

Concepts

Cold Start

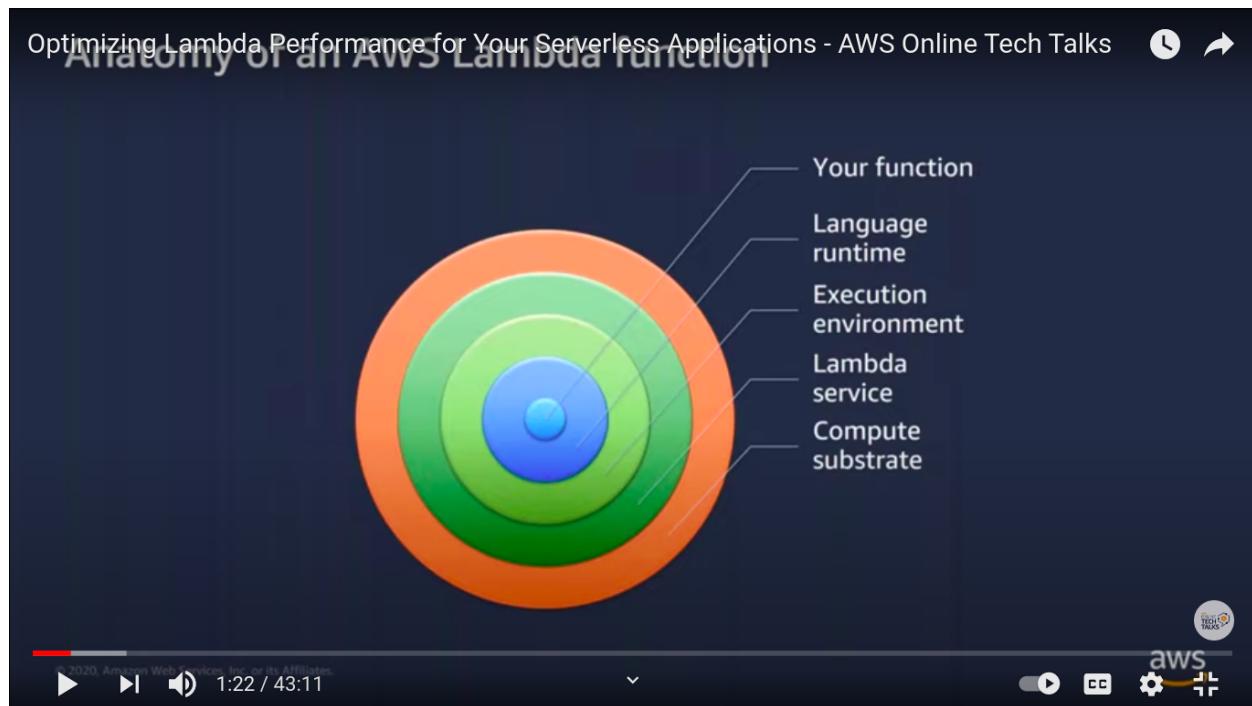
Memory and Profiling

Architecture of Lambda /Functions

Best Practices

How Lambda works

Pic Source : You Tube



Out of these 5 layers following layers have direct impact on execution

Lambda Service

Execution Environment

Your Function Code

Anatomy of AWS Lambda

Anatomy of an AWS Lambda function

Press Esc to exit full screen

Handler () function	Event object	Context object
Function to be executed upon invocation	Data sent during Lambda function Invocation	Methods available to interact with runtime information (request ID, log group, more)

```
// Python
import json
import mylib

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello World!')
    }
```

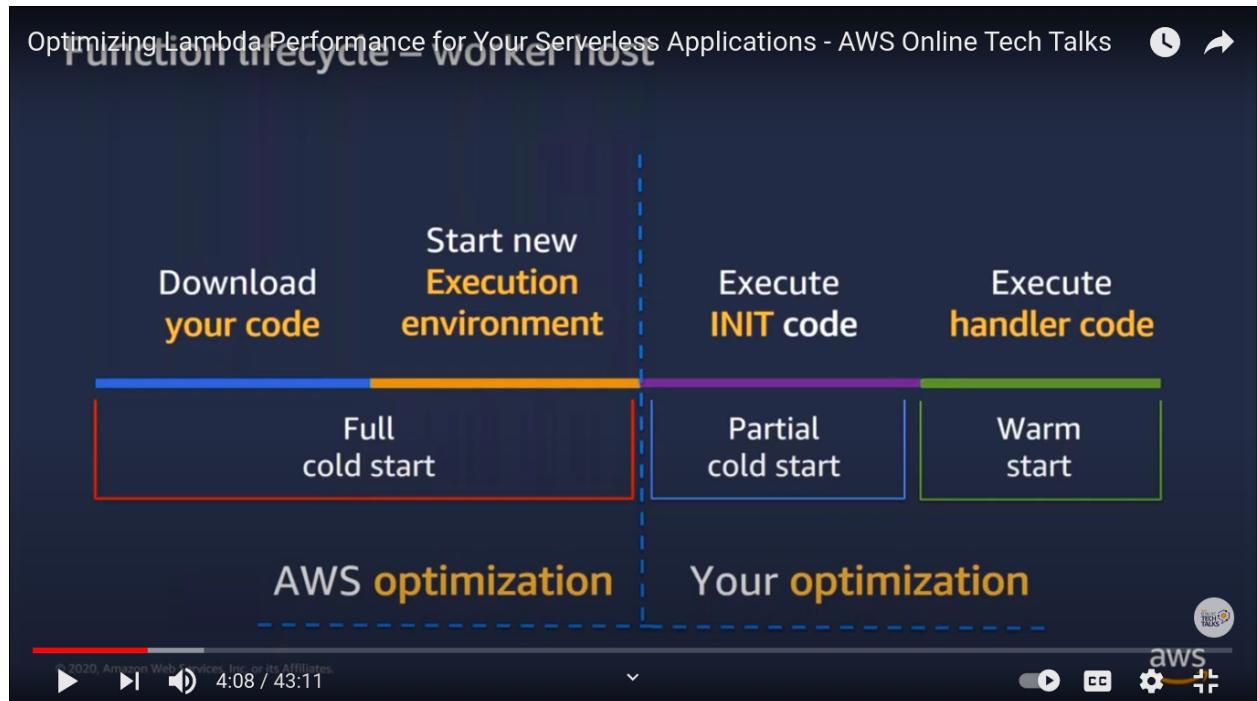
```
// Node.js
const MyLib = require('my-package')
const myLib = new MyLib()

exports.handler = async (event, context) => {
    # TODO implement
    return {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!')
    }
}
```

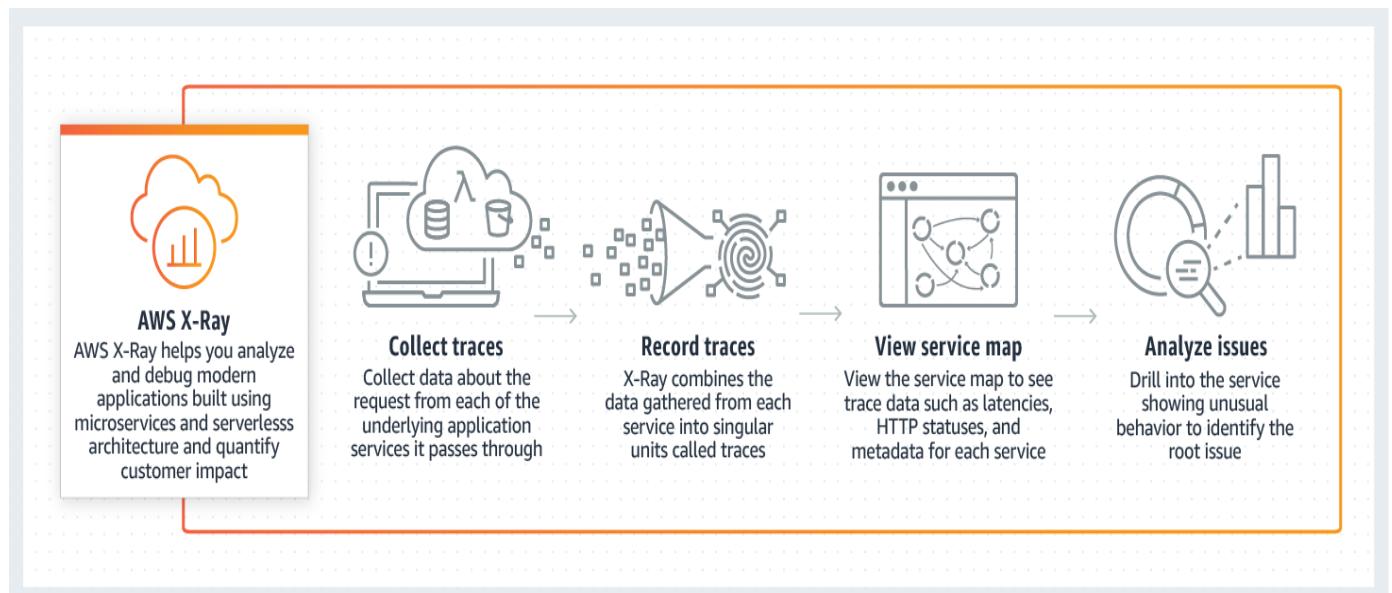
© 2020, Amazon Web Services, Inc. or its Affiliates.



Lambda Function LifeCycle



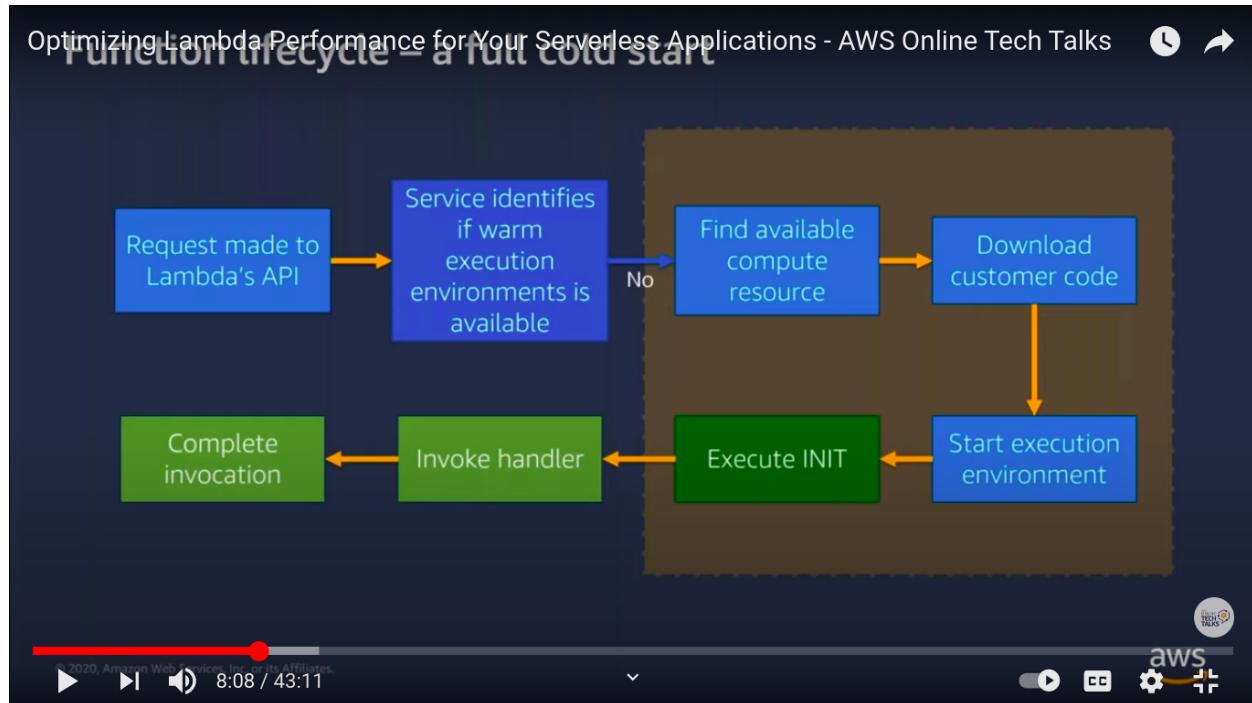
Measuring with AWS X-Ray



Area of Optimization

1. Latency
2. Throughput
3. Cost

Function Lifecycle : Full Cold Start



Cold Start usually take time in rage of 100 ms to 1 sec

We can not manage Warm environment : Managed by AWS

Pinging function to keep it warm is limited

Cold Start Execution Environment Influenced by

- Memory Allocation
- Function Size
- Frequency of Function Call
- Internal Algorithm

Lambda + VPC improve performance

Optimizing Lambda Performance for Your Serverless Applications - AWS Online Tech Talks

Lambda + VPC – Major performance improvement

AWS Compute Blog

Announcing improved VPC networking for AWS Lambda functions

by Chris Munns | on 03 SEP 2019 | In Amazon VPC, AWS Lambda, Serverless | Permalink | Comments | Share

Method Response Duration Age ID

internet-access 200 14.8 sec 1.5 min (2019-08-06 17:59:17 UTC) 1-5d49bf5-73c368a004d56a805680a3e0

Name Res. Duration Status 0.0ms 2.0ms 4.0ms 6.0ms 8.0ms 10ms 12ms 14ms 16ms

internet-access AWS Lambda

internet-access 200 14.8 sec

internet-access AWS Lambda Function

internet-access - 511 ms

Initialization - 572 ms

Invocation - 491 ms

Overhead - 1.2 ms

Timeline Raw data

Method Response Duration Age ID

internet-access 200 933 ms 52.1 sec (2019-08-06 18:12:12 UTC) 1-5d49c2fc-82899913a8dc997e71c6352a

Name Res. Duration Status 0.0ms 100ms 200ms 300ms 400ms 500ms 600ms 700ms 800ms 1.0s

internet-access AWS Lambda

internet-access 200 933 ms

internet-access AWS Lambda Function

internet-access - 495 ms

Initialization - 167 ms

Invocation - 456 ms

Overhead - 39.2 ms

← Before: 14.8 sec duration

After: 933ms duration →

Read more at <https://bit.ly/vpc-lambda>

Play (k)

2020, Amazon Web Services, Inc. or its Affiliates.

12:16 / 43:11

Cold Start : Static Initializer (INIT)

Facts :

- Managed by Developer
- Code Run Before Handler
- Used to establish connection
- Have much impact on Cold Start
- Occur when Lambda execute first time
- Also execute when Scaling Up lambda

Influenced By :

- ❖ Size of Function package
- ❖ Function Code
- ❖ Initialization work

What can help :

Code Optimization

- Trim SDK
- Reuse Connection
- Don't Load if not used
- Lazily load variables
-

Provisioned Concurrency

The screenshot shows a presentation slide with the following content:

Cold starts - Static Initialization

Optimizing Lambda Performance for Your Serverless Applications - AWS Online Tech Talks

Influenced by:

- Size of function package
- Amount of code
- Amount of initialization work

The developer is responsible for this part of a cold start.

What can help...

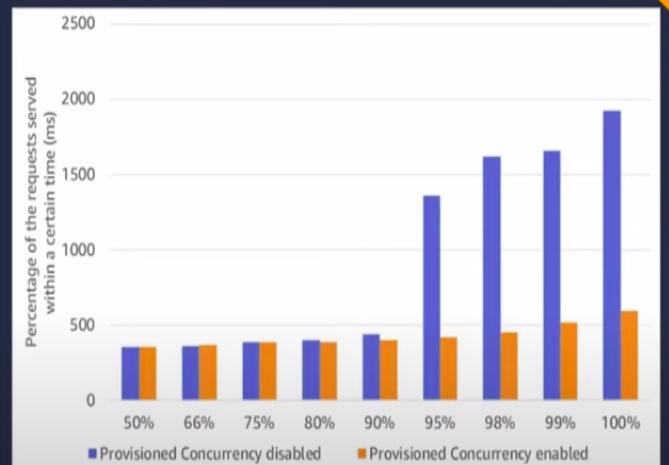
- Code optimization
 - Trim SDKs
 - Reuse connections
 - Don't load if not used
 - Lazily load variables
- Provisioned Concurrency

At the bottom of the slide, there is a navigation bar with icons for back, forward, and search, along with a timestamp (15:06 / 43:11) and an AWS logo.

Provisioned Concurrency

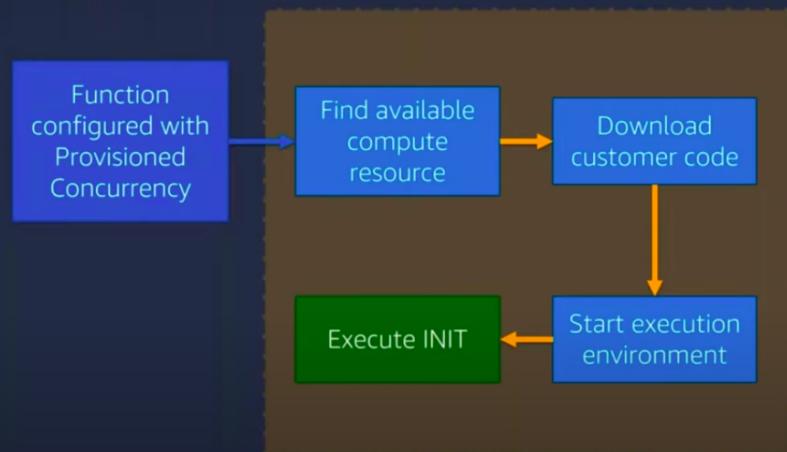
Pre-creates execution environments, running INIT code.

- Mostly for latency-sensitive, interactive workloads
- Improved consistency across the long tail of performance
- Minimal changes to code or Lambda usage
- Integrated with AWS Auto Scaling
- Adds a cost factor for per concurrency provisioned but a lower duration cost for execution
 - This could save you money when heavily utilized



Function Lifecycle with Provisioned Concurrency

Optimizing Lambda Performance for Your Serverless Applications - AWS Online Tech Talks



© 2020, Amazon Web Services, Inc. or its Affiliates

▶ | ⏪ | ⏹ | 16:37 / 43:11

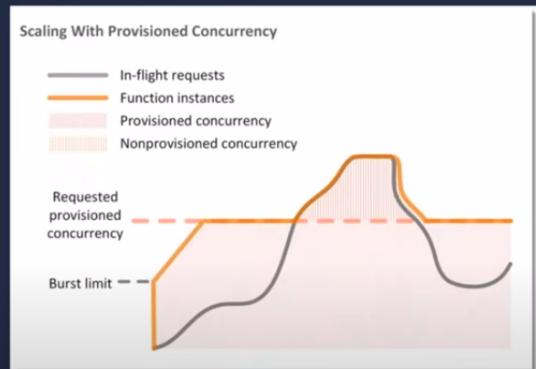


Provisioned Concurrency : Things to Know

Provisioned Concurrency - things to know



- Reduces the start time to <100ms
- Can't configure for \$LATEST
 - Use versions/aliases
- Provisioning rampup of 500 per minute
- No changes to function handler code performance
- Requests above provisioned concurrency follow on-demand Lambda limits and behaviors for cold-starts, bursting, pricing
- Still overall account concurrency per limit region
- Wide support (CloudFormation, Terraform, Serverless Framework, Datadog, Epsagon, Lumigo, Thundra, etc.)



© 2020, Amazon Web Services, Inc. or its Affiliates

▶ | ⏪ | ⏹ | 18:48 / 43:11



Memory and Profiling

128 MB to 3000 MB

The screenshot shows a video player interface for a AWS Online Tech Talks video titled "Optimizing Lambda Performance for Your Serverless Applications". The video is currently at 23:07 / 43:11. The main content displays a table comparing execution times and costs for different memory sizes:

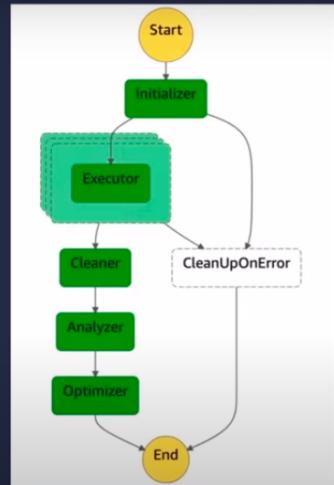
Memory Size	Execution Time	Cost
128 MB	11.722 sec	\$0.024628
256 MB	6.678 sec	\$0.028035
512 MB	3.194 sec	\$0.026830
1024 MB	1.465 sec	\$0.024638

The table highlights the 1024 MB row. The AWS logo is visible in the bottom right corner of the player.

AWS Lambda Power Tuning

Features:

- Data-driven cost and performance optimization for AWS Lambda
 - Available as an AWS Serverless Application Repository app
 - Easy to integrate with CI/CD



<https://github.com/alexcasalboni/aws-lambda-power-tuning>



© 2020, Amazon Web Services, Inc. or its Affiliates.



How billing rounding impact Cost Optimization

Example 1

Time = 480ms
Billing = 500ms

A 15% performance optimization...

Time = 408ms
Billing = 500ms

... leads to a 0% cost optimization

Example 2

Time = 310ms
Billing = 400ms

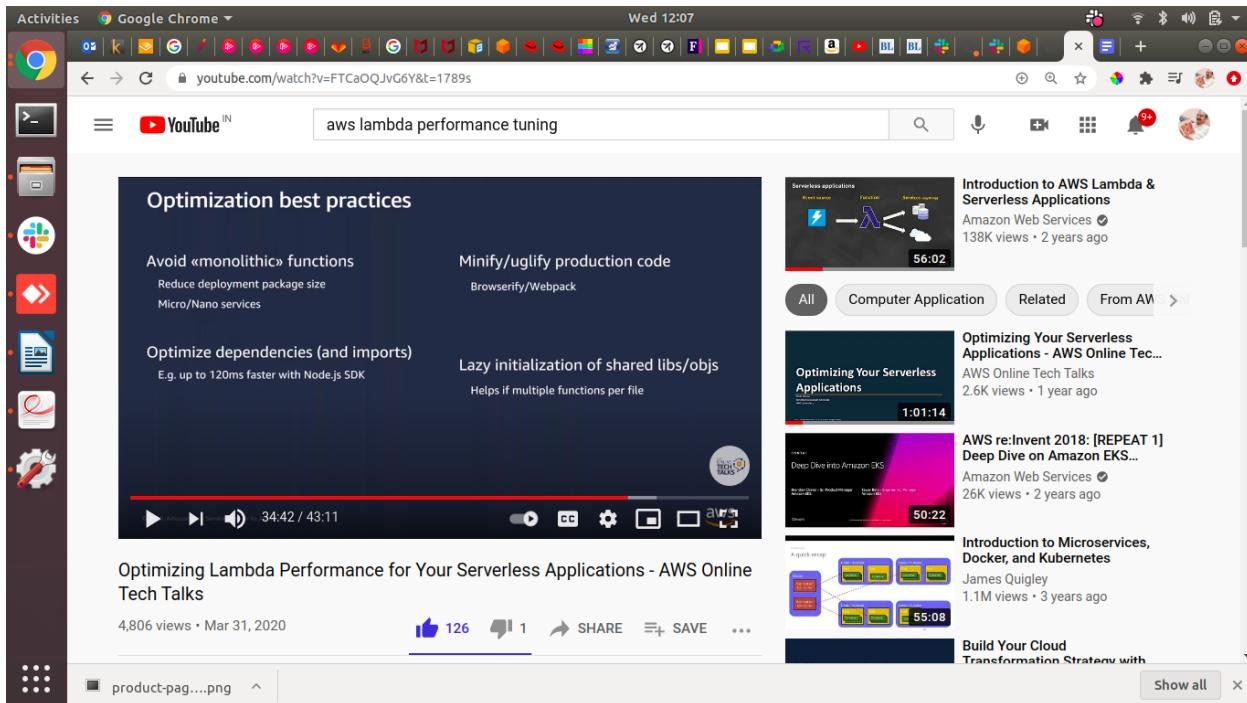
A 5% performance optimization...

Time = 294ms
Billing = 300ms

... leads to a 25% cost optimization

Architecture and Best Practises

Optimization best practices



Sync Vs Async

Optimizing Lambda Performance for Your Serverless Applications - AWS Online Tech Talks

Comparing sync and async

Synchronous:

- The caller is waiting
- Waiting incurs cost
- Downstream slowdown affects entire process
- Process change is complex
- Dependent on custom code

Asynchronous:

- The caller receives ack quickly
- Minimizes cost of waiting
- Queueing separates fast and slow processes
- Process change is easy
- Uses managed services

2020, Amazon Web Services, Inc. or its Affiliates

Summary :

The screenshot shows a video player interface for an AWS Online Tech Talks session titled "Optimizing Lambda Performance for Your Serverless Applications". The video is currently at 42:41 / 43:11. The content of the video is a presentation slide with the following sections and bullet points:

- Cold starts**
 - Causes of a cold start
 - VPC improvements
 - Provisioned Concurrency
- Architecture and optimization**
 - Best practices
 - RDS Proxy
 - Async and service integration
- Memory and profiling**
 - Memory is power for Lambda
 - AWS Lambda Power Tuning
 - Trade-offs of cost and speed

At the bottom of the slide, there is a footer bar with the AWS logo and the text "2020, Amazon Web Services, Inc. or its Affiliates".