Speaker 1 (00:00):

Podcast confirmed. Welcome to the official AWS podcast.

Hawn Nguyen-Loughren (00:06):

Hello everyone and welcome back to the Official AWS podcast. We got some exciting updates with AWS Glue for Ray. I'm Hawn Nguyen-Loughren, also known as Han Solo, your friendly neighborhood co-host of the Official AWS podcast. And I'm joined by Kinshuk Pahare, welcome.

Kinshuk Pahare (00:22):

Hi. Thank you very much.

Hawn Nguyen-Loughren (00:24):

Cool. So before we get started, can you tell us a little bit about yourself and what do you do at Amazon?

Kinshuk Pahare (00:29):

Awesome. So my name is Kinshuk Pahare. I'm a principal product manager on AWS Glue. I've been with AWS for now three and a half years, and I manage a team who is responsible for platform and the engine for AWS Glue.

Hawn Nguyen-Loughren (00:48):

Nice. Well, we're here to discuss a really neat service update called AWS Glue, which makes it easier to scale Python code to process large scale data in AWS Glue. So AWS Glue for Ray lets data analysts and engineers run their existing Python code at scale with a pay as you go pricing. And because AWS Glue is serverless, there's no infrastructure to manage. So before we get started, tell us a little bit about AWS Glue.

Kinshuk Pahare (01:15):

Awesome, thank you. Yeah, so AWS Glue is a serverless data integration service. And the idea behind a serverless data integration service is that we actually take care of all the infrastructure and operational complexity and allow customers to focus on authoring their data integration logic in the favorite tool of their choice. And the new addition that we did with Ray is in line with helping more users, author, data integration logic in the language of their choice.

Hawn Nguyen-Loughren (01:47):

So what customer problem does Glue solve exactly?

Kinshuk Pahare (01:50):

So Glue helps customers integrate data from a variety of data sources. Now, what has happened in last few years is that data integration itself has seen rapid evolution. Customers want to improve the quality and the velocity of their decision making. And the first step in achieving that is to ensure that we can integrate data from a variety of sources into either a data lake or data warehouse, depending on customer preference.

(02:25):

Now with Glue, the way we solve this problem is that we actually help customers integrate and connect to over 80 data sources and manage your data in a centralized data catalog. And then you can actually make it part of the ETL, which is Extract, transform and Load or ELT, which is extract, load and transform. So we support both patterns from a broad variety of sources. So that's how Glue solves the first problem of helping customers integrate data from a variety of sources.

(02:55):

Now the second problem and second sort of aspect of improving the quality and the velocity of decision making is democratization of the data access. I like this phrase because it's sort of crisply summarized the fact that you want to enable access to the data for a variety of users across the skillset and across the tool set of their choice. For example, you have some users who are very comfortable writing code, they're comfortable authoring job in a Spark or Python. Then you have users who are comfortable writing code in SQL or some users who just prefer the visual way. So with the democratization of data access, you basically want to make sure that users across the skill and the tool spectrum are able to access the data to ensure that they actually help contribute in the decision making, whether it is improving the quality of the decision or improving the velocity of the decision.

Hawn Nguyen-Loughren (03:54):

Gotcha. So really helping with the ingestion, curation and eventually getting that insight. And I love democratizing the data for sure to really empower your users. So in terms of what we're releasing, so what is AWS Glue for Ray exactly about?

Kinshuk Pahare (04:10):

Yeah, so AWS Glue for Ray is a new data integration engine option in Glue. And the idea is that today we support Apache Spark and Python Shell and AWS Glue for Ray is the third engine that will help Python users scale their Python code to support large amount of data. It is using the same pay as you go pricing model, it is serverless, which means you don't have to worry about managing your infrastructure, you can focus your organizational energy on just authoring your business logic.

Hawn Nguyen-Loughren (04:49):

So what were some of the motivations to add Ray engine?

Kinshuk Pahare (04:53):

Yeah, so at Amazon we work backwards from customer and what we have seen is that Python obviously is one of the most popular programming language because it's just very easy to get started. The best part is that the easy tasks are easy and then the complex tasks are possible in Python. So you can actually go from very simple tasks to most complex tasks and it just flows very naturally. As a developer, it's very easy to get started. For those users, Glue offers a single node Python engine and the idea behind single node Python engine is that it's very useful if you want to process small amount of data. The engine is serverless and obviously it offers support for popular analytics libraries and has connected support for popular data sources like Redshift and S3. Customers love the serverless Python engine, they can offer simple or complex job and some of these Python job runs for hours, but the features, the features that make it easier to write Python code, they become, quote, unquote, "bug" when it comes to scaling Python beyond a single node.

(05:59):

So case in point, Python has a global interpreter log, it's GIL. Now it's a Python feature that allows developer to not worry about memory management. That feature makes it difficult to scale Python beyond a single machine. Now the problem that occurs, like if you cannot scale Python beyond a single machine, and if you are tasked with processing terabytes of data or petabytes of data, your options suddenly become very limited.

(06:29):

So either you have an option of let's just buy the most expensive and bigger machine that we can, which means that it's just going to cost you more. Or you basically process data in manageable chunks, which will slow down your data ingestion process and slow down your data processing because you obviously cannot process and your single node processing becomes the bottleneck or it becomes very complex also. So suppose if one of the job fails, you have to now redo the entire pipeline or build a complex logic so that you can account for the lost data. So to address this problem, we evaluated multiple engines and decided to add Ray.io, which is an open source unified compute framework that makes it easy to scale in Python jobs.

Hawn Nguyen-Loughren (07:17):

Gotcha. So let's dive a little bit deeper on Ray.io. Like you said, it was open source framework to make it easier for AI and Python workload. So what is it exactly that we're using from that?

Kinshuk Pahare (07:29):

So the best part is that Ray.io is open source, it's flexible. If you are a native Python developer, you will find coding and writing your data integration job on Ray.io. very familiar. What I've heard from few developers is they call it like it's pythonic way of distributing your single note Python jobs. And on top of that, Ray is a task parallel asynchronous engine. So the distinction is that you have the data parallel engines like Spark, and then you have the task parallel engine like Ray.io, which means that with Ray.io you can actually create a highly paralleled application and you can create multiple tasks which can operate asynchronously and that leads to creating very high throughput application that you can build using Ray.io.

Hawn Nguyen-Loughren (08:22):

That's really cool. I love the way that we democratize data, but also leveraging open source to democratize development. Super cool. So how is Glue for Ray different from other data integration engine already offered by Glue?

Kinshuk Pahare (08:36):

Yeah, that's a very good question. So the idea behind offering a variety of data integration engine is to offer choice. With the introduction of Ray, now we have three separate data integration engines. Obviously Spark, spark is the most dominant data integration engine that we offer. It's the most popular framework for data integration and data processing. And in addition to Spark, we have the Python Shell engine that I just talked about. And with the introduction of Ray, now you have a choice where you can actually now write task parallel application using Ray.io engine.

(09:11):

And all three engines are serverless. So you have the same basic primitives that you can use across the three engines. So let me sort of double click on the primitive. So what we do is we have two basic primitives that we offer on Glue. One is called jobs primitive. And the way I describe jobs primitive is it's a fire and forget system where you can author your data integration transform logic, which could be as simple as ingest data from S3, drop couple of columns, apply some data quality checks, do some sensitive data detection or PII detection, or save the data into Data Lake or into data warehouse.

(09:52):

Now that's a very simple pipeline. Now you can achieve this pipeline by authoring this in Python, Spark and Ray. And then once you have authored this, you can submit this as form of a Glue job and you can schedule the job to run on a schedule or you can run it recurringly or based on certain events. So that's the fire and forget system. The best part is that here you don't have to think about creating and managing infrastructure or how many nodes do we need and kind of compute that we need. Glue takes care of everything.

(10:27):

The second primitive that we offer is interactive primitive, and it's more like coffee with the data. And the idea behind coffee with the data is that you can actually in the real time observe the reserves of your data transforms as you apply them. For example, you can open your favorite notebook or IDE, which is integrated development environment and connect it with the interactive session's API, which Glue offers.

(10:58):

You can also use SageMaker Studio Notebook and Glue Studio Notebook, which is built in notebook interface that we offer. And regardless of the interface, what you get is an interactive experience where you can say, you know what, let's apply a group by transform to the data. And as soon as you apply group by transform, you can immediately see the results on your Jupyter Notebook or your Glue Studio Notebook or SageMaker. And you can also, with the notebook, you can pick the engine that you prefer. So you can say, you know what, I want to run this on Spark because I'm about to submit Spark code, or I can run this on Ray.io because I'm about to submit a Ray.io. And regardless of the interface, you get a very, very fast start time. You can create hundreds of nodes of cluster, Spark cluster, Ray cluster in matter of seconds, and you can also enable automatic scaling.

(11:52):

So what will happen with automatic scaling is that we will right size the compute so that you have the most optimum cost experience. And when we right size the compute, we only bill you for the compute hour that is used. And again, all of this is possible with just a simple check mark on the job or the interactive perimeter. And these are some of the enhancement, which means that you don't have to worry about infrastructure management or scheduling, auto scaling or dealing with some complex files to tune your cluster.

Hawn Nguyen-Loughren (12:21):

So can you walk me through how I would use Glue for Ray?

Kinshuk Pahare (12:25):

How do you get started with Ray, like in this case, what will happen is you identify what business logic you want to run. Suppose you want to process a hundred kicks off data from S3 to S3 and in the form of a data lake, what you will do is you will author your job in Ray.io and once you have authored it, you will submit this in the form of a job and submit this into the form of job, it's just a very simple step. You add your code into an S3 bucket and tell Glue where to pick the code and then glue will take care of the rest. So what we'll do is we will pick up the code from your preferred location and we will automatically scale the cluster depending on the compute requirement for that job. And what that does is that you are only responsible for authoring your data integration job and Glue takes care of the rest.

(13:18):

So we spin up the head node, we spin up the worker nodes, and then we scale those worker nodes down if they're not processing any data. So that's one of the biggest benefit of enabling auto scaling is that we identify the idle workers, the workers which are not processing any data, are not processing any task, and we shut those things down. And the best part is that with Ray, these things happen in fraction of second. So in millisecond we will identify those things and then identify idle worker and shut those down. Or suppose on the other side, if you want to scale your cluster up, suppose the Ray driver needs 10 new nodes to process the next set of transforms, within a second we will be able to add 10 more nodes to the cluster, automatically scale it up, process the task, and then scale those down back.

(14:09):

As a developer, you don't have to worry about creating the cluster or managing the infrastructure that comes with sub submitting a Ray job. And because we can do this in a sub-second automatic scaling, you're able to squeeze the maximum benefit of your compute cluster without having to rely on complex coding logic. And once the job finishes, we automatically shut down all the instances and we stop your billing.

Hawn Nguyen-Loughren (14:42):

Love it. I love that serverless factor of it to provide that undifferentiated heavy lifting. Super cool. So Kinshuk, thank you so much for coming to the podcast today.

Kinshuk Pahare (14:51):

Thank you very much for having me.

Hawn Nguyen-Loughren (14:53):

And as always, we'd love to get your feedback. There's a link in the show notes to submit feedback. And until next time, keep on building.