

Aws is serverless.

The data stored in S3 is a record of Dragons.

Json file contain all info about drangons.

One record of Dragon example:

```
{
  "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"
}
```

We provide this data through API, using amazon API gateway not querying directly from S3 bucket.

Create one Amazon API gateway as front door to our backend

Create one Amazon API Endpoint as endpoint

In Endpoint we can get all dragons by GET http method from /dragons.

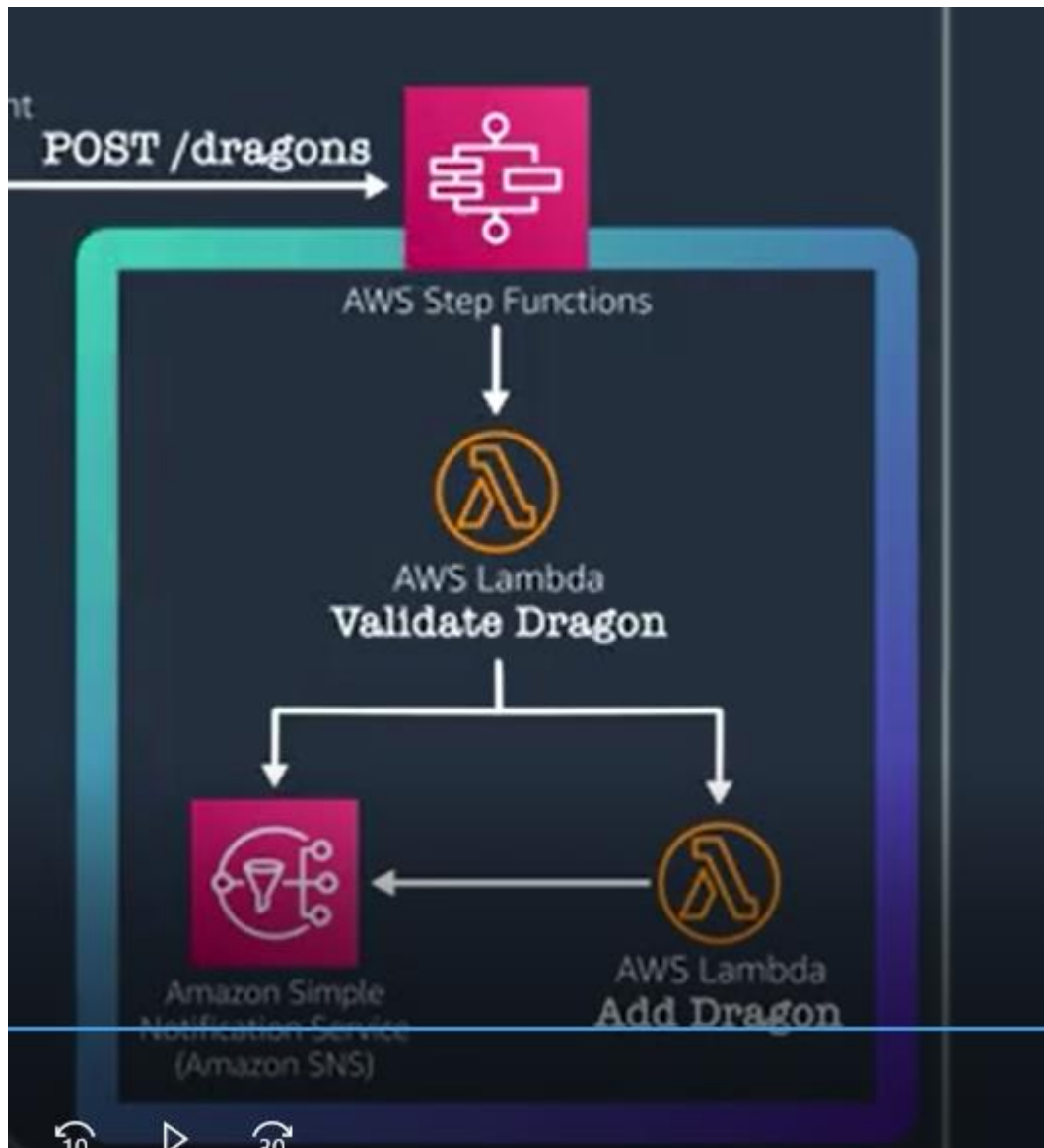
Users send requests to Amazon Gateway API, Amazon Gateway API does the authorization using Amazon Cognito, if it passed, it query the data from backend, by using the API Gateway Endpoint, the backend will be hosted by AWS Lambda, AWS Lambda read the data using the API for S3 select,

Note: instead of S3 we can use the other databases as well, or aggregation of these databases.

POST /Dragons

To use the AWS Step Function, AWS Lambda and AWS Simple Notification Service(Amazon SNS) > which creates the asynchronous Dragon reporting system.

Note that this process is asynchronous since we need validation in backend, when it finished, we alert the user which it is completed. This called request offloading.



AWS Cloud9 is Cloud based IDE, includes code editor, terminal and debugger.

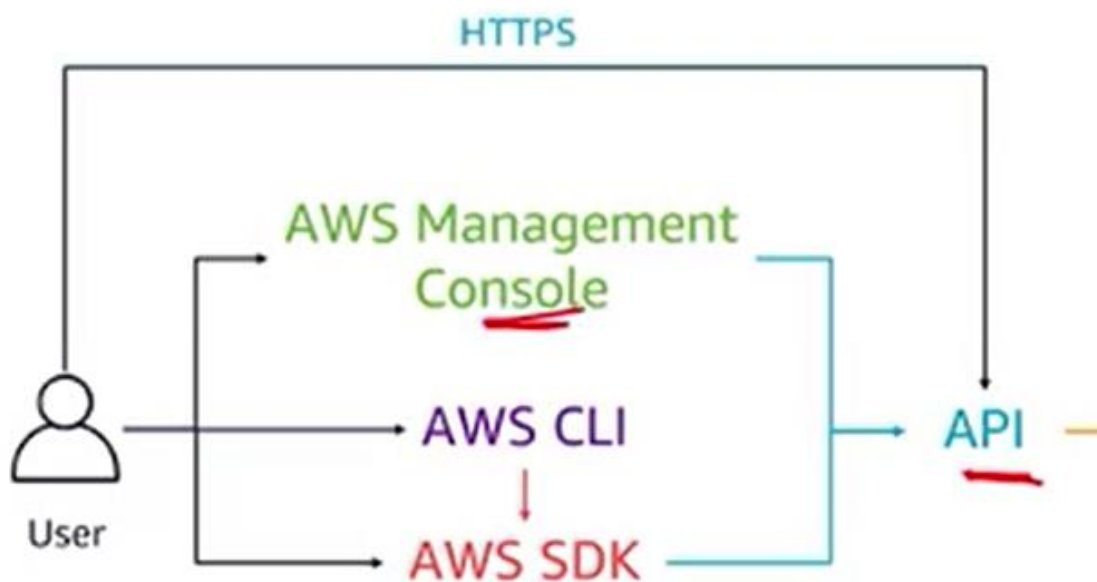
	Amazon	- made by Amazon
API	— Web	- HTTPS
	Services	- storage, servers, Cloud9

AWS command:

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

Handwritten annotations for the command above:

- `aws`: Command
- `--region ca-central-1`: Option
- `s3`: Sub command
- `mb s3://building_modern_webapp`: Parameter



AWS CLI is good for repeated tasks

AWS SDK is to use the code to interact with AWS.

To work with AWS CLI, we need to configure it first:

AWS configure

And give the credentials which are stored in ~/.aws/credentials

The region and output are stored in AWS config path or folder by default.

Default output format: default is json

We can use YAML, text,

If you append --profile to AWS configure command> we can create one profile with specific keys, default region and output.

For example for test and production levels we create two users, one example to access the production user:

Example command: `aws ec2 describe-instances --profile prod-user`

AWS SDK for python: named Boto3:

Setup Boto3 using cloud9:

Pip install boto3

In boto3 we can use the **client** or **resource** to interact with AWS API.

Client: low level interface to AWS whose methods map one-to-one with the service API. Return responses as python dictionaries.

Resource: higher level interface, and it is object oriented in nature, it is a wrapper around low level API (client), Exposes a subset of AWS APIs not all, it is easier.

Once u have a client or resource, u can invoke the method on that object which then call AWS API, example: listing all objects on s3 bucket as a client and then resource:

Client:



```

1 import boto3
2
3 client = boto3.client('s3')
4
5 response = client.list_objects(Bucket='aws-bmo-test')
6
7 for content in response['Contents']:
8     obj_dict = client.get_object(Bucket='aws-bmo-test', Key=content['Key'])
9     print(content['Key'], obj_dict['LastModified'])
10

```

Resource:



```

1 import boto3
2
3 s3 = boto3.resource('s3')
4 bucket = s3.Bucket('aws-bma-test')
5 # the objects are available as a collection on the bucket object
6 for obj in bucket.objects.all():
7     print(obj.key, obj.last_modified)
8
9 # access the client from the resource
10 s3_client = boto3.resource('s3').meta.client

```

In boto3 when we create a client or resource, a default session is created, this session contains info like aws credentials, aws region,

The code we are using, read the dragons from s3 bucket using s3 select.

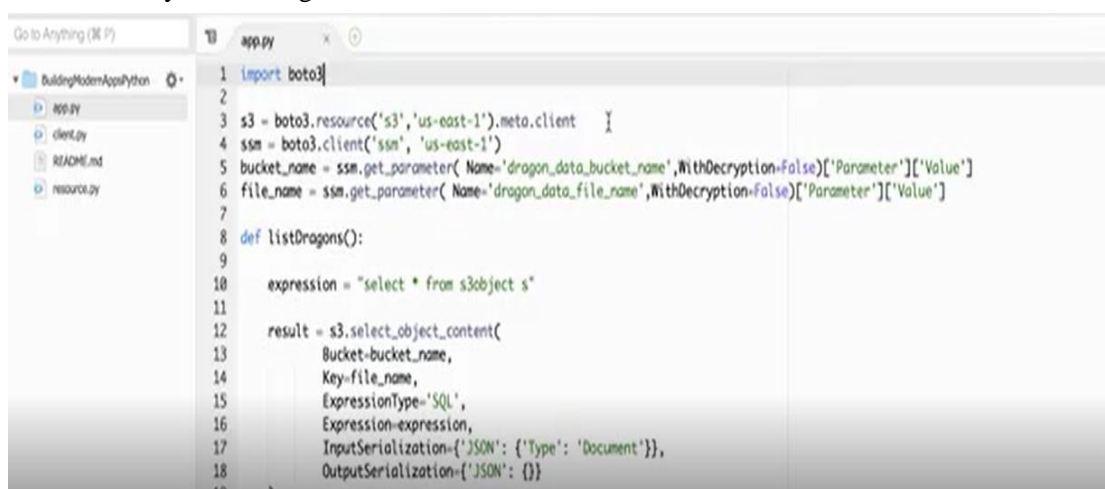
S3 select is feature of s3 that allows u to read a subset of data out of an s3 object using sql queries. Making the query from s3 bucket is powerful, especially s3 charges based on every request, amount of data and data transferred outside of region. For example we might download all data from s3, and go line by line, search for relevant data, apply filter to the data>> which is expensive and time consuming. So we use the s3 select.

Using s3 select submit the SQL query to read/filter a subset of data in an object. Data filtering is done by s3 not our code > cheaper and faster. In this course we query the json data.

Another service in our code: AWS Systems Manager Parameter Store: store/retrieve user-defined key value pairs. We store our s3 bucket name where their dragon data is located as well as file name, so If the bucket or file changes, we wont go back and change the code, instead the parameter can be updated in parameter store, and the code pull the latest information every time it runs.

Example:

Note: ssm is system manager



```

1 import boto3
2
3 s3 = boto3.resource('s3', 'us-east-1').meta.client
4 ssm = boto3.client('ssm', 'us-east-1')
5 bucket_name = ssm.get_parameter( Name='dragon_data_bucket_name', WithDecryption=False)['Parameter']['Value']
6 file_name = ssm.get_parameter( Name='dragon_data_file_name', WithDecryption=False)['Parameter']['Value']
7
8 def listDragons():
9
10     expression = "select * from s3object s"
11
12     result = s3.select_object_content(
13         Bucket=bucket_name,
14         Key=file_name,
15         ExpressionType='SQL',
16         Expression=expression,
17         InputSerialization={'JSON': {'Type': 'Document'}},
18         OutputSerialization={'JSON': {}}
19

```

```

20
21     for event in result['Payload']:
22         if 'Records' in event:
23             print(event['Records']['Payload'].decode('utf-8'))
24
25 listDragons()

```

Note: in our code we didn't use the credentials, AWS SDK handle locating the credentials,

Note: we must not hardcode the AWS credentials in code.

The SDK check the credentials in this order:

First check the credentials passed into boto client method: don't do this until the time u have to.

Then check the credentials passed into the session: again don't this

Next it will check the environment variables

Then it will check the credentials in the config file created when we run aws configure command:
this also not good

Finally it checks the instance metadata service which exists on Amazon EC2 instances that have IAM role associated: this is the preferred way to use AWS credentials. IAM allows u to access the temporary credentials which is more secure than embedding them or hardcoding them or using a file.

In this case we are developing on a cloud9 instance.

Note: if u are using the local, since we don't have instance metadata service, so we need to make sure about configuration files and credentials are configured with our region and our keys.

For more services, we can check the BOTO3 documentation.

Note: AWS Cloud9 is IDE.

In Cloud9, by default, Roles are not being used for permissions, instead Cloud9 uses managed credentials.

Very great:

<https://docs.aws.amazon.com/cloud9/latest/user-guide/welcome.html>

studied until <https://docs.aws.amazon.com/cloud9/latest/user-guide/environments.html>

Using Temporary Credentials in AWS Cloud9

Here we learn to Manage the temporary credentials in cloud9

Note: cloud9 automatically creates managed temporary credentials when it's hosted on EC2 instance.(not any other server).

Command: cat ~/.aws/credentials/> shows access key, secret key, and a token.>>> these credentials do replaced every 5 minutes. >never modify this file.

With these temporary credentials all actions can be implemented with the *:*

Except the following actions: (cloud9 credentials policy):

Some cloud9:...

Some sts:...

Some IAM:...

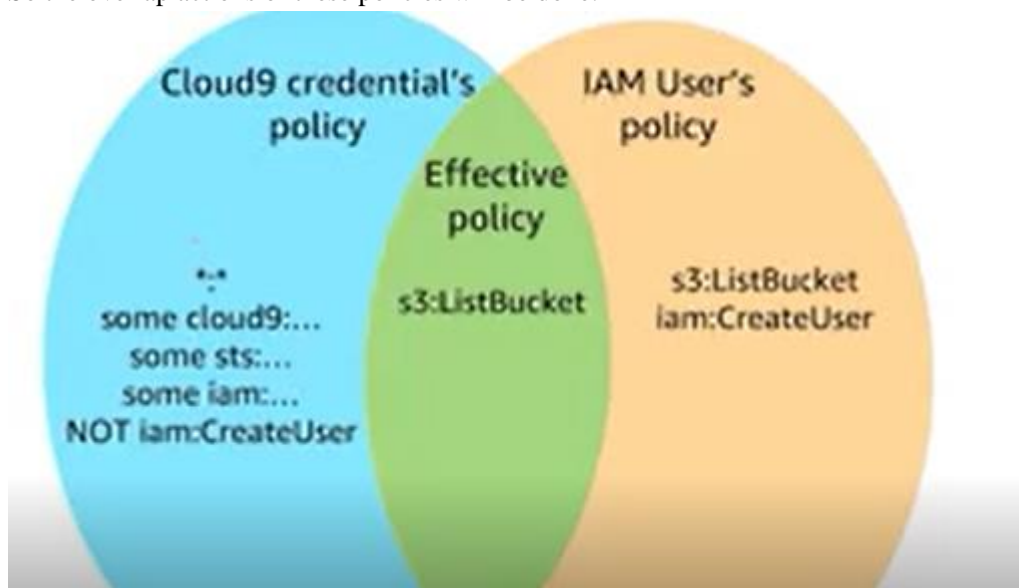
For example IAM createuser

And for IAM User's Policy, (we give the permission to IAM) for example the following are allowed:

S3:listBucked

Iam:createuser

So the overlap actions of these policies will be done:



What if we wont use these temporary credentials, since we want to share the environment to someone else, but we don't want them have the same access as me?

One solution is go to menu in aws cloud and disable temporary credentials>so we see:

Before disabling:

Aws s3 ls s3://dragon-data > shows the objects

After disabling:

Aws s3 ls s3://dragon-data > unable to locate the credentials.

And also ls ~/.aws> it gets empty

Solution:

First: Attach the IAM role to the EC2 instance that was created by launching the cloud9 instance.

So we go to EC2 console(explained in video) then in there we click on the attach/replace IAM role.

This time If we go back in our command line and s3 ls s3://dragon-data > then it shows the objects

Second:

Before start this, we need to cancel the first one by detaching the IAM role.

Add the access key and secret key to the credentials file. So we can add them using aws cli >

aws configure > enter the access key and secret key and... > in this way, the credential file will be generated.

Note: this is not good way!

Note: if we enable AWS managed temporary credentials again, it override this file in 5 minutes, Although we discussed the aws cli, this is also with aws sdk.

Aws serverless application model(SAM): open source framework for building the serverless applications. It provides the shorthand syntax to express APIs, functions, databases. We can define the application we want to model. During deployment, SAM transforms and expands the SAM syntax to AWS cloudFormation syntax and enables us to build serverless applications faster.

Server-Less Application:

Combination of AWS Lambda Functions, event sources, and other resources that work together to perform distributed tasks.

SAM consist of two main components:

1. AWS SAM Template Specification: used to define the server-less application, provides us with the syntax to describe everything to make our server-less app.
2. AWS SAM Command Line Interface (SAM CLI): is the tool to build the apps that defined by the SAM templates. This provides the commands that enable us to verify the template files are written according to the specification, work with resources package and deploy the app to the AWS cloud, and so on. Note: this is different CLI from the AWS CLI! And this only used with SAM. It needs to be installed and configured separately in order to run. So SAM sounds a lot like AWS cloudformation, since it is extension of CloudFormation, so we get the same deployment capabilities: we can define resources, use CloudFormation in our SAM template, and we can use the full suite of resources, intrinsic functions and other template features that are available in CloudFormation. Since SAM integrates with other AWS services, there are several benefits:
 - Ease of organization of related components and resources and operate on single stack, so we can use it to share the configuration between resources and deploy-related resources together as a single-versioned entity.
 - Enforce best practices: since SAM enables us to templatize what we are deploying, we can enforce best practices such as code reviews.
 - Ease of Deployment: since this is template, we are able to test and deploy the same environment
 - Integrated with AWS for building the server-less apps. We can discover new applications in the AWS server-less app repository. We can use the AWS cloud9 IDE to author, test, debug our SAM-based server-less apps, we can build deployment pipeline for server-less app using AWS CodeBuild, AWS CodeDeploy and AWS CodePipeline. And even we can use the AWS CodeStar to build out project structure, code repository and a CI/CD pipeline that is automatically configured.

Video: Lecture AWS Toolkit for PyCharm

Important:

<https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html>



<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

Identity and Access Management (IAM)

Amazon Elastic Compute Cloud (Amazon EC2) instance