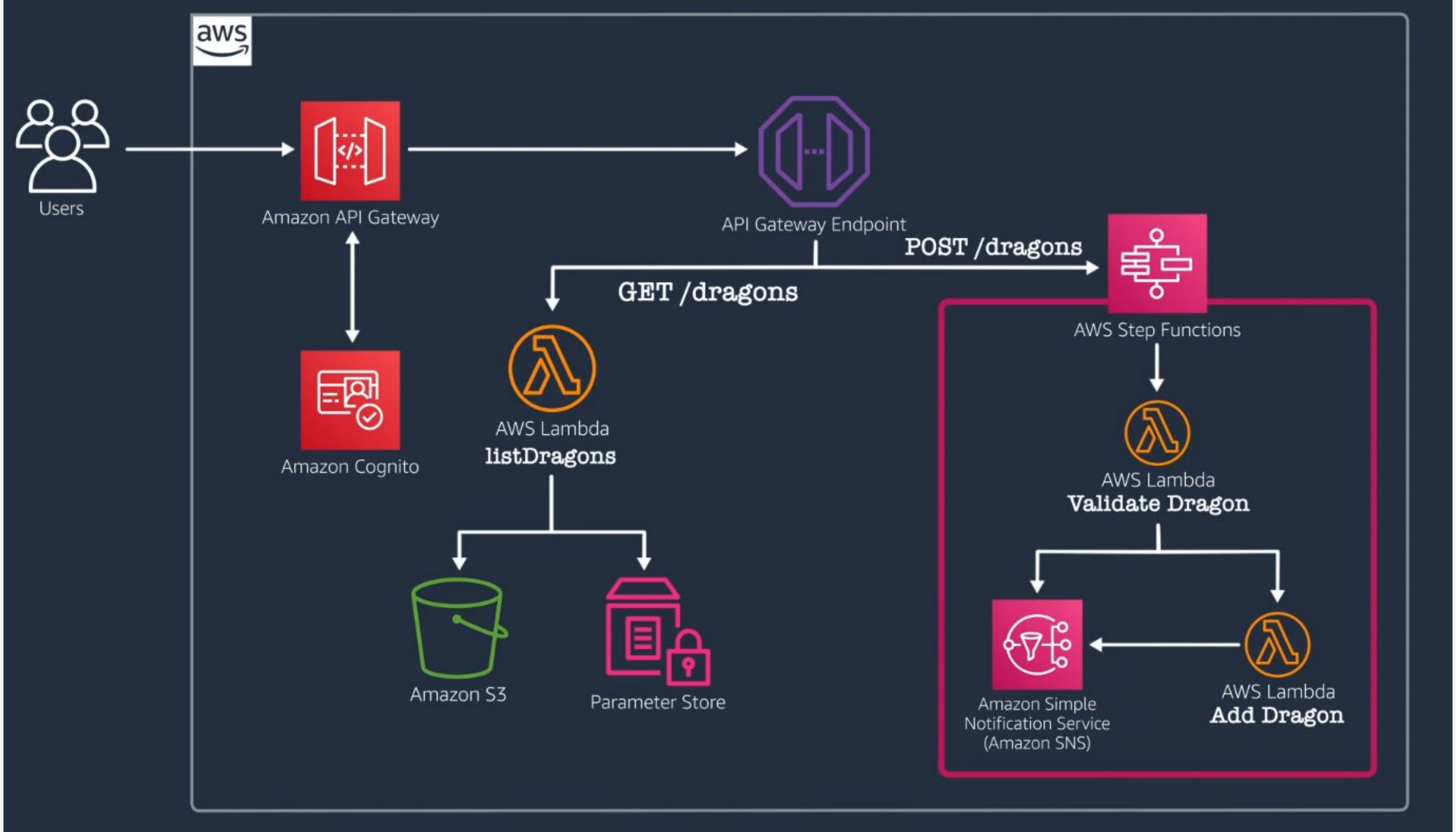
That backend for the GET request will be hosted by **AWS Lambda**.
The Lambda function will be reading the JSON data file in S3 using the API for S3 Select.



In other situations, you might be using a database to host this data, or maybe aggregating data from multiple places.

But for us, we will just be using S3 though it could always be converted to using a database later on if you wish.

The other feature our API supports
is we will allow people to report new dragon sightings.


Users
Amazon API Gateway
Amazon Cognito
API Gateway Endpoint
GET /dragons
POST /dragons
AWS Lambda
listDragons
Amazon S3
Parameter Store
AWS Step Functions
AWS Lambda
Validate Dragon
Amazon Simple
Notification Service
(Amazon SNS)
AWS Lambda
Add Dragon

The way users will interact with this feature is by sending a **POST HTTP request** to the /dragons resource in API Gateway.

For the report dragon feature, we will be using AWS Step Functions, AWS Lambda and Amazon Simple Notification Service, or SNS, to create an asynchronous dragon reporting system.

This reporting process is asynchronous
because we need to do some backend data validation that could take some time
and we don't want people waiting on an immediate response.

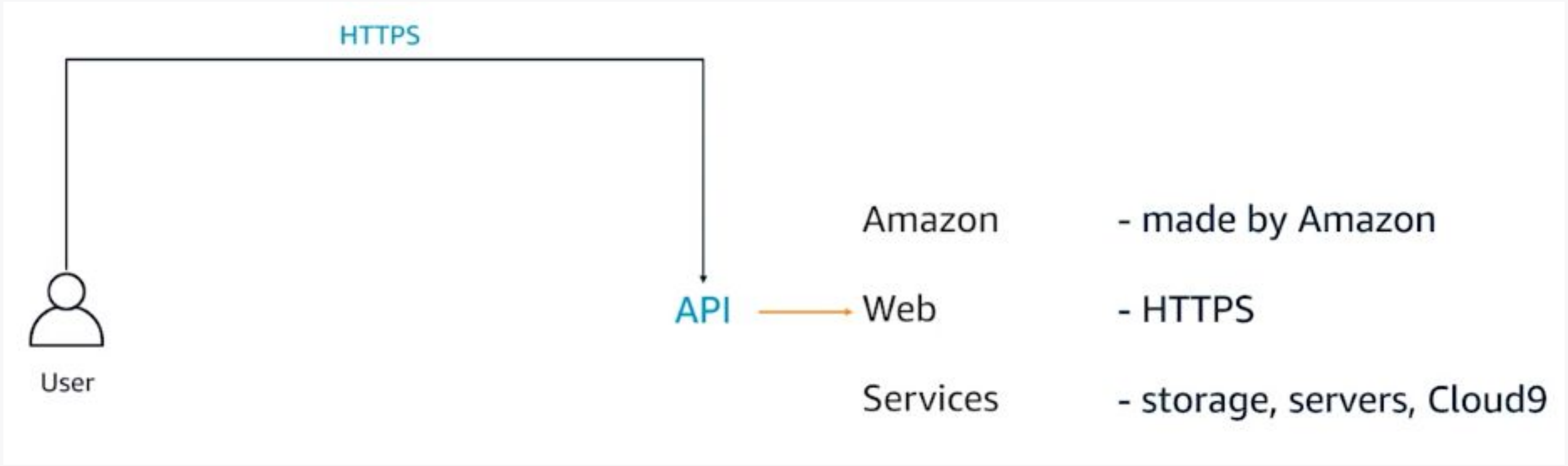
Instead, we alert the user when the process has completed.

This is a good example to show you how to set up a workflow:

1) a request hits your API
2) your backend would do whatever work needs to be done
3) and return the response when it's ready via SNS.

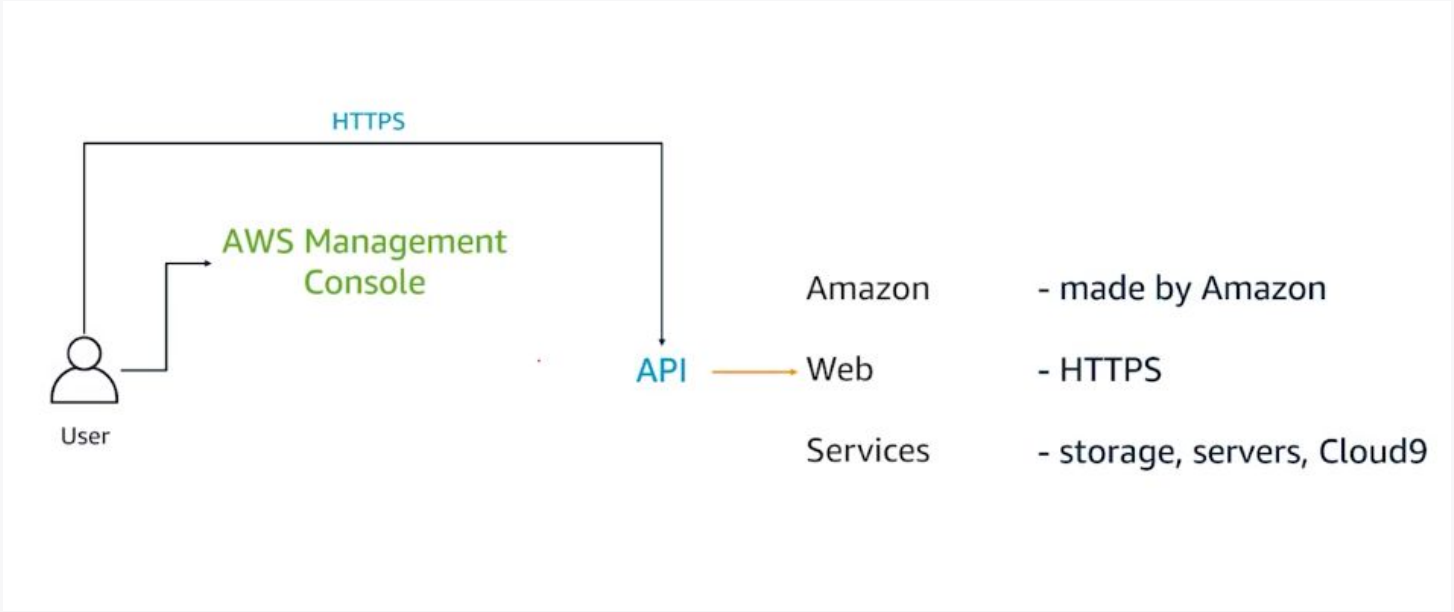
This is known as "**request offloading**" and it allows you to respond to your client without the entire process having finished yet.

Introduction to AWS API Management Console CLI SDK



Each of AWS services have an API that is made available to you.

AWS APIs are the entry point of AWS and to interact with them, and a user has to make an HTTPS request to that API.



AWS Management Console interacts with the AWS APIs directly.

This graphical user interface sounds easy enough
to work with when you, as a user, want to interact with it.

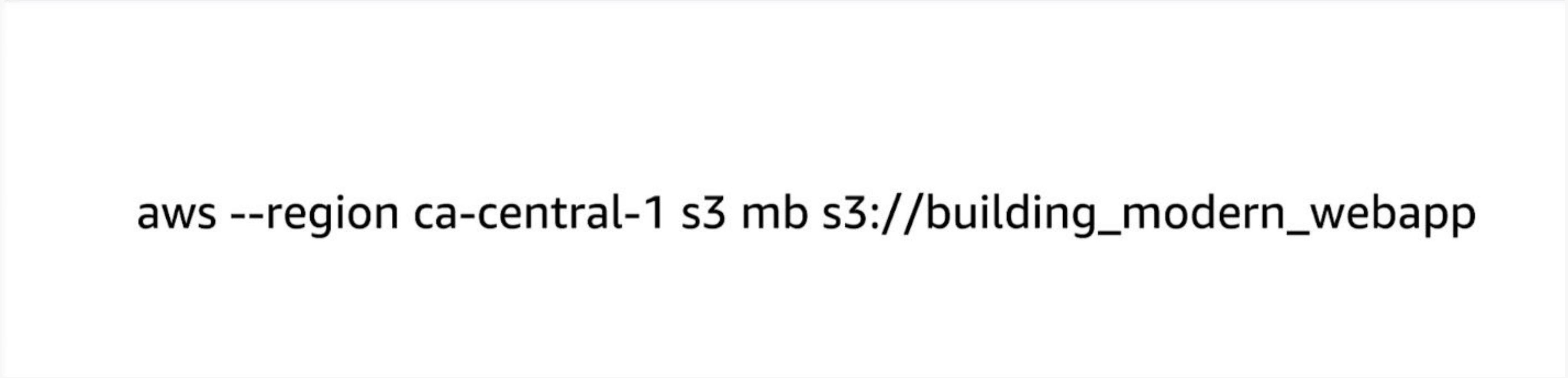
However, it becomes harder to work with when you want to repeat a task.

For example, let's say you wanted to create a few buckets in different regions.

Now, you will have to change your region, click on 'create bucket' again, fill everything like you did before, and really remember everything that you've done and then not mess up in between all of those tries.

The better option here is for you to use the AWS Command Line Interface, or CLI

which is like the AWS Management Console, it uses the API of AWS behind the scenes.



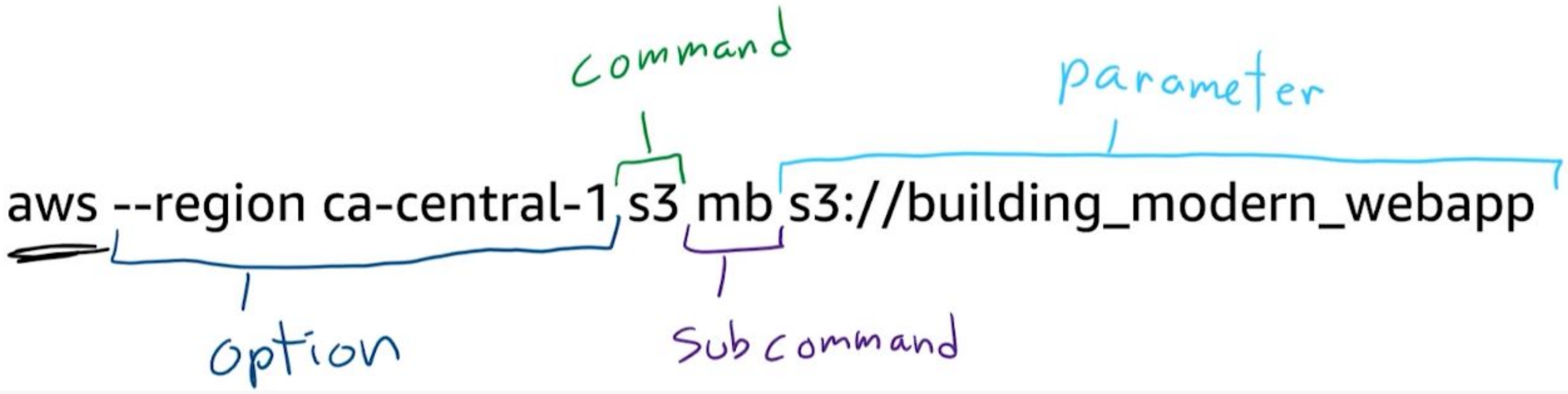
```
aws --region ca-central-1 s3 mb s3://building_modern_webapp
```

`- -region ca-central-1` (option)

`S3` is the command

`mb` is the sub command (mb stands for make bucket)

`S3://building-modern-webapp` is the parameter



```
aws --region ca-central-1 s3 mb s3://building_modern_webapp
```

Creating an S3 Bucket

```
aws s3 mb s3://<bucket-name>

aws s3 mb s3://my-new-bucket
```

Removing an S3 Bucket

```
aws s3 rm --recursive s3://<bucket-name>
```

```
aws s3 rm --recursive s3://my-old-bucket
```

The --recursive flag is necessary to delete
all objects within the bucket before deleting the bucket itself.

`aws s3 cp`: Copies files between S3 buckets or local storage.

`aws s3 ls`: Lists objects in an S3 bucket.

`aws s3 sync`: Synchronizes files between local storage and an S3 bucket.

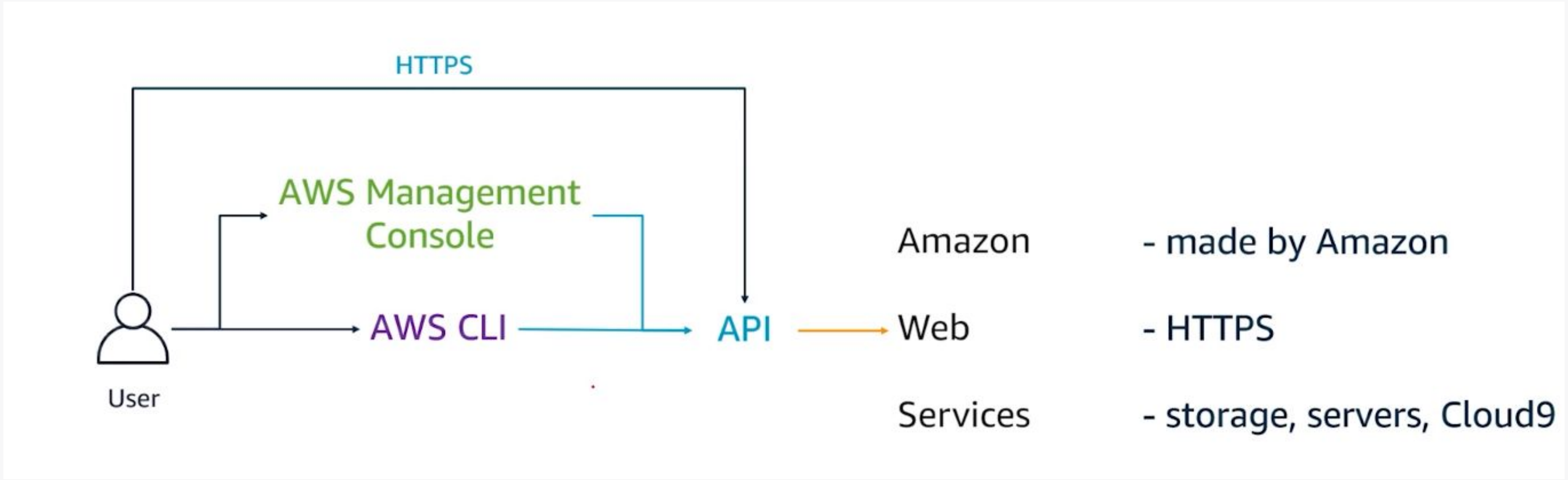
EC2 commands:

`aws ec2 run-instances`: Launches new EC2 instances.

`aws ec2 describe-instances`: Lists information about your EC2 instances.

`aws ec2 terminate-instances`: Terminates EC2 instances.

AWS CLI Command Reference



At this point, we saw three ways to interact with AWS:

the Management Console, the AWS CLI, as well as one that you probably shouldn't be directly be interacting with, which is the API itself.

Recap:

There are four ways to interact with AWS:

1. Directly use the API (not recommended)
2. Management Console, for when you start
3. CLI, for repeatable tasks
4. SDK for your code to interact with AWS.

All of those uses the API of AWS to reach the services of AWS.

Serverless Application Model

The AWS Serverless Application Model, or SAM,
is an open source framework for building serverless applications.

It provides shorthand syntax to
express functions, APIs, databases, and more
giving you the ability to define an application you want to model,
using a few short lines per resource.

During deployment, SAM transforms and expands the SAM syntax into AWS CloudFormation syntax, enabling you to build serverless applications faster.

A serverless application in this case is a combination of AWS Lambda functions, event resources, and other resources that work together to perform your distributed tasks.

SAM consists of two main components:

- AWS SAM template specification
- AWS SAM Command Line Interface

The SAM template specification is
used to define the serverless application.

It provides you with the syntax to describe everything
that makes up your serverless application.

The SAM Command Line Interface, or SAM CLI,
is the tool used to build the applications that are defined by the SAM templates.

This provides the commands that enable you to verify that
template files are written according to specification, work with resources,
package and deploy the application to the AWS cloud, and so on.

Keep in mind that this is a different CLI from the traditionally used AWS CLI,
as this is specifically used with SAM.

It needs to be installed and configured separately in order to run.

SAM is an extension of CloudFormation.

Because of that, you get the same deployment capabilities,
you can define resources, use your CloudFormation in your SAM template,
and you can use the full suite of resources, intrinsic functions,
and other template features that are available in CloudFormation.

Because SAM integrates with other AWS services,
there are a lot of benefits that are provided.

Lab 1

The labs for this serie of lectures are build out the Dragon APIs.

You will complete the labs in your own AWS account.

Lab 1: Setting Up and Exploring the SDK

https://aws-tc-largeobjects.s3.amazonaws.com/DEV-AWS-MO-BuildingRedux/exercise-1-exploring.html

Here some notes about this lab
that you might find useful.

```
$ echo "Please enter a bucket name: "; read bucket; export MYBUCKET=$bucket
```

- `echo` prints the message "Please enter a bucket name: " to the console.
- `read bucket;` reads input from the user and assigns it to a variable named bucket.
- `export MYBUCKET=$bucket` exports the value stored in the bucket variable to an environment variable named MYBUCKET. This means that MYBUCKET will be available for other commands or scripts to use.

```
$ aws s3 mb s3://$MYBUCKET
```

- `aws` is the AWS Command Line Interface, a tool that allows you to interact with various AWS services from the command line.
- `s3` specifies that you want to interact with the Simple Storage Service (S3).
- `mb` stands for "make bucket". It's a command used to create a new S3 bucket.
- `s3://$MYBUCKET` is a URL-like notation used by AWS to refer to S3 buckets. $MYBUCKET is a variable that holds the name of the bucket. This variable is assumed to be previously defined or set in your environment.

```
echo "export MYBUCKET=$MYBUCKET" >> ~/.bashrc
```

- The command is adding a line to the .bashrc file that sets the environment variable MYBUCKET to its current value. This can be useful if you want to persistently keep the MYBUCKET environment variable across shell sessions.

```
echo "export MYBUCKET=$MYBUCKET" >> ~/.bashrc
```

- `echo "export MYBUCKET=$MYBUCKET` prints the string export MYBUCKET=$MYBUCKET
- `>> ~/.bashrc` appends the output of the previous echo command to the file ~/.bashrc. The >> operator is used for appending text to a file.
- `~` is a shorthand notation for the user's home directory
- `~/.bashrc` is the file that is being modified. It's a hidden file in the user's home directory that is read by the Bash shell when it starts. It's typically used for setting environment variables and customizing the behavior of the shell.