AWS re:Invent

SVS325-R

Serverless big data processing

Doug Gartner

Solutions Architect

Amazon Web Services

Serverless review

No infrastructure provisioning, no management

Automatic scaling

Pay for value

Highly available and secure

AWS Lambda and AWS Fargate

AWS Lambda

Serverless event-driven code execution

Short-lived

All language runtimes

Data source integrations

AWS Fargate

Serverless compute engine for containers

Long-running

Bring existing code

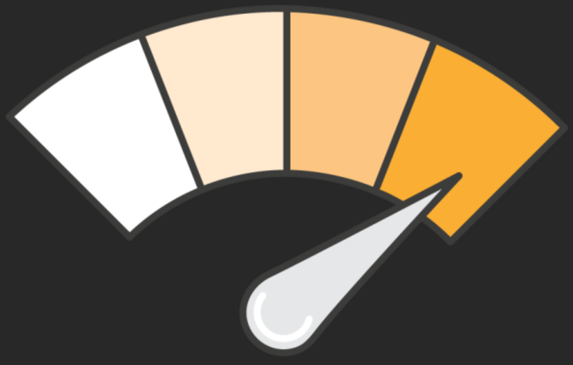
Fully managed orchestration

AWS Lambda best practices

- Know the limits and concurrency behavior
- Minimize package size to necessities
- Avoid using recursive code in your Lambda function
- Use environment variables to modify operational behavior
- Self-contain dependencies in your function package
- Consider use layers for reuse
- Delete large unused functions (75-GB limit)

https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html

Tweak your function’s computer power



Lambda exposes only a memory control, with the % of CPU core and network capacity allocated to a function proportionally

Is your code CPU, network, or memory-bound? If so, it could be cheaper to choose more memory.

AWS Glue components

Data catalog

- Hive Metastore compatible with enhanced functionality
- Crawlers automatically extract metadata and create tables
- Integrated with Amazon Athena, Amazon Redshift Spectrum

Job authoring

- Automatically generates ETL code
- Build on open frameworks (e.g., Python and Spark)
- Developer-centric: editing, debugging, sharing

Job execution

- Runs jobs on a serverless Spark platform
- Provides flexible scheduling
- Handles dependency resolution, monitoring, and alerting

What is an AWS Glue job?

An AWS Glue job encapsulates the business logic that performs extract, transform, and load (ETL) work

- A *core building block* in your production ETL pipeline
- Provide your PySpark ETL script or *have one automatically generated*
- Supports a *rich set of built-in AWS Glue transformations*
- Jobs can be *started, stopped, monitored*

What is an AWS Glue trigger?

Triggers are the “glue” in your AWS Glue ETL pipeline

Triggers

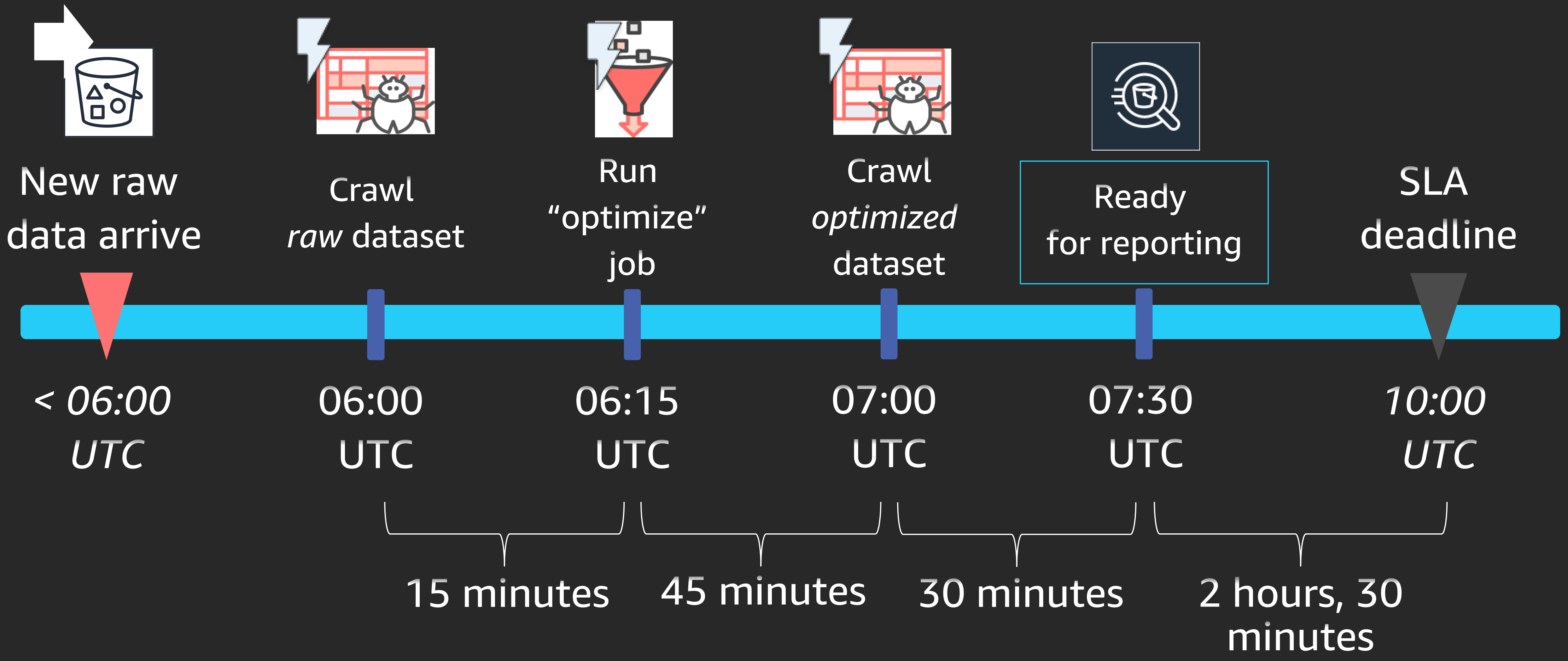
- Can be used to *chain* multiple AWS Glue jobs in a series
- Can start *multiple jobs at once*
- Can be *scheduled, on-demand*, or based on *job events*
- Can *pass unique parameters* to customize AWS Glue job runs

Three ways to set up an AWS Glue ETL pipeline

- *Schedule*-driven
- *Event*-driven
- *State machine*–driven

Schedule-driven AWS Glue ETL pipeline

We work our way backward from a daily SLA deadline



New raw data arrive — *< 06:00 UTC*

Crawl *raw* dataset — 06:00 UTC

Run “optimize” job — 06:15 UTC

Crawl *optimized* dataset — 07:00 UTC

Ready for reporting — 07:30 UTC

SLA deadline — *10:00 UTC*

15 minutes

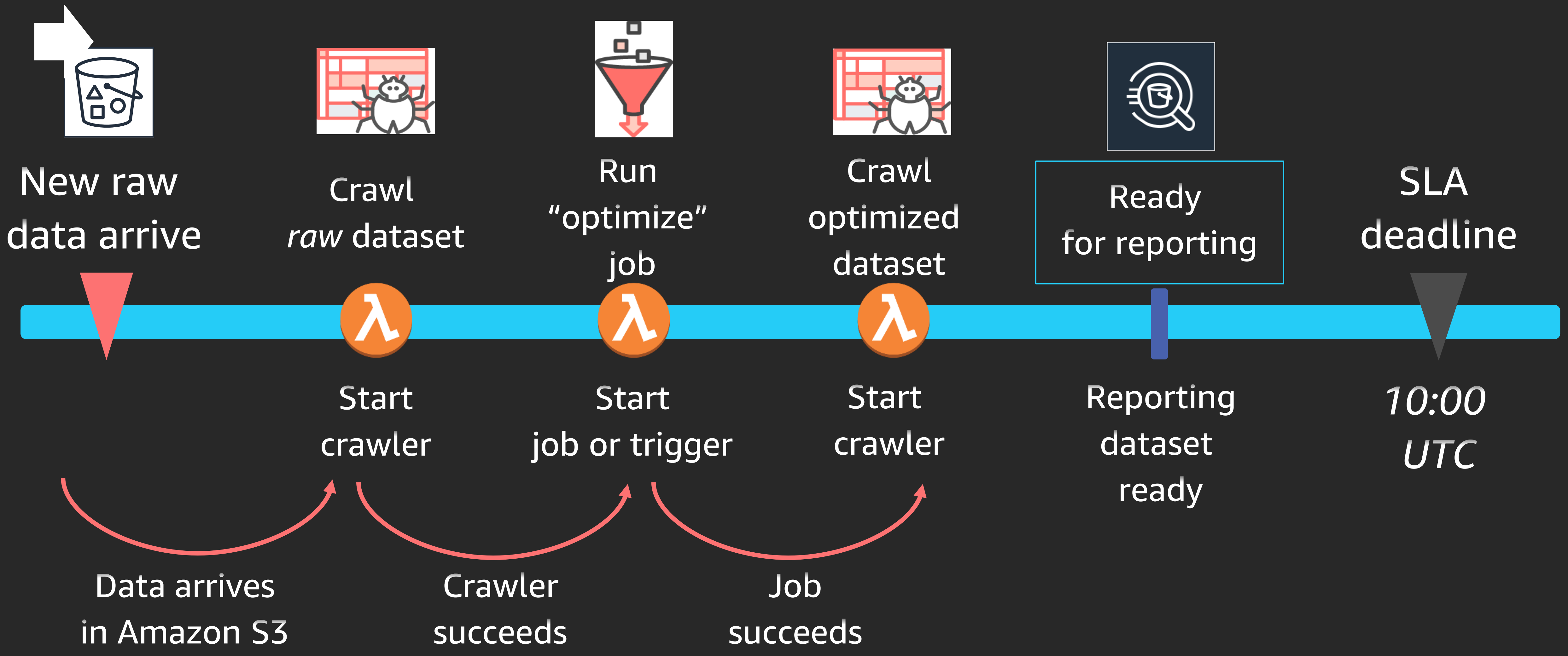
45 minutes

30 minutes

2 hours, 30 minutes

Event-driven AWS Glue ETL pipeline

Let Amazon CloudWatch Events and AWS Lambda drive the pipeline

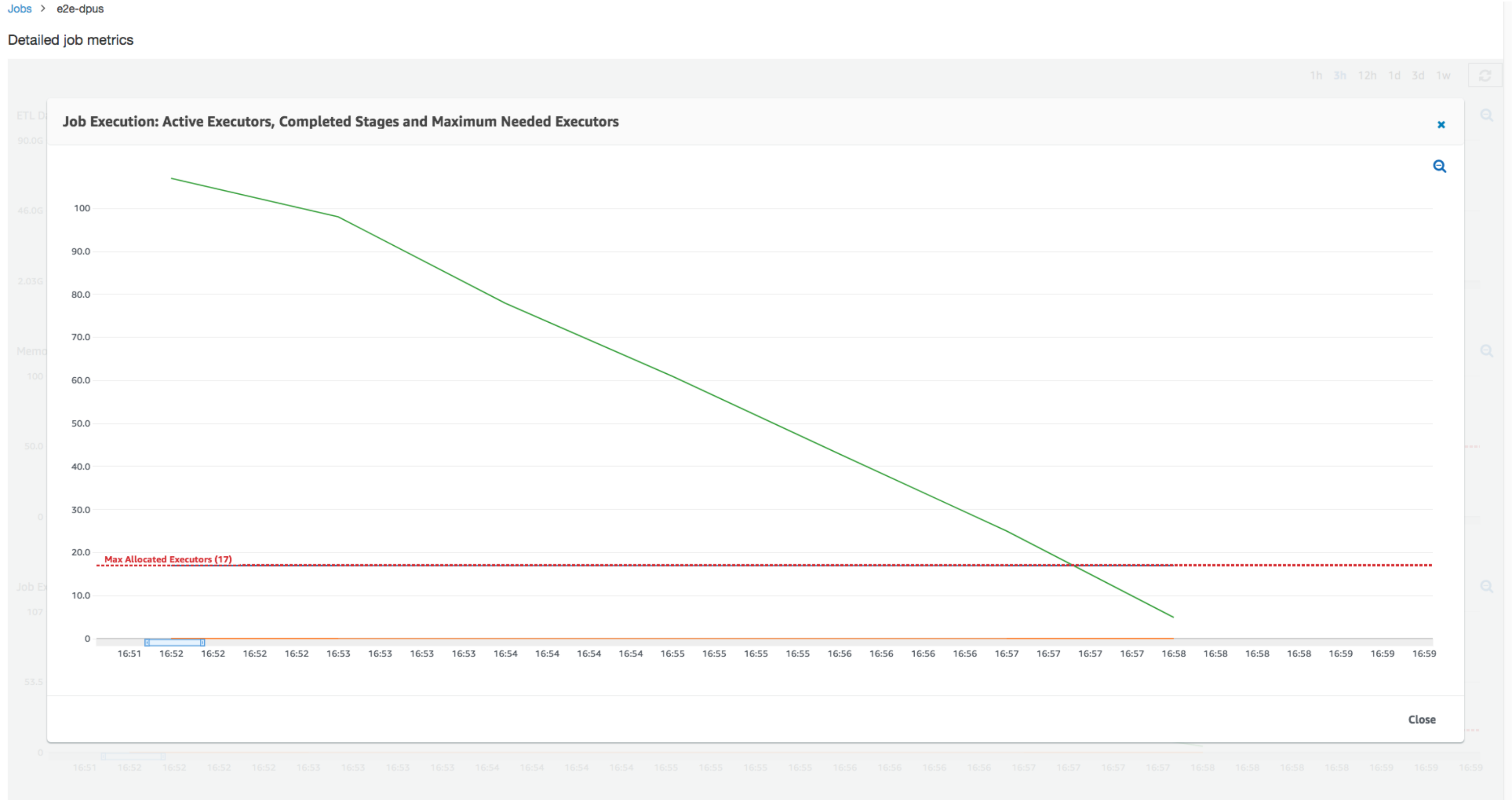


Serverless optimization

AWS Glue data processing units

Job execution in AWS Glue

- Number of actively running executors
- Number of completed stages
- Number of maximum needed executors



Right tool for the right job

When to use AWS Glue versus AWS Lambda versus Amazon EMR?

- Size of data?

  If your data volume isn’t heavy, don’t overengineer

- Frequency of data ingest?

  Is the data analysis fairly constant and consistent, or does it come in on regularly scheduled intervals (e.g., 1 hour)?

- In-line analysis?

  Do you need to perform streaming analysis of the data? (see Amazon Kinesis Data Analytics)

Downstream datastores

When to use a different datastore

- Amazon S3 is an excellent "catch-all"

- Use data characteristics and metrics to determine when to use Amazon Redshift, Amazon Relational Database Service (Amazon RDS), or another option

- Work backward from your main objectives while remaining flexible

Amazon Simple Storage Service (Amazon S3)

Amazon RDS

Amazon Neptune

Amazon Elasticsearch Service (Amazon ES)

AWS Lake Formation

Amazon DynamoDB

Amazon DocumentDB (with MongoDB compatibility)

Amazon Redshift

Serverless query and analysis

Amazon Athena

- Optimize for storage, optimize for compute
- Use Amazon Redshift Spectrum if your queries are computationally heavy and need to take advantage of active cluster memory
- Use approximate functions for exploratory analysis

Amazon QuickSight

- Iterate on exploratory analysis with eventual publishing to dashboards
- Leverage Cross Source Join when ad-hoc analysis is necessary
- Use Templates for common dashboards

Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development

Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures

Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.

AWS re:Invent

aws

Python libraries

Numpy, SciPy, and Pandas

Lambda layer published to support both

Lambda Runtime API

```
import numpy as np

from scipy.spatial

import ConvexHull

def lambda_handler(event, context):
```

Smart resource allocation

Match resource allocation (up to 3 GB) to logic

Stats for Lambda function that calculates 1000 times all prime numbers up to 1,000,000

128 MB	11.722965 sec	$0.024628
256 MB	6.678945 sec	$0.028035
512 MB	3.194954 sec	$0.026830
1024 MB	1.465984 sec	$0.024638

Green = best Red = worst

Smart resource allocation

Match resource allocation (up to 3 GB) to logic

Stats for Lambda function that calculates 1000 times all prime numbers up to 1,000,000

128 MB	11.722965sec	$0.024628
256 MB	6.678945sec	$0.028035
512 MB	3.194954sec	$0.026830
1024 MB	1.465984sec	$0.024638

−10.256981 sec

+$0.00001

Green = best

Red = worst