Architecture Optimizations for Week 1

In this reading, you will find further information about the topics that Morgan and Raf talked about in the video where they both played Solutions Architects.

Caching for Amazon DynamoDB by using Amazon DynamoDB Accelerator

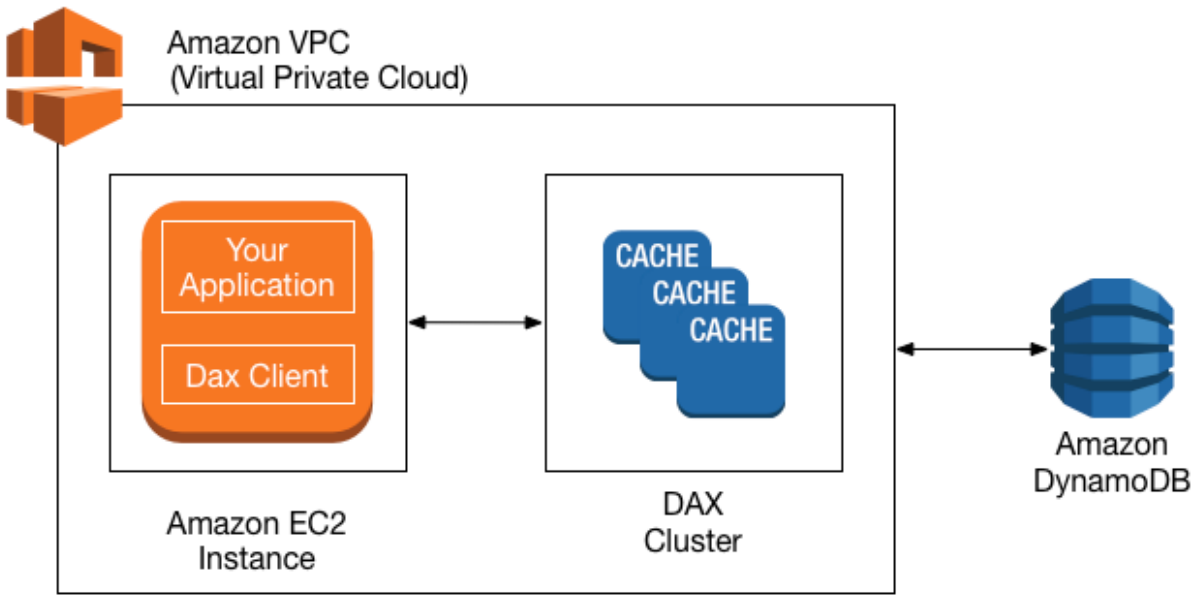
Raf mentioned that if you need to improve Amazon DynamoDB performance to microsecond latency, to look into using Amazon DynamoDB Accelerator (DAX).

DAX is a fully managed, highly available, in-memory cache for DynamoDB that's designed to deliver up to a 100-times performance improvement—from milliseconds to microseconds—even at millions of requests per second.

DAX does the heavy lifting that's required to add in-memory acceleration to your DynamoDB tables—and developers don't need to manage cache invalidation, data population, or cluster management.

A benefit of using DAX is that you don't need to modify the application logic because DAX is compatible with existing DynamoDB API calls.

DAX is designed to run within an Amazon Virtual Private Cloud (Amazon VPC) environment. Amazon VPC defines a virtual network that closely resembles a traditional data center. With a VPC, you have control over its IP address range, subnets, routing tables, network gateways, and security settings. You can launch a DAX cluster in your virtual network, and control access to the cluster by using Amazon VPC security groups.



For more information about DAX, see Amazon DynamoDB Accelerator.

Optimizing AWS Lambda

AWS Lambda Power Tuning

Morgan mentioned that you can fine-tune the memory or power configuration for your AWS Lambda functions to potentially increase performance and lower costs.

AWS Lambda Power Tuning is an open-source tool that helps you visualize and fine-tune the memory or power configuration of Lambda functions. It runs in your own AWS account, and it supports three optimization strategies: cost, speed, and balanced.

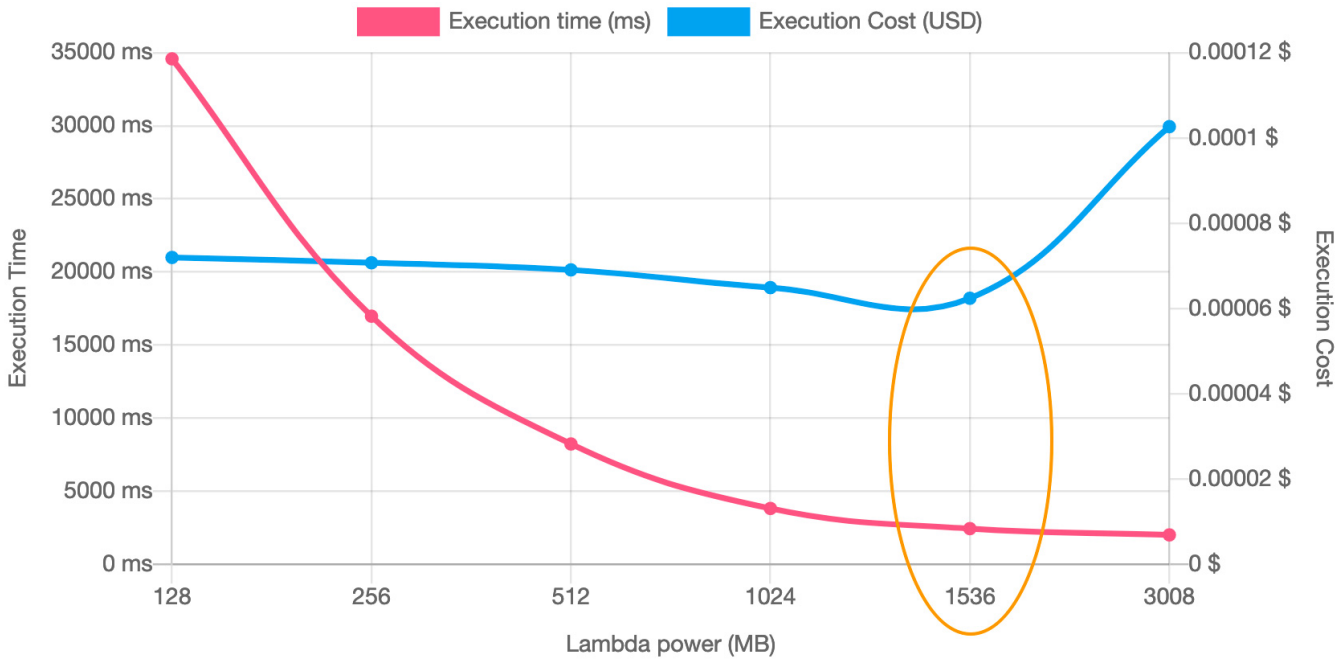
AWS Lambda Power Tuning is a state machine that's powered by AWS Step Functions. It helps you optimize your Lambda functions for cost or performance in a data-driven way.The state machine is designed to be easy to deploy and fast to execute. Also, it's language agnostic, so you can optimize any Lambda functions in your account.

To work with AWS Lambda Power Turning, you provide a Lambda function Amazon Resource Name (ARN) as input. The state machine then invokes that function with multiple power configurations (from 128 MB to 10 GB—you decide which values). Then, it analyzes all the execution logs and suggests the best power configuration to minimize cost or maximize performance.

Note that the input function will run in your AWS account, which means that it will perform HTTP requests, SDK calls, cold starts, and so on. The state machine also supports cross-Region invocations, and you can enable parallel execution to generate results in a few seconds.

The state machine generates a visualization of average cost and speed for each power configuration.

For example, the following diagram shows results for two CPU-intensive functions, which become both cheaper and faster with more power.



For more information, see AWS Lambda Power Tuning.

AWS Lambda Powertools

Morgan also mentioned that you can use another suite of tools called AWS Lambda Powertools to optimize your Lambda functions and use best practices. AWS Lambda Powertools is a suite of utilities for AWS Lambda functions that is designed to make it easier to adopt best practices such as tracing, structured logging, custom metrics, idempotency, batching, and more.

For more information, see AWS Lambda Powertools.

AWS Lambda execution environment reuse

Another optimization technique that Morgan mentioned for Lambda is to move certain initialization tasks in your code so they are outside the handler. These tasks can then be reused across invocations (which is also known as execution environment reuse).

You can take advantage of execution environment reuse to improve the performance of your function. To do this, initialize SDK clients and database connections outside of the function handler, and cache static assets locally in the /tmp directory. Subsequent invocations that are processed by the same instance of your function can reuse these resources. This reuse saves cost by reducing function run time.

To avoid potential data leaks across invocations, don't use the execution environment to store user data, events, or other information with security implications. If your function relies on a mutable state that can't be stored in memory within the handler, consider creating a separate function or separate versions of a function for each user.

To learn more, see Best practices for working with AWS Lambda.

Resources

- For more information about how to optimize serverless applications for cost, see Building well-architected serverless applications: Optimizing application costs in the *AWS Compute Blog*.
- For more information about well-architected serverless applications, visit Serverless Applications Lens - AWS Well-Architected Framework.