Audio file

Your Recording 27.wav

Transcript

00:00:02 Speaker 2

In this lesson, we're going to introduce the Sagemaker SDK for Python And see why we might use that instead of the regular Photo Three SDK. So we're going to look at what SDK's software developer kits are available for automating AWS and why they might not be ideal for machine learning. We're then going to look at what the Sagemaker SDK for Python is and why we might prefer to use that. We're then going to think about using that SDK form within Jupyter Notebooks to automate steps within the machine learning pipeline. And lastly, we'll talk about some benefits of using the SDK.

00:00:32 Speaker 2

And see some examples of real world customers who have successfully implemented it. So why is a problem to use the regular AWS SDK for Python which we call boto three? Well, we can install the SDK for Python, we can download it from the AWS website and it will allow us to automate almost any service within AWS. So if I wanted to create or power on or power off a virtual machine in the EC2 service, then I can do that. Or maybe I wanted to interact with data in S3 objects held in a bucket, upload object, download object or create bucket.

00:01:02 Speaker 2

In Dynamo DB, a key value store, we could use that for querying, we could use that for insertion, deletion. And in Sagemaker, we could use the SDK for performing actions such as starter data processing job, starter training job, or host my model for inference. So I could use this general purpose SDK for any one of those actions. Now when I do that, what's actually happening is that SDK is turning simple Python function calls into REST API calls that are directed at the target AWI service. So there's a REST API endpoint for EC2, there's a separate one, REST 3 dynam.

00:01:32 Speaker 2

Sagemaker in each AWS region worldwide. But the wonderful thing here is we can create in just one line a client reference pointing at one of the services. And then all of the methods that that service offers are now available to us programmatically. So when we want to automate anything in our AWS account, the AWS SDK for Python is our friend. We could create an SNS topic, a notification topic for fanning out notifications. We could launch an EC2 instance or virtual machine. We could upload an object to an S3 bucket. We could even create or update a virtual call center in Amazon Connect or even download.

00:02:02 Speaker 2

From a satellite via satellite ground station. In other words, regardless of which AWS service you want to use, Boto 3 is a general purpose software developer kit that will expose the functionality of that target AWS service to you as simple Python function calls. So why would this regular SDK not be ideal for machine learning? Let's take a look at a code example of using Boto 3 to automate AWS. In this simple example, what we're doing is we're importing Boto 3. That's the simple and regular SDK that is general purpose for using with any AWS ServiceNow. How we get a service.

00:02:32 Speaker 2

Reference to a target service well, you can see that in the second line here I can see I'm creating a handle called S3 client and I'm saying OK portal 3 dot client is a client method of portal 3 and connect to S3. I'm naming the official service name of the service I want to connect to I'm passing that as a parameter so you can imagine if you want to create a client EC2 or Dynamo DB or Sagemaker, it's just a matter of changing what's inside of those single quotes there that then gives you a service reference and once you've got that service reference, you can then use that Now in this example we're just getting.

00:03:01 Speaker 2

A file and we're going to upload a file.

00:03:03 Speaker 2

Up into an H3 bucket. So we're setting some variables about what the bucket name is, what the local file path is, what we want the name of the object to be once we save it in test 3. And then down at the bottom, that's where the crucial operation happens. S3 client dot upload file. So we're using the upload file method of our S3 client. So upload file requires some input parameters that'll be the bucket, the file path, and what to call the object when it gets there. So it's quite easy to kind of interpret what is going on in Python when using boto 3. And the wonderful thing here is that any REST API operation the S3 supports.

00:03:32 Speaker 2

We can actually gain access to through our service reference client here. So upload file we know will be an API method of S3. Equally there will be a download, there will be a rename, there will be a create bucket, whatever operation you need to do.

00:03:43 Speaker 2

One other thing I need to be aware of here is when I import my SDK here, I would need to have that SDK already installed on the system where my Python code is running. So I would need to have that downloaded and installed first, either globally or within a virtual environment within my Python install. Let's now look at an example of using Photo 3 to automate Sagemaker.

00:04:04 Speaker 2

Here we can see the top that we are importing Boto 3 as normal. Then we are creating our service reference to Sagemaker. We're calling our variable Sagemaker, but we're calling the client method Boto 3. I'm passing the name of the service that we want to service reference to. We can then use that Sagemaker service reference to invoke different methods that are exposed by that ServiceNow.

Here we're building a response object and I can see I'm using my Sagemaker service reference with the Create training job method. So create training job must be an API method of Sagemaker. That is well documented.

00:04:31 Speaker 2

Now creating a training job is going to be telling Sagemaker, I want you to spin up a dedicated compute instance to perform some model training and therefore we need to supply some input parameters. Now don't worry at this point too much about model training and what's needed here. We have a whole lesson covering this later on. What we want to concentrate on for the moment is how does our code look and how much do I need to configure to get this started. So I'm specifying a job name, I'm specifying what algorithm I want to use by way of using a particular container image. I'm specifying the security role I want to run.

00:05:01 Speaker 2

Under I'm having to specify my source data, whereas in S3 that my training data resides, I'm having to specify where I want the output to be of the training process. In other words, the model that the training process will produce and some hyperparameters to guide the training process in how it should perform its job.

00:05:16 Speaker 2

So it's not terrible, but it could be better. It's quite verbose. We need to put in a lot of additional parameters to get it to do this job. And this is because Photo 3 is a general purpose SDK, it's not dedicated just to ML. I could use the same SDK for talking to EC2S3, SNS, Dynamodb and so on. So could we do this in a different way? And the answer is yes, with the Sagemaker SDK for Python.

00:05:38 Speaker 2

So we know we have got the regular SDK for Python that can access any AWS service we point at.

00:05:44 Speaker 2

But if we use this new Sagemaker SDK for Python, that is a new separate SDK that we will need to import, then that allows us to communicate with Sagemaker. Sure, still under the hood it's vRS API, but the high level constructs that it makes available to us in Python make it far easier to use and perform the activities that we expect to do in a machine learning pipeline. So it's not about saying that we can do different things in the Sagemaker SDK for Python, it's that we can do things simpler and by using high level constructs that makes our code cleaner, are more understandable to the data scientist, and we can.

00:06:14 Speaker 2

Understand one another's intentions when we look at each other's code. Remember, when we use things like Jupiter notebooks, we are encouraging collaboration. We are encouraging the ability for another to follow in our footsteps, modify, and make better. So if we can use constructs that align with our intentions in data science, then that's always going to lead to better, more understandable code than we would find elsewhere.

00:06:34 Speaker 2

So the Sagemaker SDK gives me high level abstraction centered around my activities. Its purpose built just for machine learning, and therefore things like supplying parameters. There are default values baked into the Sagemaker SDK, meaning I don't always have to specify everything like I do with Photo 3. It's user-friendly because it's targeted at just Sagemaker and just the activities that a data scientist is going to be performing. It's a lot more easy to understand and to use, and it supports integration with many of the Sagemaker automation capabilities.

00:07:02 Speaker 2

In particular, Sagemaker Pipelines. Now Sagemaker pipelines. We have a dedicated lesson on that later on, but for now, we just need to know that it's a sequence of activities performed as one operation, and we can invoke pipelines and monitor pipelines directly from the Sagemaker SDK. Let's now look at an example of using the Sagemaker SDK. Now this example will create a training job in Sagemaker, exactly the same as the previous example we looked at using Boto 3, but I think you will agree that the code here looks cleaner, simpler, and most importantly, shorter. There are fewer lines of code needed.

00:07:32 Speaker 2

Let's break it down right at the top. We are importing Sagemaker instead of Moto 3, so we will need to have the Sagemaker SDK downloaded and installed on our system first before we can run this code. Now here's the most important part of our code. We are creating an estimator object that will represent the training job that Sagemaker will run for us. How do we get that object? Well, we are calling the Sagemaker a package, We're importing the subclass Estimator, and we're calling the estimator method. Now I can see here when I call that to build my estimator object that represents the training job.

00:08:02 Speaker 2

I need to specify some properties of that object. For example, the algorithm that will be used, which is contained in a container image. Here we're using linear learner. We need to specify what IAM role I'm going to be running under, how much compute power should be allocated to the training job. In this case, just one M5 large instance will be enough. Where to store the model once it's been produced will be. Ultimately it will be a TGZ file, so that's the model artifact. So it's going to be put into that S3 bucket and what session to run this under. So that builds us.

00:08:29 Speaker 2

Abstract high level object that represents the training job.

00:08:32 Speaker 2

But we have not yet started the training job. We can see below the highlighted area. We are then using the set parameters method on the estimator object. And that allows us, in this case here to specify any hyperparameters that are relevant, such as batch size or the steps learning rate, things like that. So again, that's further specifying properties of the job. And I like the way they've done that here because it means that we've created the estimator object. But maybe on subsequent

iterations of running this code, you only want to modify the hyperparameters. So I would only need to adjust that one line, let's say, if I wanted to change the mini batch size from 32 to 64.

00:09:02 Speaker 2

It's just one line I need to change. I'm not changing anything else about the object creation. So minimizing changes between experiments is always going to be easier to follow and understand for another data scientist you might be collaborating with. But right at the bottom here, we call the fit method on the estimator object. And that's what's actually going to start our training job. Now calling a fit method against a particular object is something that many data scientists will be familiar with because when they have used toolkits like scikit learn, calling the fit method, it would be typical that would be the invoke method. So fit is what we use to start that training job. Lastly, when.

00:09:32 Speaker 2

Calling that training job, we're supplying the data set that we want to use for training. And again, think about experimentation here, that if you were running multiple experiments, maybe using different permutations of datasets, you could run a training job, then modify that line to use a different training data set and run the job. Again, you're only changing one line. And again, think about how this will look in terms of tracking your experiments as a data scientist. It's a lot cleaner. It's a lot more focused to the workflow that a data scientist would follow. Using the Sagemaker SDK is not limited to just Sagemaker, other AWS services and features that make sense.

00:10:02 Speaker 2

To a machine learning workflow are included in this SDK. For example, we have S3, the simple storage service. It's likely that our datasets will reside there and it's likely that's where we are going to write out our model artifacts. So the ability to upload or download data. Checkpoint storage in S3 is built into the Sagemaker SDK. We have integration with the Identity and Access Management service, our Security Service for authenticating and authorizing users to perform actions. Primarily we will be working with IAM rules, a role typically attributed to a resource like an instance or a training job or.

00:10:32 Speaker 2

Job. You're defining the security context under which a processing job or training job will run under and therefore determining what other resources that job can access. At a minimum, it would likely need to be able to access your S3 bucket, and we have access to AWS Step Functions now. Step Functions are a wonderful way of orchestrating a number of activities across AWS. It's kind of like building a flow chart of actions, and these can be serverless actions like AWS Lambda or database insert actions or actions within Sagemaker.

00:11:02 Speaker 2

But the Siegemaker SDK will certainly allow us to build and orchestrate pipelines using step functions. We also have access to the Sagemaker feature store now. We're going to cover this feature store in more detail in our advanced inferencing lesson later on. But for the moment, we just need to know that the features store could be helpful to us if we are going to be doing multiple

machine learning projects using the same data set or similar datasets. It allows us to engineer features and then save those engineered features so we may reuse them again. We have integration with Cloudwatch. Amazon Cloudwatch is our.

00:11:32 Speaker 2

And metrics system, so we can have automatic logging for data processing jobs and our training jobs and for inference if we choose to host our models on Sagemaker and we have access to the Sagemaker model registry. So when we have built a model, we really want to version control that. We want to track it because when we deploy it, what happens if we our model starts to drift in terms of quality, We'd want to train a new version of the model and then roll out that new version. So what keeps track of those model artifacts and their versions and their approval for use in production?

00:12:01 Speaker 2

The model registry is ideal for that, and the Sagemaker SDK allows us to interact with it. But it is likely that what you want to achieve in your Jupyter notebook code is a combination of working with Sagemaker and other AWS services, and not every AWS service is reachable using the Sagemaker SDK. So the Sagemaker SDK is ideal for automating Sagemaker and a few select services that make sense to my machine learning pipeline, such as S3 or Step Functions, things like that. But what about if I want to automate other things in AWS? Maybe I want to?

00:12:31 Speaker 2

Notifications. Maybe I want to send e-mail, or maybe I need to update a configuration management database. In those cases, the Sagemaker SDK is not going to be helpful to me. That drives us to a hybrid approach. We can use the Sagemaker SDK for those ML activities such as data processing, training and inference. But we can still use Photo 3 for other interactions with the likes of the Simple Notification Service or Dynamodb or indeed any other AWS service that the Sagemaker SDK is not aware of. So I might want to use Photo 3 to communicate with Simple Notification Service to send out a notification that maybe training.

00:13:01 Speaker 2

Completed or maybe updated database, maybe it's a configuration management database and we've now created a new model artifact and maybe that should be recorded in a CMDB. Or maybe there is another AWS service we wish to take advantage of like the simple e-mail service or something like that. Remember, Boto 3 is able to automate almost all AWS services. So whatever else we need to do, we can still do it from within our Python code in our Jupyter notebooks. Let's look at a code example of using both SDKS. So here at the top we can see in our import statement we are importing both Boto 3 and Sagemaker. We can see we are creating our estimator object.

00:13:31 Speaker 2

Using the Sagemaker SDK, remember we are creating an instance of an object that represents our training job, and that's where we specify the properties of that object, your bucket where the model is to go, the compute required, and the algorithm within the container image. Then you can see a little bit further down where we're setting our hyperparameters, and we can see a little bit further down where we're calling our fit method, and we call our fit method on our estimator object that is

created that is starting the training job. And we're passing in our data set that is to be used for training. But notice then we create a client using boto 3 to the simple notification.

00:14:02 Speaker 2

Then once we've got that client, we can use whatever method of that service. In this case here publish method, we are publishing a message to a topic. A topic is just something that multiple users may subscribe to. And here I've got a topic called training job notifications and I'm putting a message out saying training job. And then I'm supplying the name as a variable, the job name has started and putting in the subject header of Sagemaker training job. So that's going to publish a message to that SNS target.

00:14:27 Speaker 2

And any person or system or other process that is subscribed to the topic called training job notifications will receive that message.

00:14:33 Speaker 2

That by getting the knowledge that the training job has started, but it's a great example of looking at the Python code and saying, all right, we'll use the appropriate SDK for the activity that I'm performing. So what results can I expect by adopting the Sagemaker SDK? Well, I can expect to see my data scientists develop their models quicker. They can do this because those high level constructs like the estimator object make it easy for me to translate my intention in the machine learning pipeline into actual executable code. The code is generally going to be easier to maintain. There are fewer lines to maintain and.

00:15:03 Speaker 2

Because those high level constructs abstract away some infrastructure complexity from me, I can remain focused on what it is I'm trying to do, what ML problem I'm trying to solve. And this generally results in a shorter time to value because you're not having data scientists having to learn all the methods of the Sagemaker API so that they can call Boto 3 with them. Instead, they are just learning the high level constructs that align with the activities that they want to perform. So this generally is going to result in more efficient management of your training process and your inference process. Many enterprise customers who use Sagemaker AI.

00:15:33 Speaker 2

Have now moved to using Sagemaker SDK instead of Moto 3 to automate training and.

00:15:38 Speaker 2

Inference For example, AstraZeneca, the large pharmaceuticals company, they have used Sagemaker to create machine learning environments in minutes instead of months. Is popular with our data scientists to use the Sagemaker SDK because it's simple to understand and it quickly delivers insights because they can concentrate on what a problem they're trying to solve rather than thinking about infrastructure tasks. NatWest Bank they also have adopted Sagemaker SDK and built 30 plus ML use cases in just four months with it. This allowed them to get personalized marketing and fraud detection use cases into production far.

00:16:08 Speaker 2

Faster than the otherwise would have.

00:16:10 Speaker 2

So what have we seen in this lesson?

00:16:13 Speaker 2

We have seen that there are two software developer kits for Python that we could use. The general purpose SDK for Python called Boto 3 and this Sagemaker SDK for Python. The dedicated SDK with higher level constructs just for machine learning. We've seen by using the Sagemaker SDK those higher level constructs make it easier to code to perform activities in mill whether it be data processing or training or inference.

00:16:35 Speaker 2

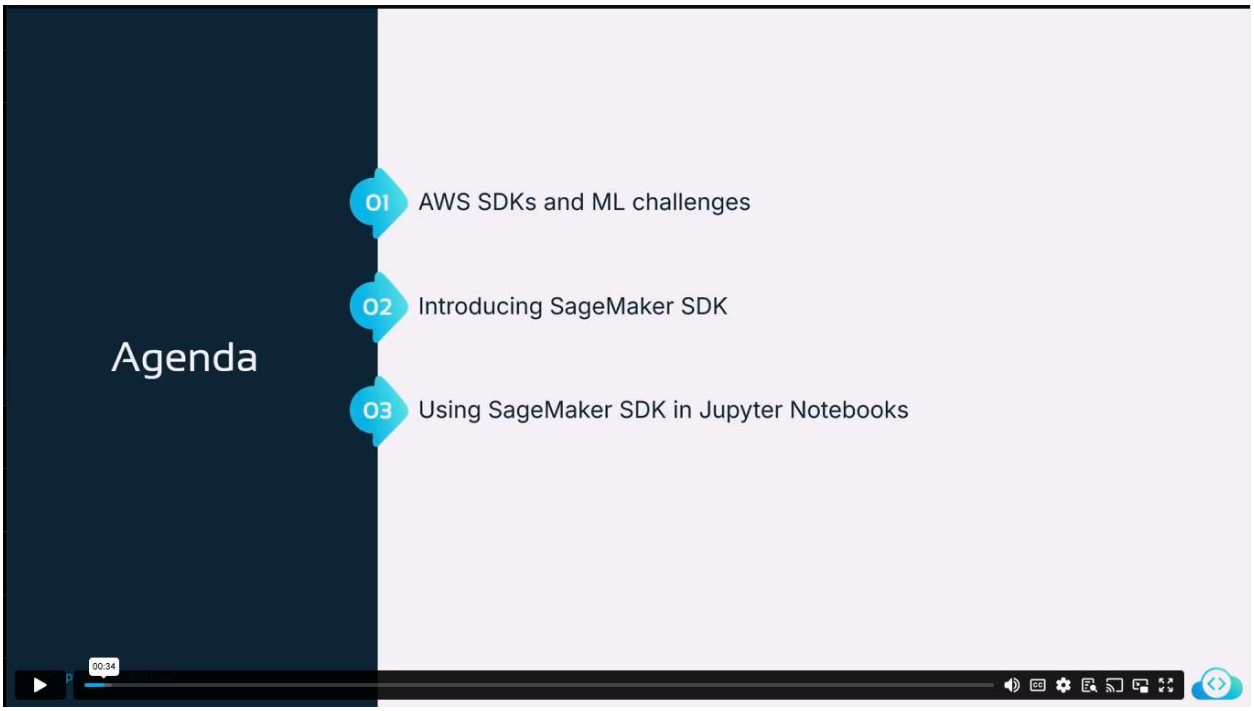
Because of those higher level constructs with default values, we need less lines of code, simpler, cleaner code aligning with the intention of the data scientist.

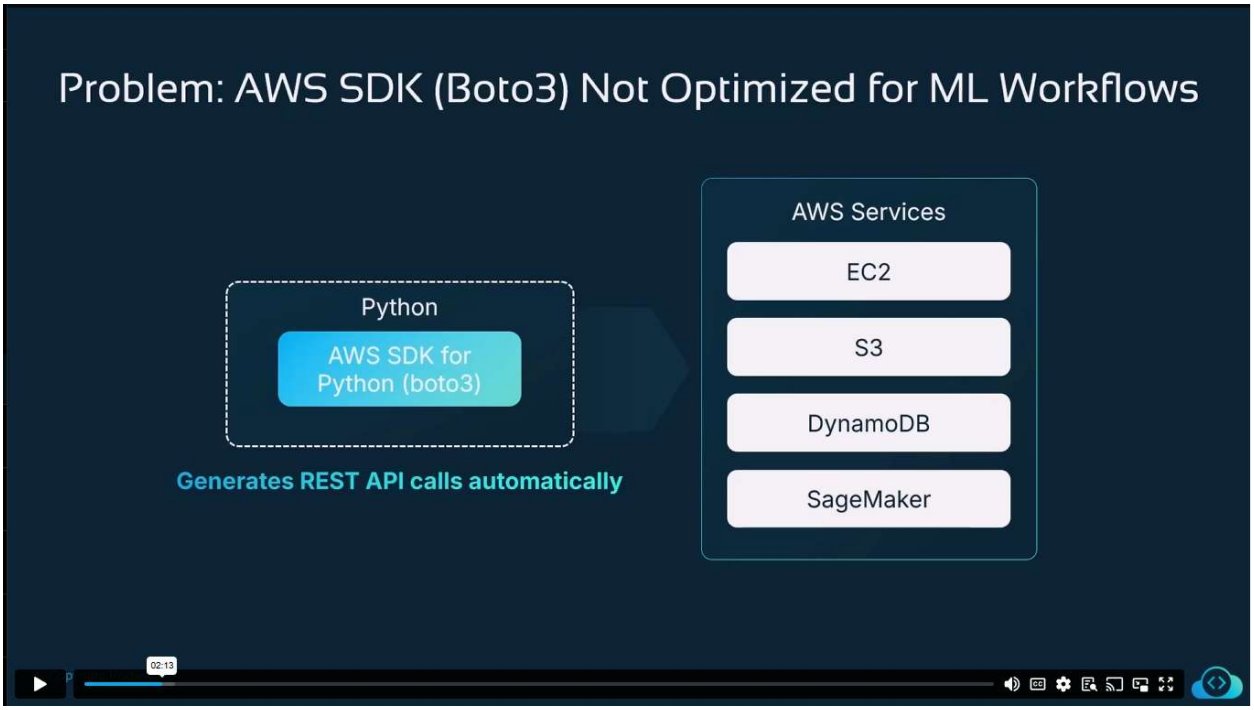
00:16:43 Speaker 2

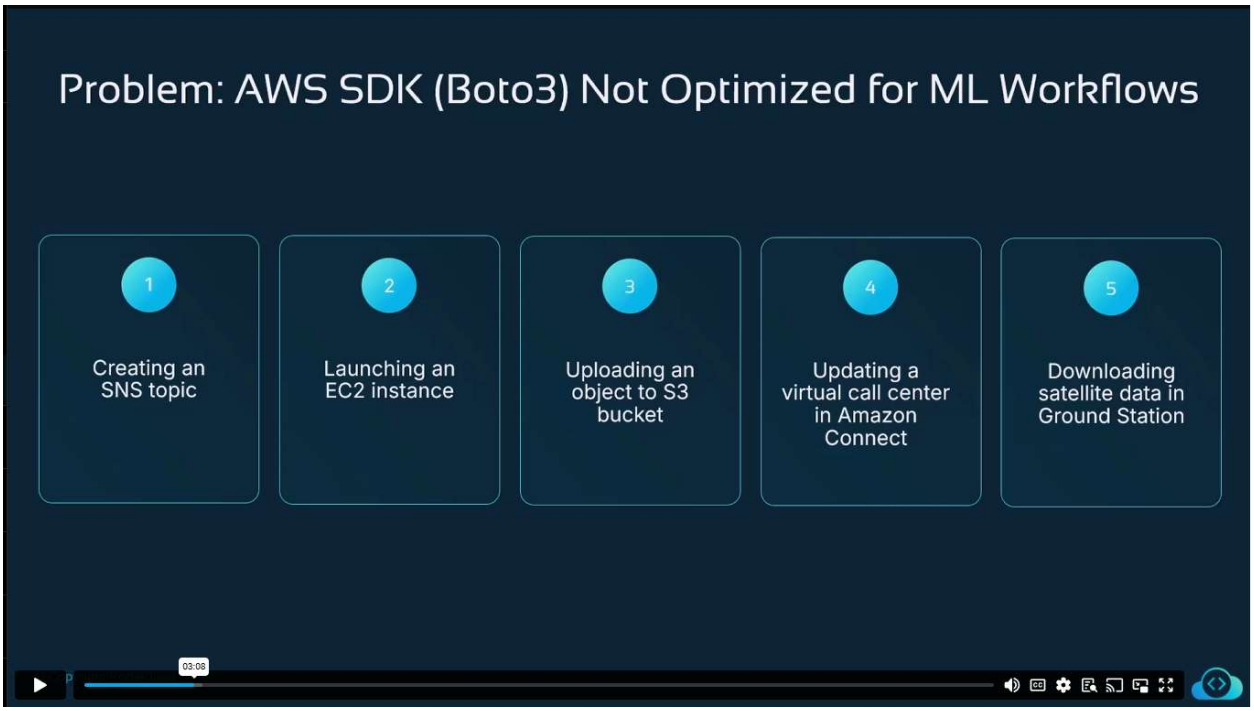
And many advanced features are built into those constructs and methods, and we can overwrite them with additional properties. So if we're struggling to identify when to use one, if we're doing ML training, inference, data processing, use the Sagemaker SDK. For everything else, such as notifications or downstream processing, insertion of items into database, anything like that, use the general purpose SDK. Photo 3. So it's common that your Jupyter notebook will have code cells that make use of both Sdks to achieve the outcome that you require.

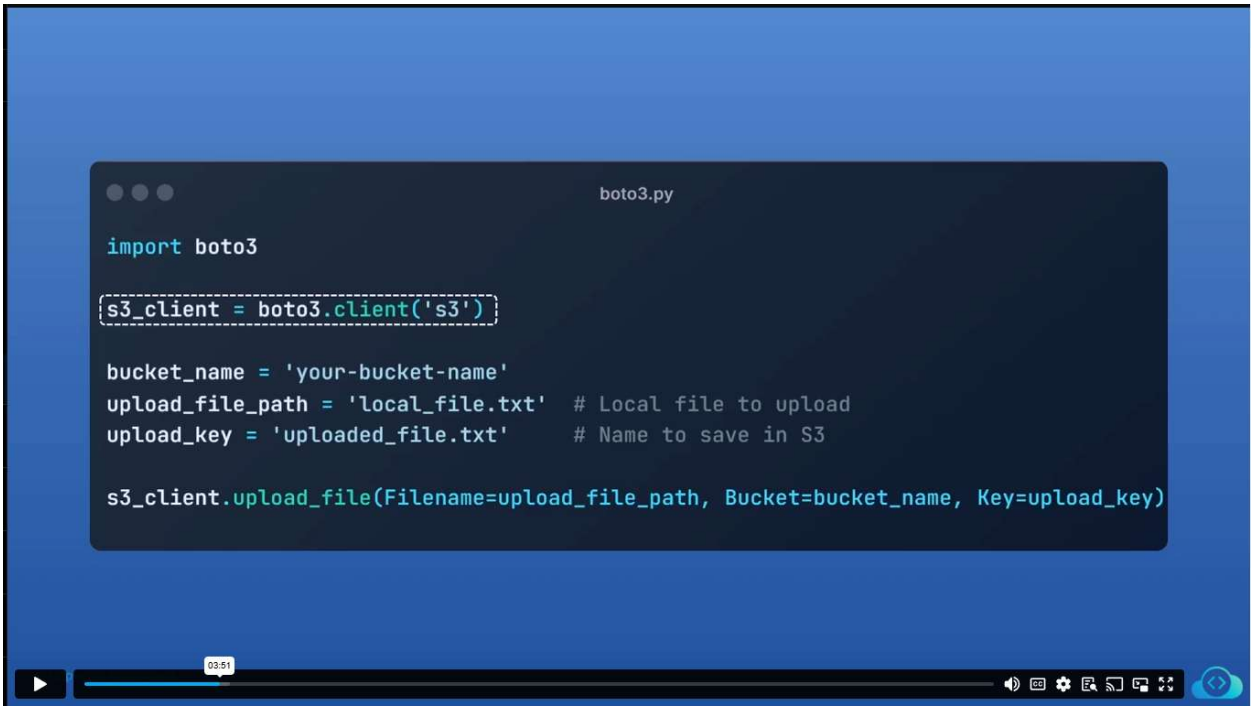
00:17:11 Speaker 2

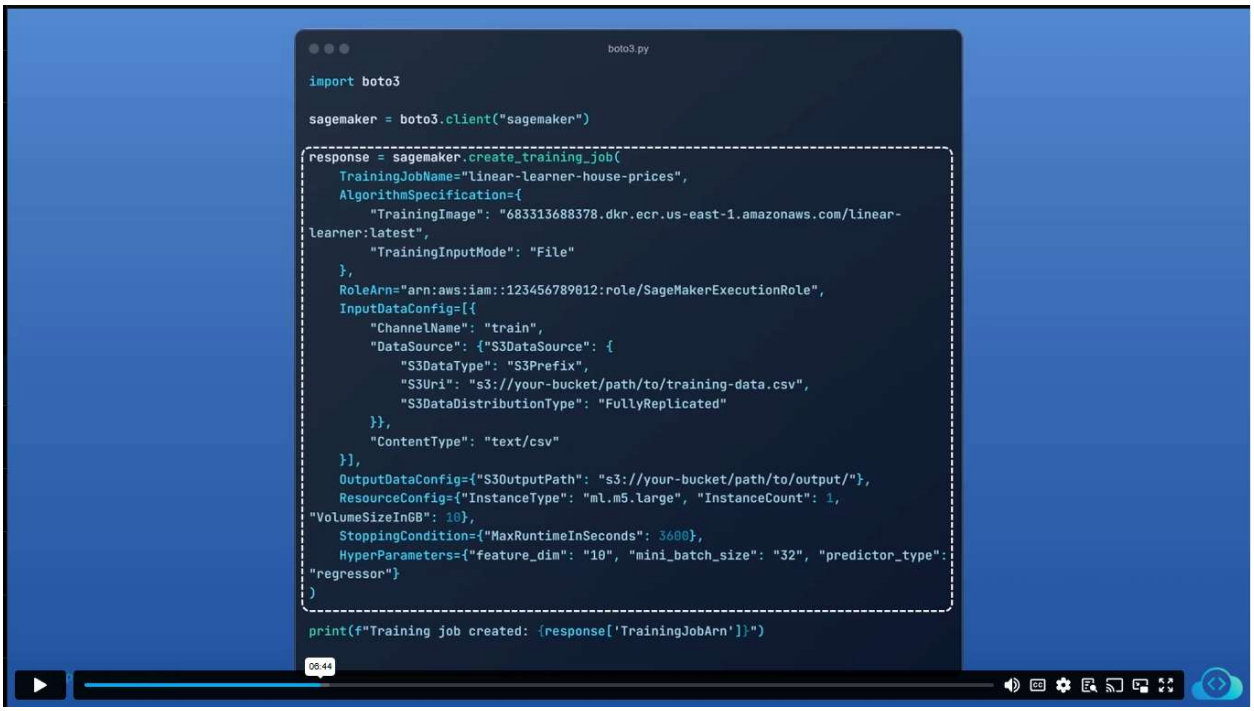
Now, in our next lesson, we're going to look at our first exploration of the AWS Management Console and the Sagemaker AI Console. I'll see you there.

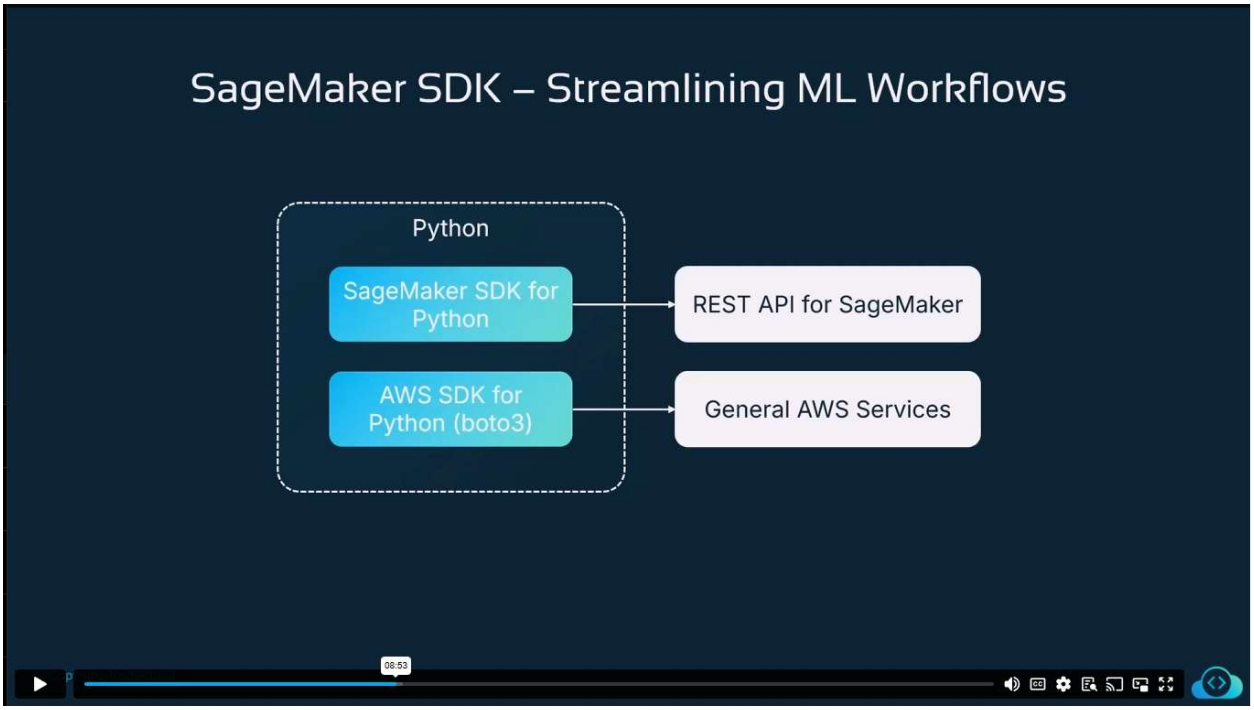

Agenda
01 AWS SDKs and ML challenges
02 Introducing SageMaker SDK
03 Using SageMaker SDK in Jupyter Notebooks
00:34

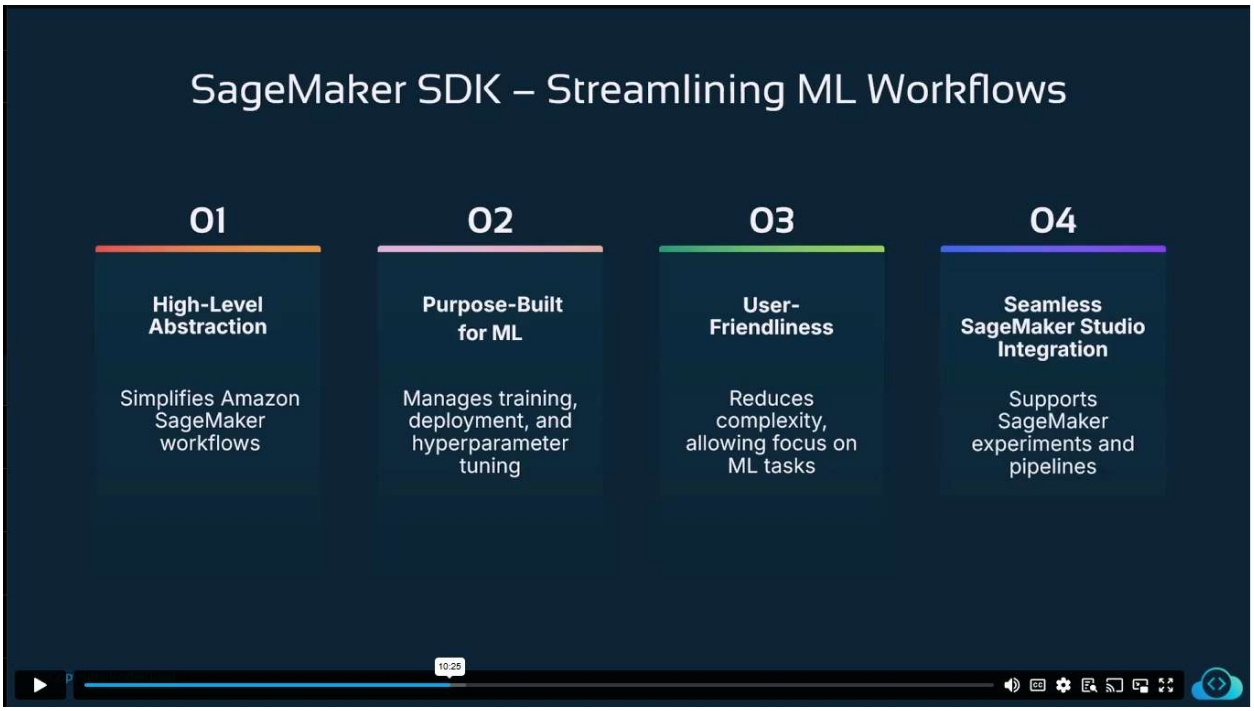

Problem: AWS SDK (Boto3) Not Optimized for ML Workflows
Python
AWS SDK for Python (boto3)
Generates REST API calls automatically
AWS Services
EC2
S3
DynamoDB
SageMaker
02:13

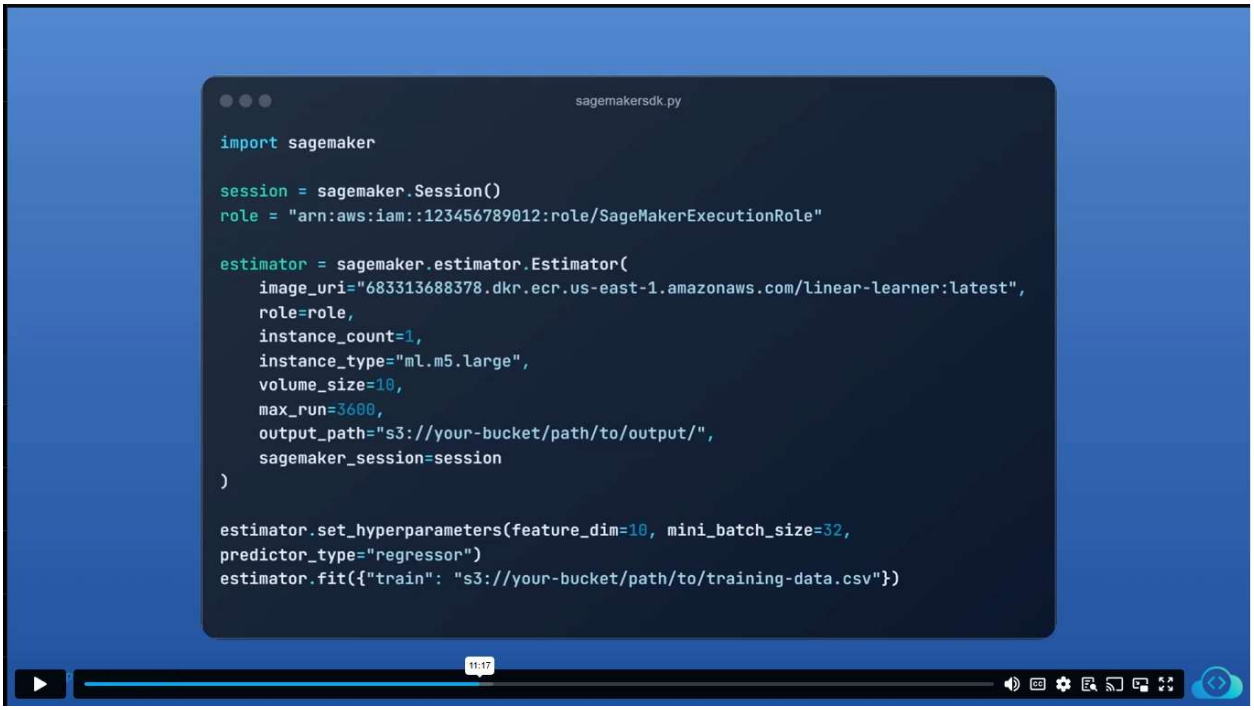

Problem: AWS SDK (Boto3) Not Optimized for ML Workflows
1
Creating an SNS topic
2
Launching an EC2 instance
3
Uploading an object to S3 bucket
4
Updating a virtual call center in Amazon Connect
5
Downloading satellite data in Ground Station
03:08

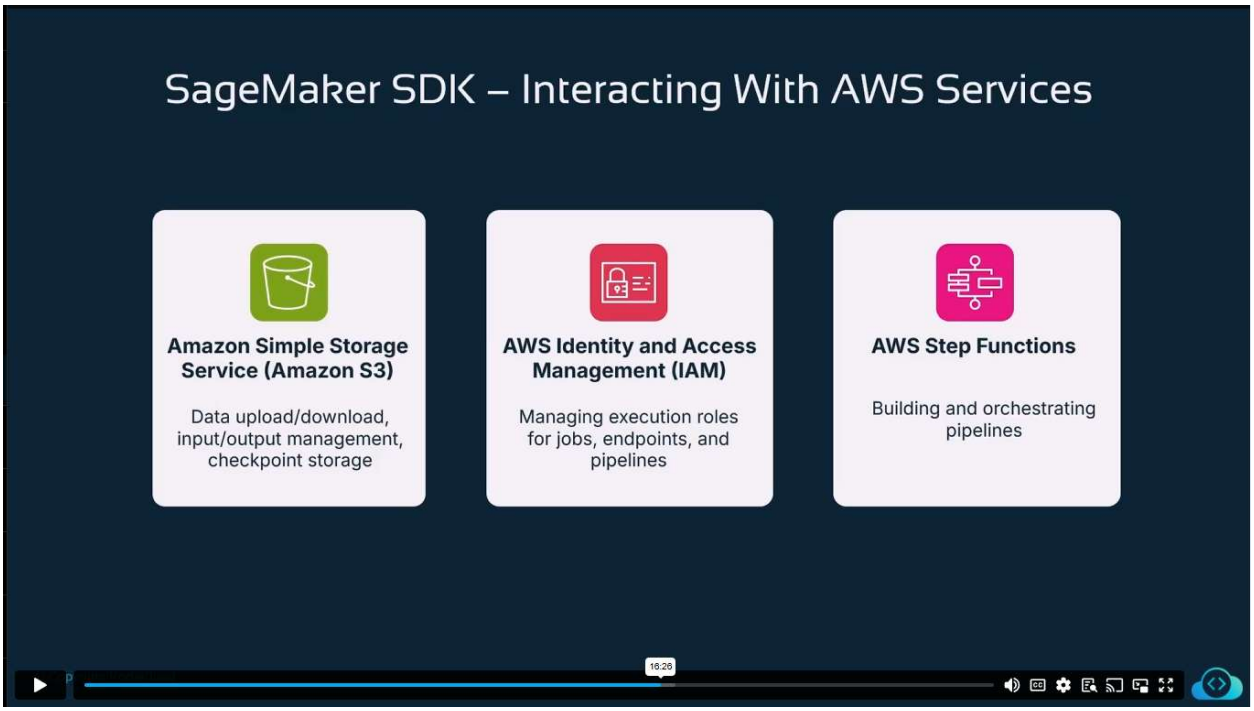

boto3.py
import boto3
s3_client = boto3.client('s3')
bucket_name = 'your-bucket-name'
upload_file_path = 'local_file.txt' # Local file to upload
upload_key = 'uploaded_file.txt' # Name to save in S3
s3_client.upload_file(Filename=upload_file_path, Bucket=bucket_name, Key=upload_key)
03:51

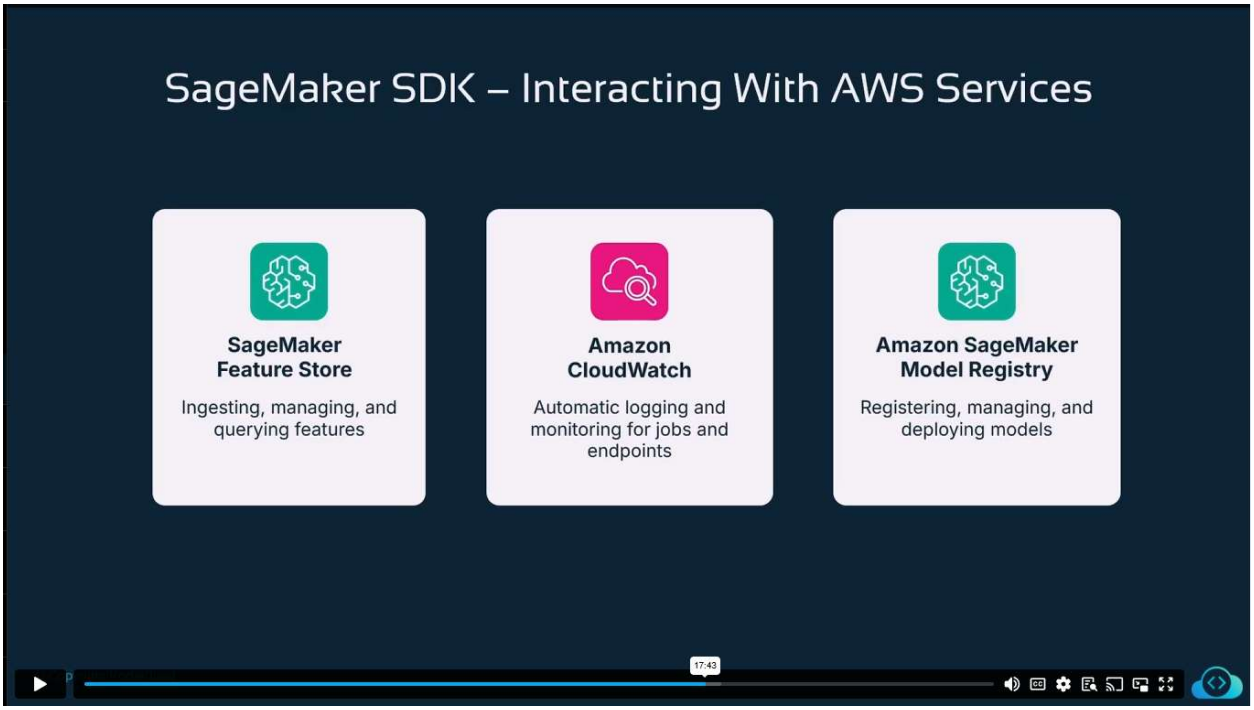

boto3.py
import boto3
sagemaker = boto3.client("sagemaker")
response = sagemaker.create_training_job(
TrainingJobName="linear-learner-house-prices",
AlgorithmSpecification={
"TrainingImage": "683313688378.dkr.ecr.us-east-1.amazonaws.com/linear-learner:latest",
"TrainingInputMode": "File"
},
RoleArn="arn:aws:iam::123456789012:role/SageMakerExecutionRole",
InputDataConfig=[{
"ChannelName": "train",
"DataSource": {"S3DataSource": {
"S3DataType": "S3Prefix",
"S3Uri": "s3://your-bucket/path/to/training-data.csv",
"S3DataDistributionType": "FullyReplicated"
}},
"ContentType": "text/csv"
}],
OutputDataConfig={"S3OutputPath": "s3://your-bucket/path/to/output/"},
ResourceConfig={"InstanceType": "ml.m5.large", "InstanceCount": 1, "VolumeSizeInGB": 10},
StoppingCondition={"MaxRuntimeInSeconds": 3600},
HyperParameters={"feature_dim": "10", "mini_batch_size": "32", "predictor_type": "regressor"}
)
print(f"Training job created: {response['TrainingJobArn']}")
06:44

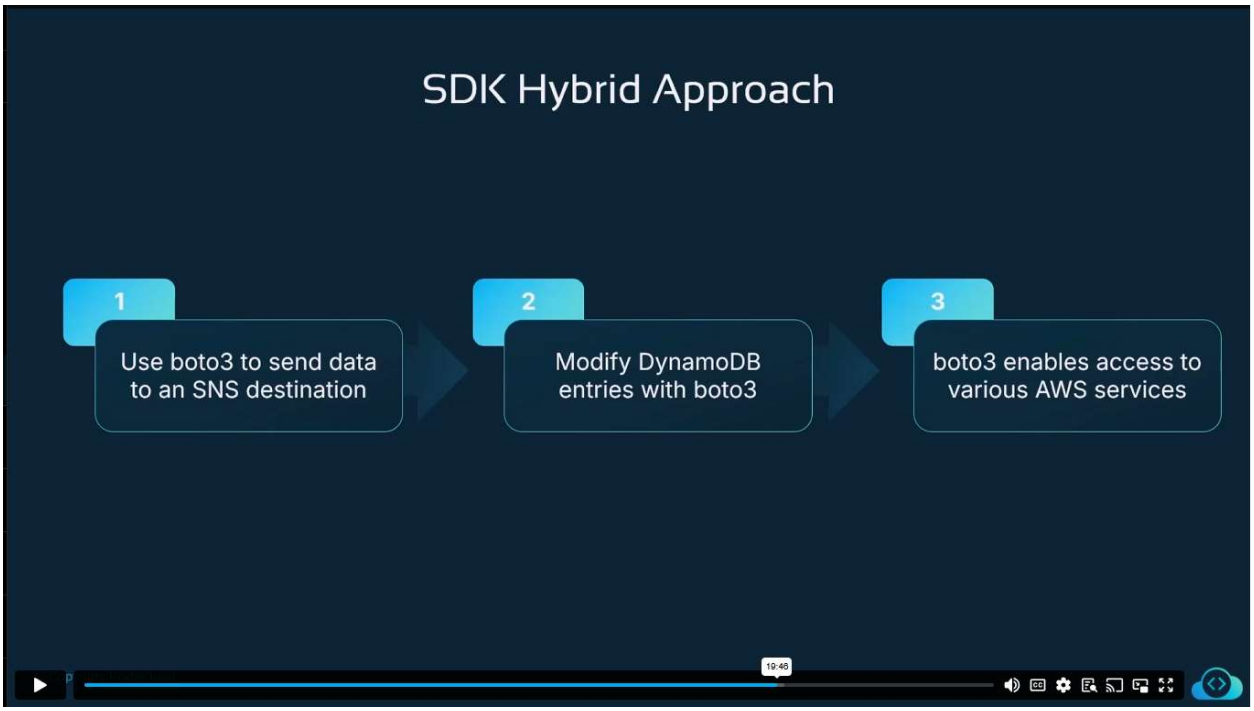

SageMaker SDK – Streamlining ML Workflows
Python
SageMaker SDK for Python
AWS SDK for Python (boto3)
REST API for SageMaker
General AWS Services
08:53

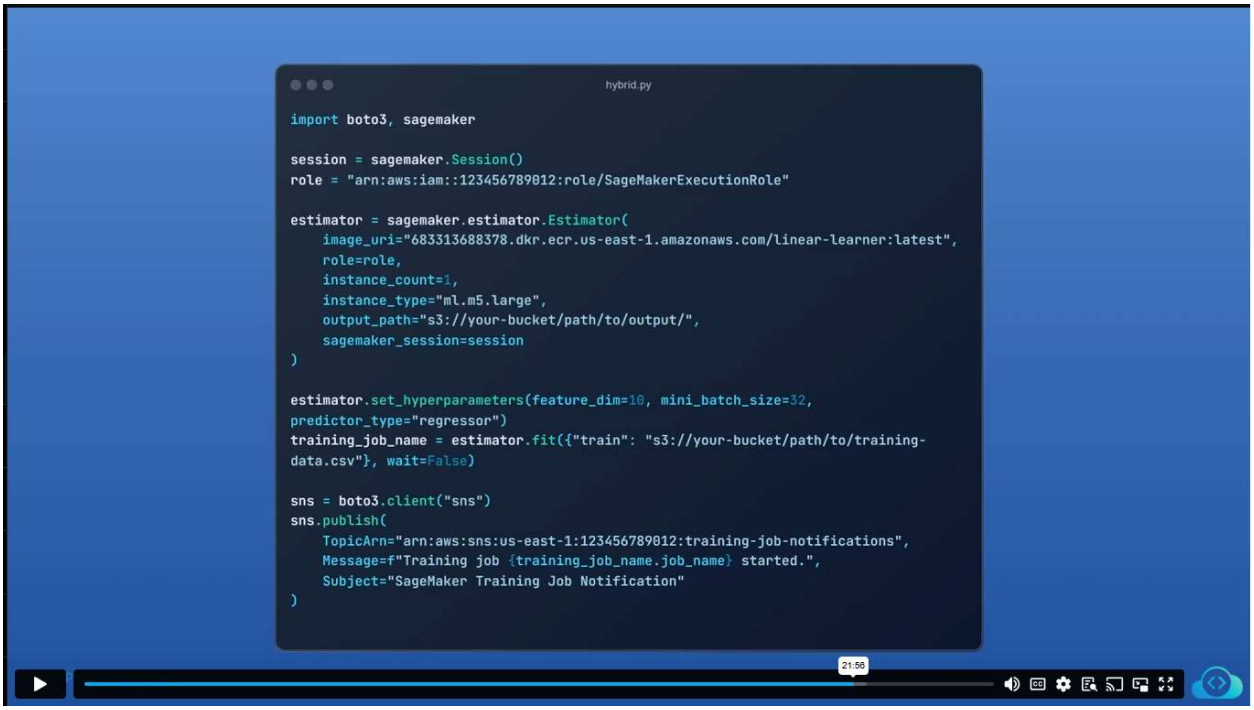

SageMaker SDK – Streamlining ML Workflows
01
High-Level Abstraction
Simplifies Amazon SageMaker workflows
02
Purpose-Built for ML
Manages training, deployment, and hyperparameter tuning
03
User-Friendliness
Reduces complexity, allowing focus on ML tasks
04
Seamless SageMaker Studio Integration
Supports SageMaker experiments and pipelines
10:25


sagemakersdk.py
import sagemaker
session = sagemaker.Session()
role = "arn:aws:iam::123456789012:role/SageMakerExecutionRole"
estimator = sagemaker.estimator.Estimator(
image_uri="683313688378.dkr.ecr.us-east-1.amazonaws.com/linear-learner:latest",
role=role,
instance_count=1,
instance_type="ml.m5.large",
volume_size=10,
max_run=3600,
output_path="s3://your-bucket/path/to/output/",
sagemaker_session=session
)
estimator.set_hyperparameters(feature_dim=10, mini_batch_size=32,
predictor_type="regressor")
estimator.fit({"train": "s3://your-bucket/path/to/training-data.csv"})
11:17

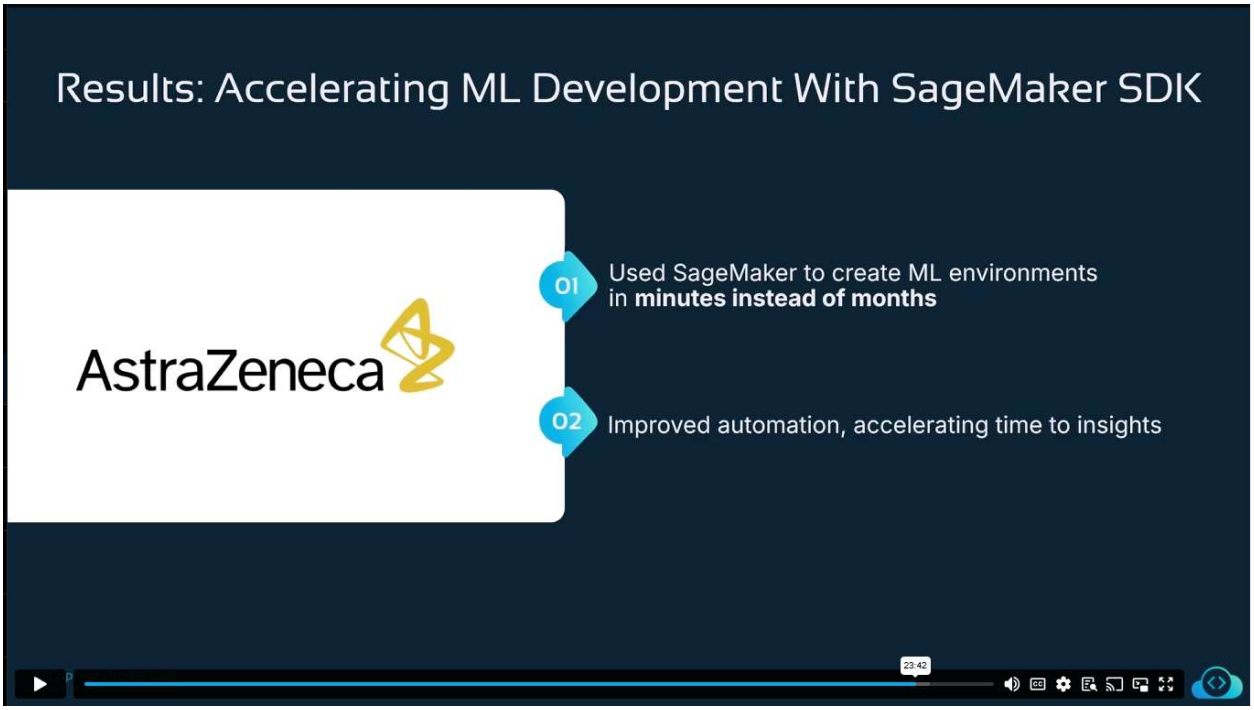

SageMaker SDK – Interacting With AWS Services
Amazon Simple Storage Service (Amazon S3)
Data upload/download, input/output management, checkpoint storage
AWS Identity and Access Management (IAM)
Managing execution roles for jobs, endpoints, and pipelines
AWS Step Functions
Building and orchestrating pipelines
16:26


SageMaker SDK – Interacting With AWS Services
SageMaker Feature Store
Ingesting, managing, and querying features
Amazon CloudWatch
Automatic logging and monitoring for jobs and endpoints
Amazon SageMaker Model Registry
Registering, managing, and deploying models
17:43

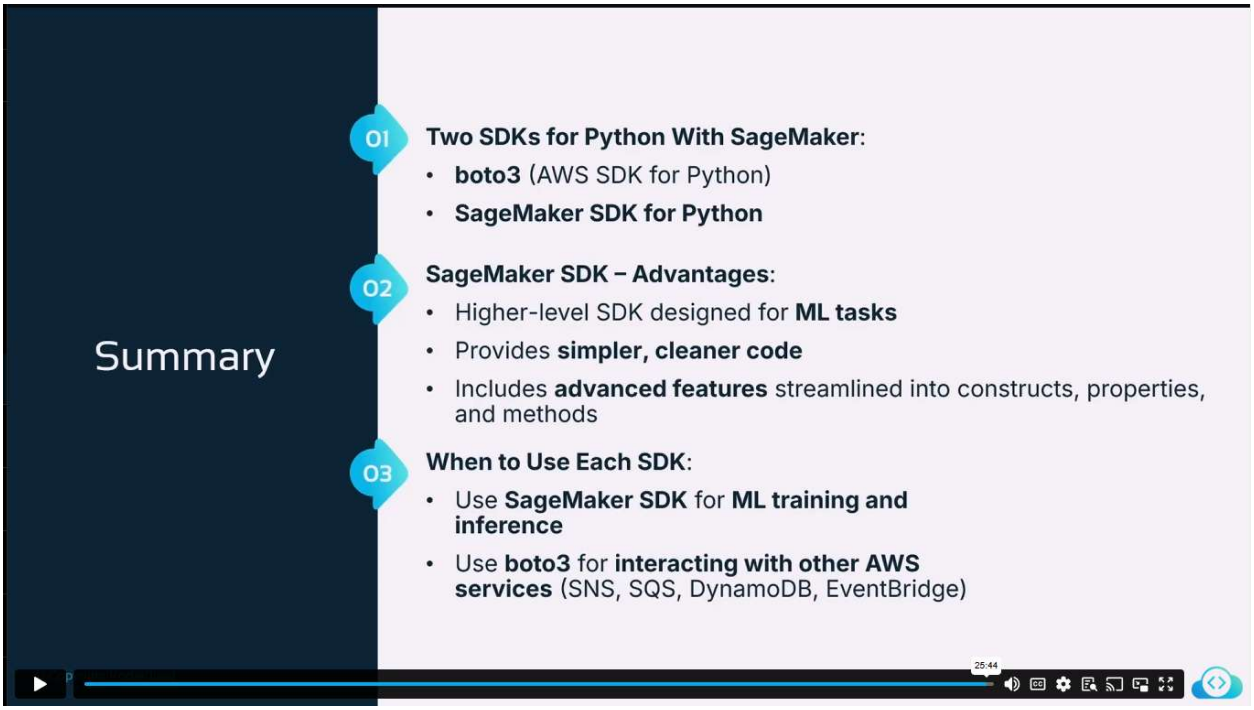

SDK Hybrid Approach
1
Use boto3 to send data to an SNS destination
2
Modify DynamoDB entries with boto3
3
boto3 enables access to various AWS services
19:46


hybrid.py
import boto3, sagemaker
session = sagemaker.Session()
role = "arn:aws:iam::123456789012:role/SageMakerExecutionRole"
estimator = sagemaker.estimator.Estimator(
image_uri="683313688378.dkr.ecr.us-east-1.amazonaws.com/linear-learner:latest",
role=role,
instance_count=1,
instance_type="ml.m5.large",
output_path="s3://your-bucket/path/to/output/",
sagemaker_session=session
)
estimator.set_hyperparameters(feature_dim=10, mini_batch_size=32,
predictor_type="regressor")
training_job_name = estimator.fit({"train": "s3://your-bucket/path/to/training-
data.csv"}, wait=False)
sns = boto3.client("sns")
sns.publish(
TopicArn="arn:aws:sns:us-east-1:123456789012:training-job-notifications",
Message=f"Training job {training_job_name.job_name} started.",
Subject="SageMaker Training Job Notification"
)
21:56


Results: Accelerating ML Development With SageMaker SDK
01
Faster ML development
02
Easier code maintenance
03
Shorter time to value
04
More efficient management
23:06


Results: Accelerating ML Development With SageMaker SDK
AstraZeneca
01
Used SageMaker to create ML environments in **minutes instead of months**
02
Improved automation, accelerating time to insights
23:42


Results: Accelerating ML Development With SageMaker SDK
NatWest
01
Built 30+ ML use cases in four months
02
Enabled personalized marketing and fraud detection
24:09


Summary
01
Two SDKs for Python With SageMaker:
• boto3 (AWS SDK for Python)
• SageMaker SDK for Python
02
SageMaker SDK – Advantages:
• Higher-level SDK designed for ML tasks
• Provides simpler, cleaner code
• Includes advanced features streamlined into constructs, properties, and methods
03
When to Use Each SDK:
• Use SageMaker SDK for ML training and inference
• Use boto3 for interacting with other AWS services (SNS, SQS, DynamoDB, EventBridge)
25:44