
AWS
re:Invent

SVS405

AWS Lambda – Part 2: Optimizing your Lambda function performance

Alex Casalboni

Sr. Developer Advocate

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Fundamentals (quickly!)

Optimization best practices

Idea: Data-driven approach

Real-world examples and patterns



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Related breakouts

SVS403:
AWS Lambda – Part 1: Optimizing your serverless applications (Chris Munns)

Fundamentals



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



“So what does the future look like? All the code you ever write is business logic.”

Werner Vogels

CTO

Amazon.com

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Anatomy of a Lambda function

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda function Invocation

Context object

Methods available to interact with runtime information (request ID, log group, more)

```
import json

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

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
             result = SubfunctionA()
         }else {
                 result = SubfunctionB()

    return result;
}

Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}

Function subFunctionA(thing){
 ## logic here
}

Function subFunctionB(thing){
 ## logic here
}
```

Your handler

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
             result = SubfunctionA()
        }else {
                 result = SubfunctionB()

    return result;
}
```

Your handler

```
Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}

Function subFunctionA(thing){
 ## logic here
}

Function subFunctionB(thing){
 ## logic here
}
```

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
             result = SubfunctionA()
        }else {
                 result = SubfunctionB()

    return result;
}
```

Your handler

```
Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```

Common helper functions

```
Function subFunctionA(thing){
 ## logic here
}

Function subFunctionB(thing){
 ## logic here
}
```

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
            result = SubfunctionA()
        }else {
                result = SubfunctionB()

    return result;
}
```

Your handler

```
Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```

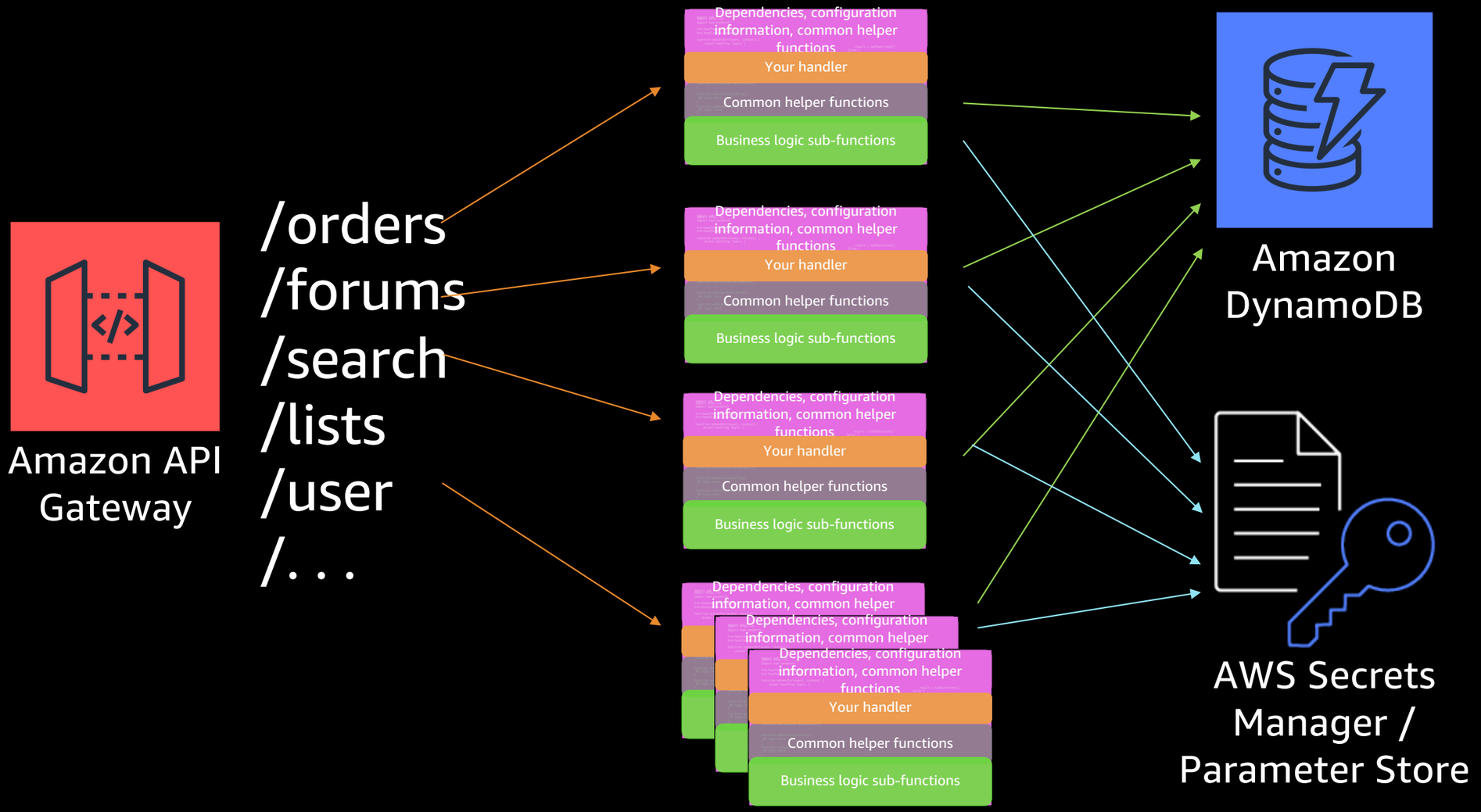
Common helper functions

```
Function subFunctionA(thing){
 ## logic here
}

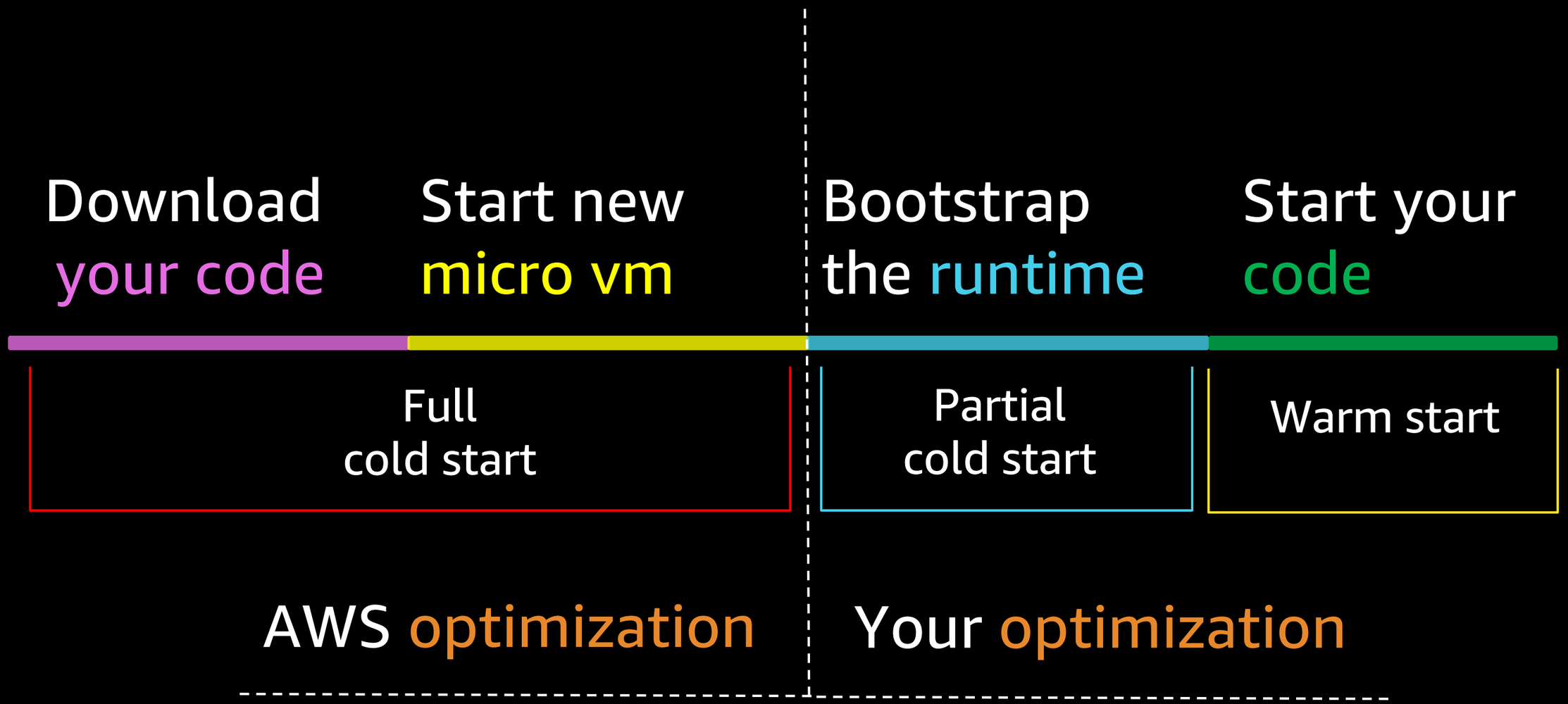
Function subFunctionB(thing){
 ## logic here
}
```

Business logic subfunctions

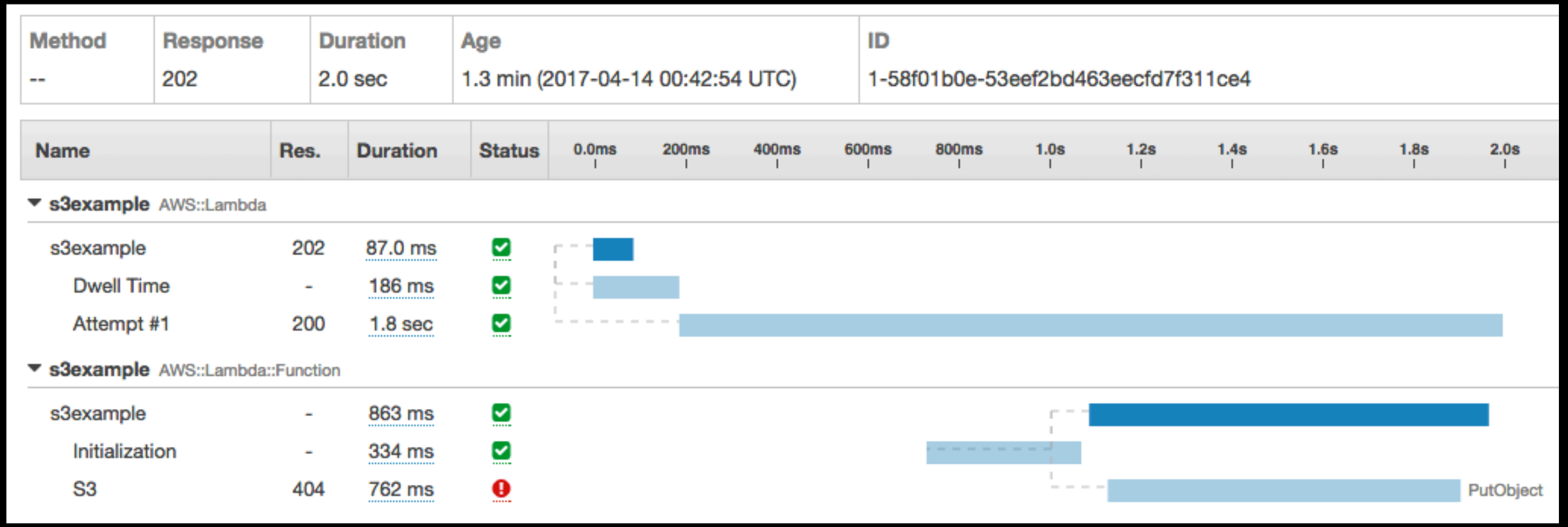
Anatomy of a serverless application



The invocation lifecycle



AWS X-Ray trace example



Optimization best practices



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Optimization best practices (cold start)

Avoid “monolithic” functions

- Reduce deployment package size
- Micro/nano services

Optimize dependencies

- Up to 120 ms faster with Node.js SDK

Reuse connections

- Keep-Alive in AWS SDKs

Minify / uglify production code

- Browserify / Webpack

Lazy initialization of shared libs

- Helps if there are multiple functions per file

Use provisioned concurrency

- On individual functions, no code changes

Lazy initialization example (Python and boto3)

```
import boto3

S3_client = None
ddb_client = None

def get_objects(event, context):
    if not s3_client:
        s3_client = boto3.client("s3")
    # business logic

def get_items(event, context):
    if not ddb_client:
        ddb_client = boto3.client(”dynamodb")
    # business logic
```

Optimized dependency usage (Node.js)

```
// const AWS = require('aws-sdk’)
const DynamoDB = require('aws-sdk/clients/dynamodb’) // 125ms faster

// const AWSXRay = require('aws-xray-sdk’)
const AWSXRay = require('aws-xray-sdk-core’)  // 5ms faster

// const AWS = AWSXRay.captureAWS(require('aws-sdk’))
const dynamodb = new DynamoDB.DocumentClient()
AWSXRay.captureAWSClient(dynamodb.service)  // 140ms faster
```

theburningmonk.com/2019/03/just-how-expensive-is-the-full-aws-sdk/



@theburningmonk

Provisioned concurrency for AWS Lambda

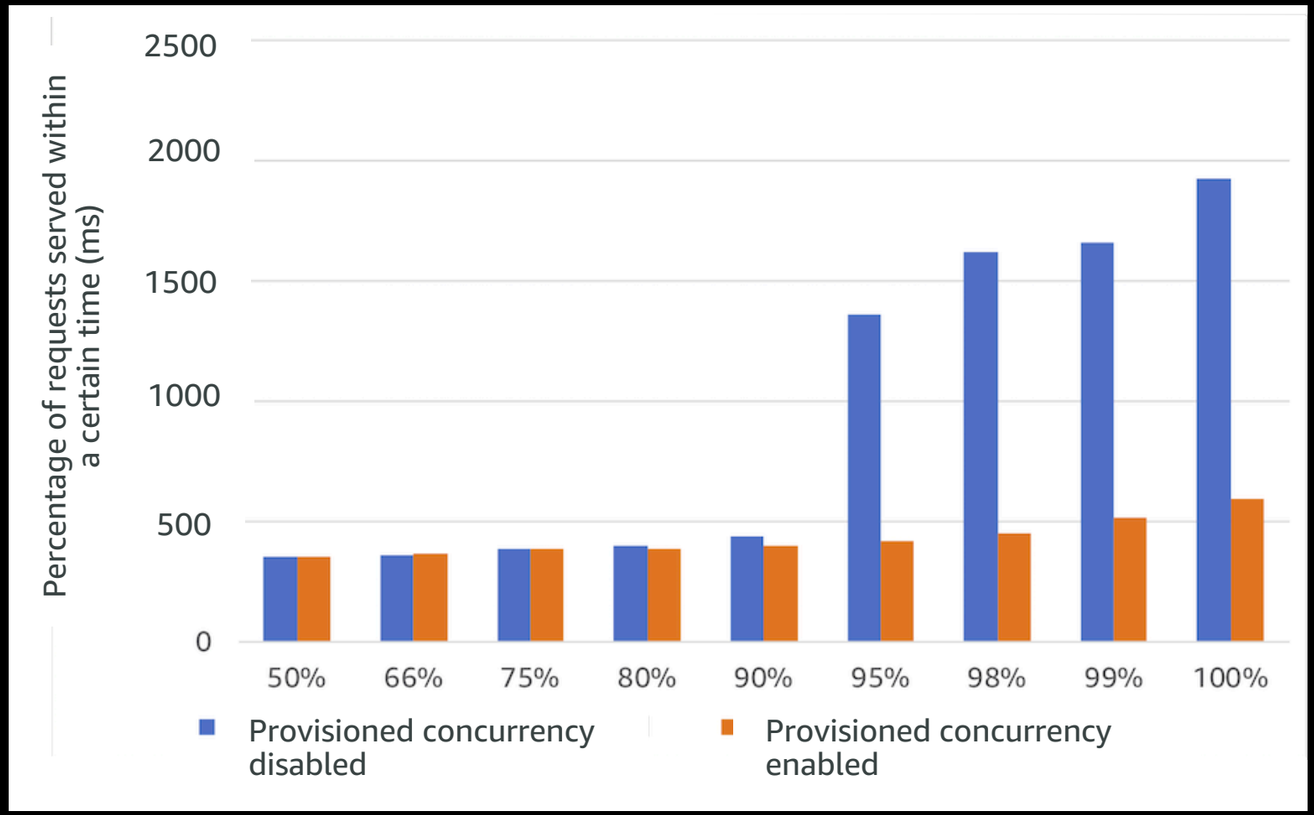
Managed solution to avoid “cold starts”

You set provisioned concurrency for individual functions

No code changes required

Integrated with AWS Auto Scaling

Provisioning ramp-up (500 per min)



Optimization best practices: Performance / cost

Externalize orchestration

Avoid idle; delegate to AWS Step Functions

Transform, not transport

Minimize data transfer (S3 Select, advanced filtering, and so on)

Discard uninteresting events as soon as possible

Trigger config (S3 prefix, SNS filter)

Fine-tune resources allocation

Don’t guesstimate function memory

Lambda destinations

Simplified chaining (async) and DLQ

Keep in mind retry policies

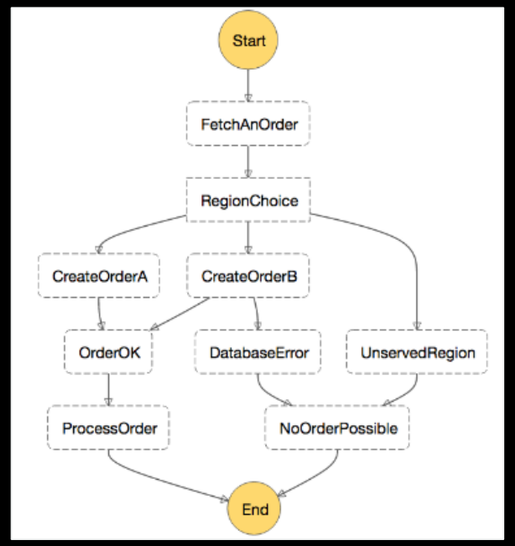
Powerful (and configurable), but not free

Externalize orchestration: Step Functions

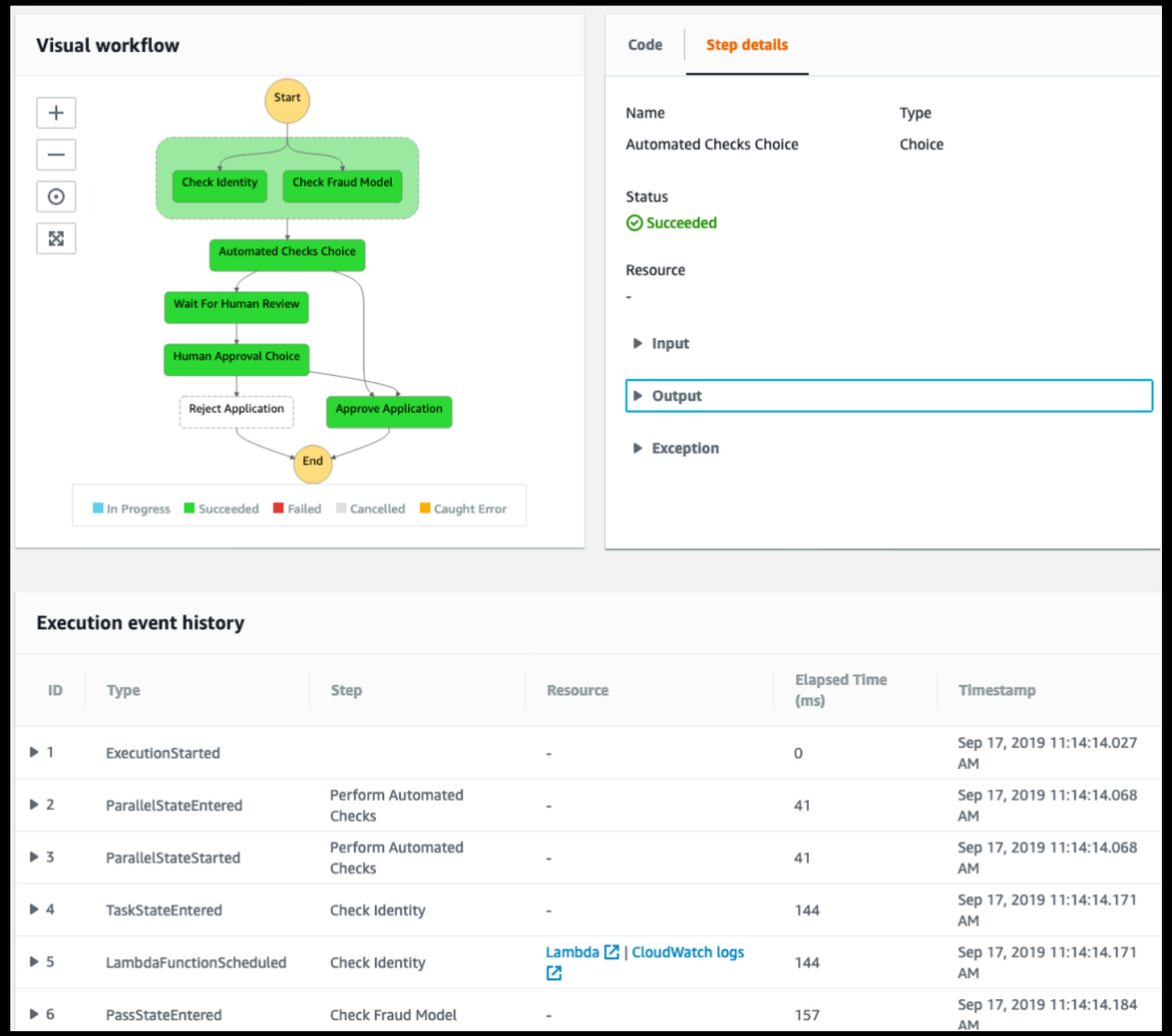
Define in JSON

```
{
    "Comment": "Manage opening an account",
    "StartAt": "Perform Automated Checks",
    "States": {
        "Perform Automated Checks": {
            "Type": "Parallel",
            "Branches": [{
                    "StartAt": "Check Identity",
                    "States": {
                        "Check Identity": {
                            "Type": "Task",
                            "Parameters": {
```

Visualize in the console



Monitor executions



Transform, not transport: Example

Before (200s, $0.112)

```
# Download and process all keys
for key in src_keys:
  response = s3_client.get_object(…)
  contents = response['Body'].read()
  for line in contents.split('\n')[:-1]:
      line_count +=1
      try:
          data = line.split(',')
          srcIp = data[0][:8]
          …
```

After (95s, $0.028)

```
# Select IP Address and Keys
for key in src_keys:
  response = s3_client.select_object_content(
    expression="SELECT SUBSTR(obj._1, 1, 8),
                obj._2 FROM s3object as obj")
  contents = response['Body'].read()
  for line in contents:
      line_count +=1
      try:
          …
```

https://github.com/awslabs/lambda-refarch-mapreduce

Optimization best practices: Performance / cost

Externalize orchestration

Avoid idle; delegate to AWS Step Functions

Transform, not transport

Minimize data transfer (S3 Select, advanced filtering, and so on)

Discard uninteresting events as soon as possible

Trigger config (S3 prefix, SNS filter)

Fine-tune resources allocation

Don’t guesstimate function memory

Lambda destinations

Simplified chaining (async) and DLQ

Keep in mind retry policies

Powerful (and configurable), but not free

Optimization best practices: Performance / cost

Externalize orchestration

Avoid idle; delegate to AWS Step Functions

Fine-tune resources allocation

Don’t guesstimate function memory

Transform, not transport

Minimize data transfer (S3 Select, advanced filtering, and so on)

Lambda destinations

Simplified chaining (async) and DLQ

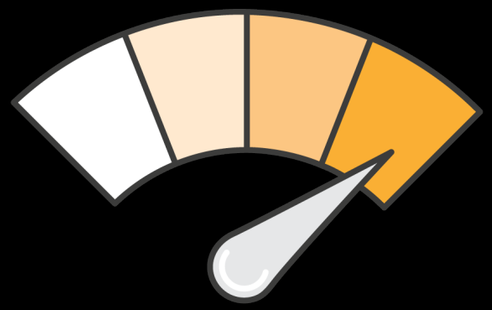
Discard uninteresting events as soon as possible

Trigger config (S3 prefix, SNS filter)

Keep in mind retry policies

Powerful (and configurable), but not free

Resources allocation



Memory 👉 Power

CPU-bound example

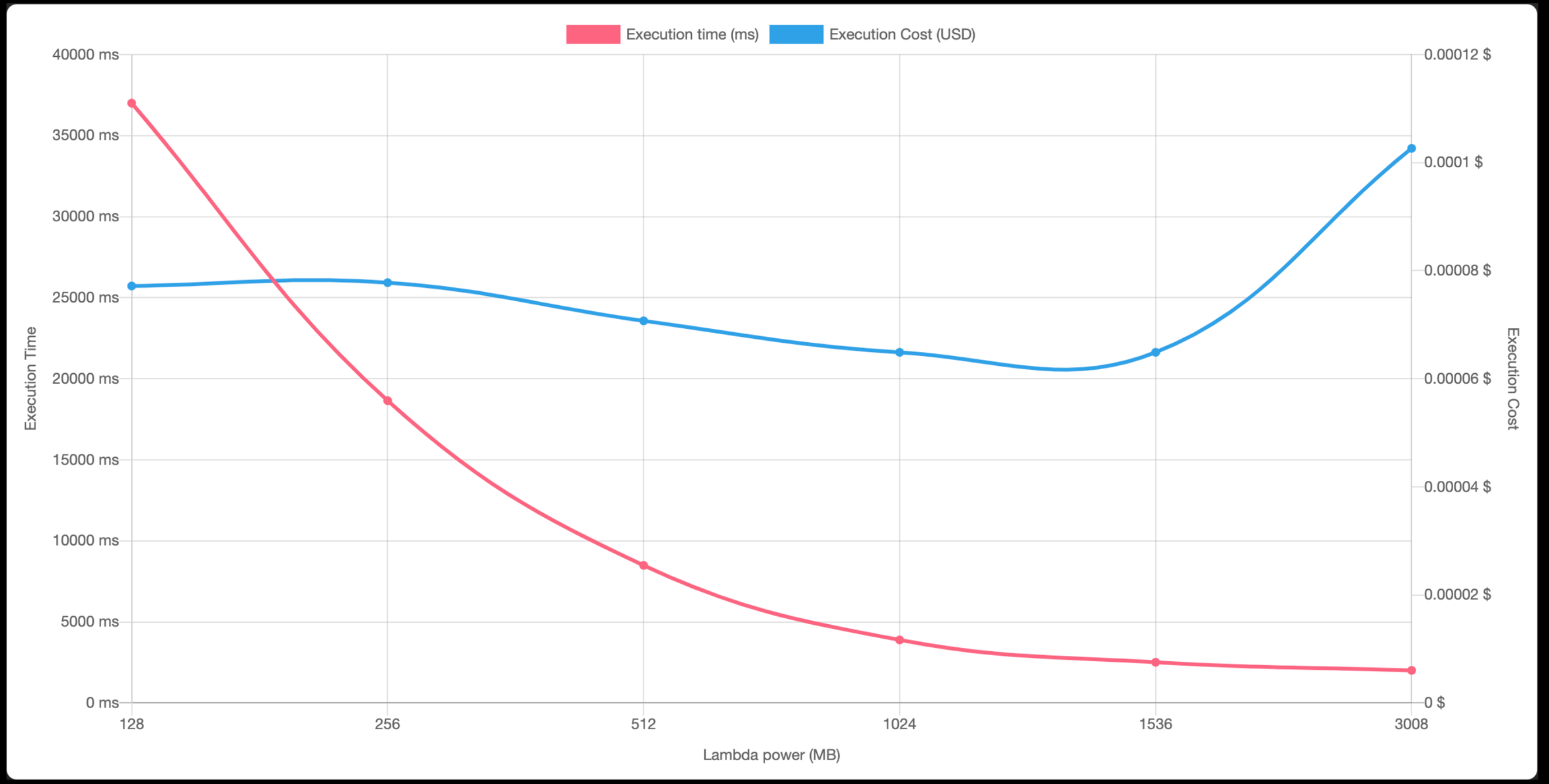
"Compute **1000 times** all prime numbers **≤1M**"

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

CPU-bound example

"Compute **1000 times** all prime numbers **≤1M**"

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

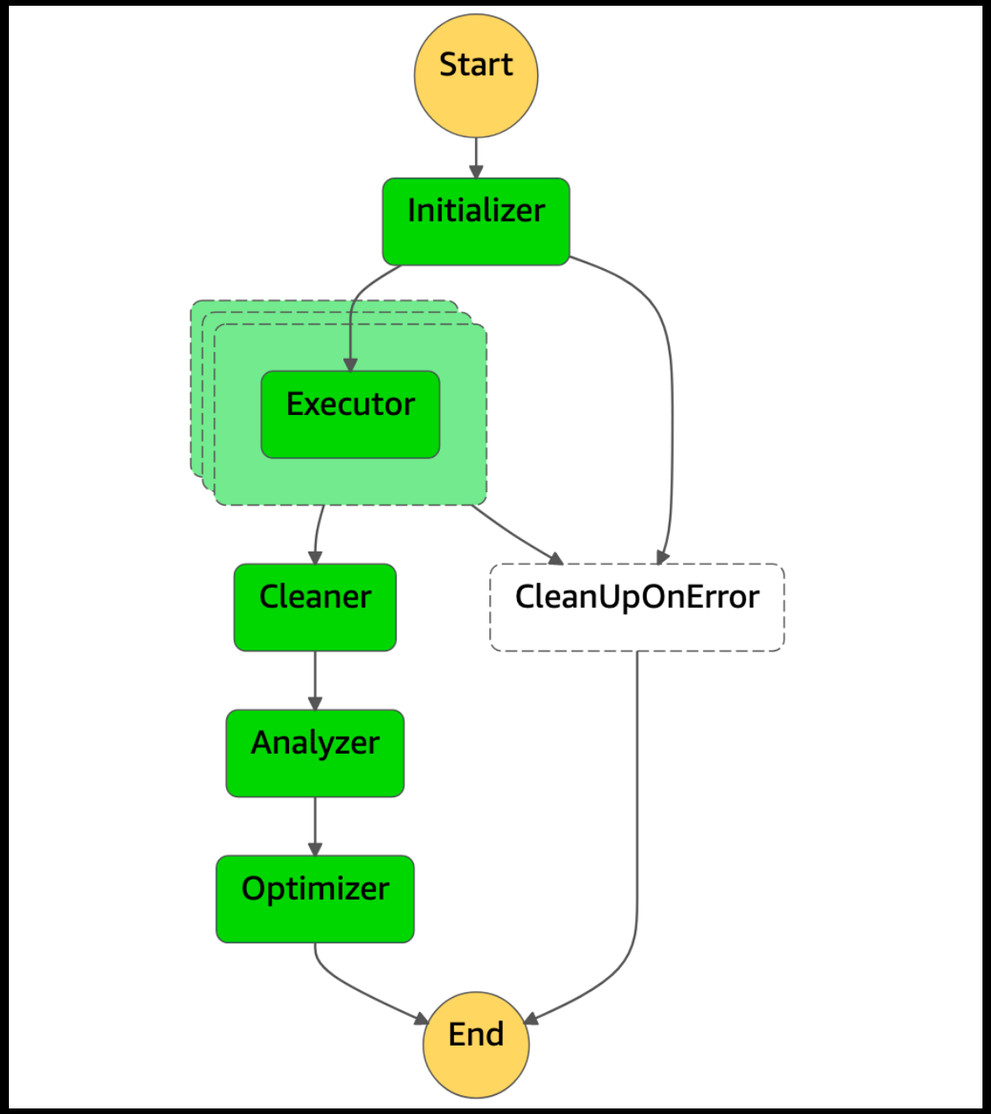

Execution time (ms)
Execution Cost (USD)
40000 ms
35000 ms
30000 ms
25000 ms
20000 ms
15000 ms
10000 ms
5000 ms
0 ms
Execution Time
0.00012 $
0.0001 $
0.00008 $
0.00006 $
0.00004 $
0.00002 $
0 $
Execution Cost
128
256
512
1024
1536
3008
Lambda power (MB)

Idea: Data-driven approach



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Meet AWS Lambda Power Tuning



Data-driven cost and performance optimization for AWS Lambda

Available as an SAR app

Easy to integrate with CI / CD

github.com/alexcasalboni/aws-lambda-power-tuning

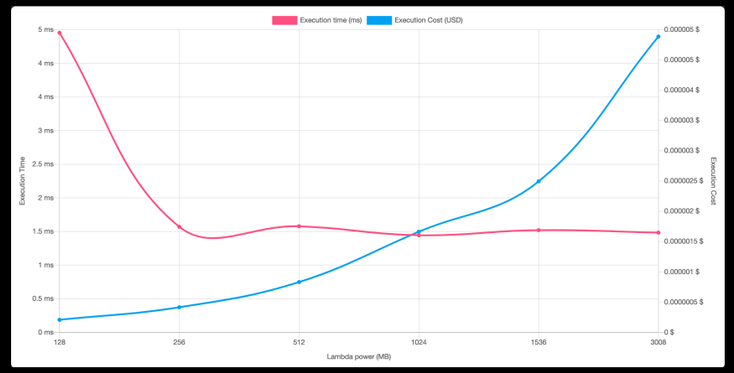
AWS Lambda Power Tuning: Input / output

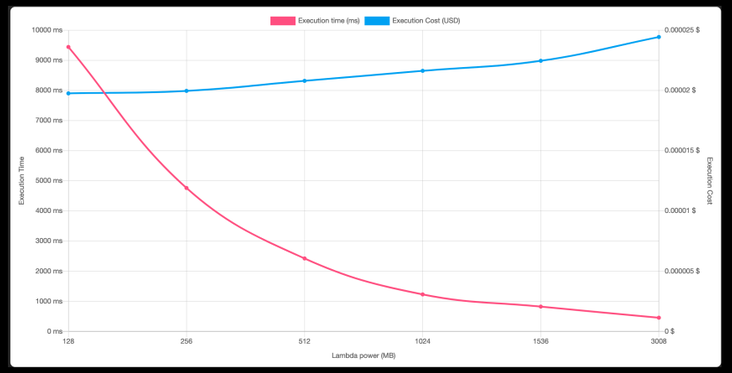
```
{
    "lambdaARN": "your-lambda-function-arn",
    "powerValues": [128, 256, 512, 1024, 2048, 3008],
    "num": 100,
    "payload": {"data": "abc"}
}
```

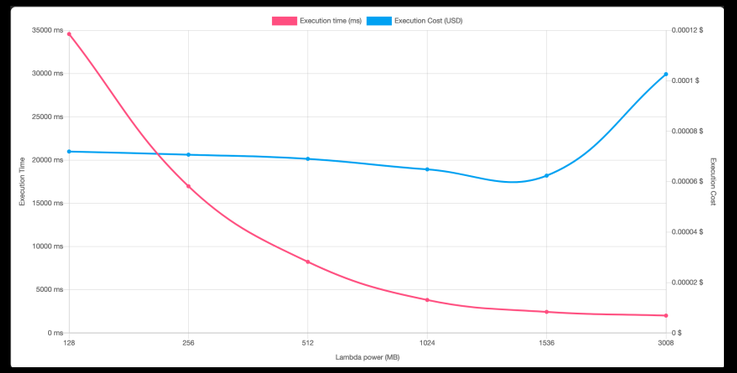
```
{
  "results": {
    "power": "128",
    "cost": 2.08e-7,
    "duration": 2.906,
    "stateMachine": {
      "executionCost": 0.00045,
      "lambdaCost": 0.0005252,
      "visualization": "https://lambda-power-tuning.show/ ... "
    }
  }
}
```

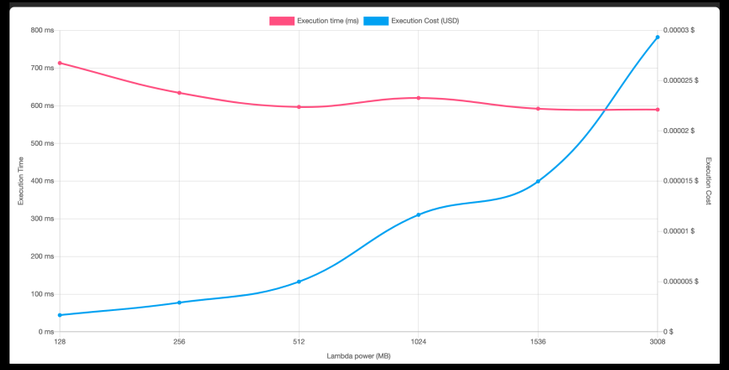
github.com/alexcasalboni/aws-lambda-power-tuning

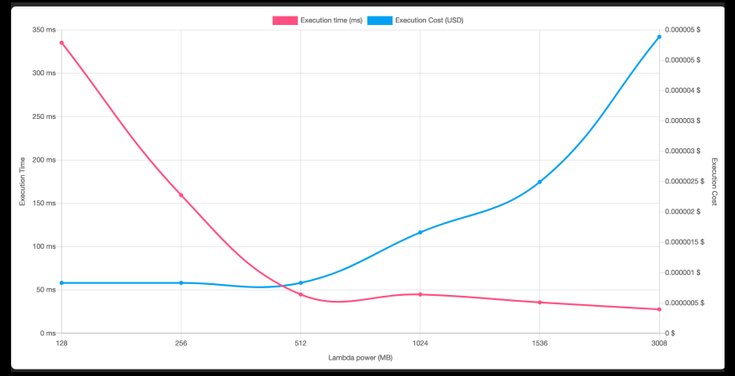
AWS Lambda Power Tuning: Visualization

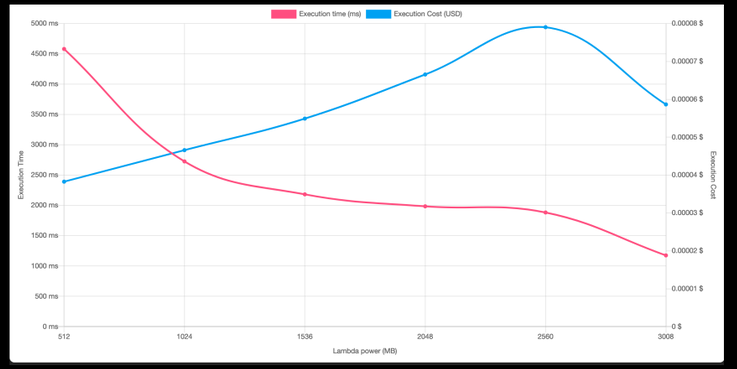












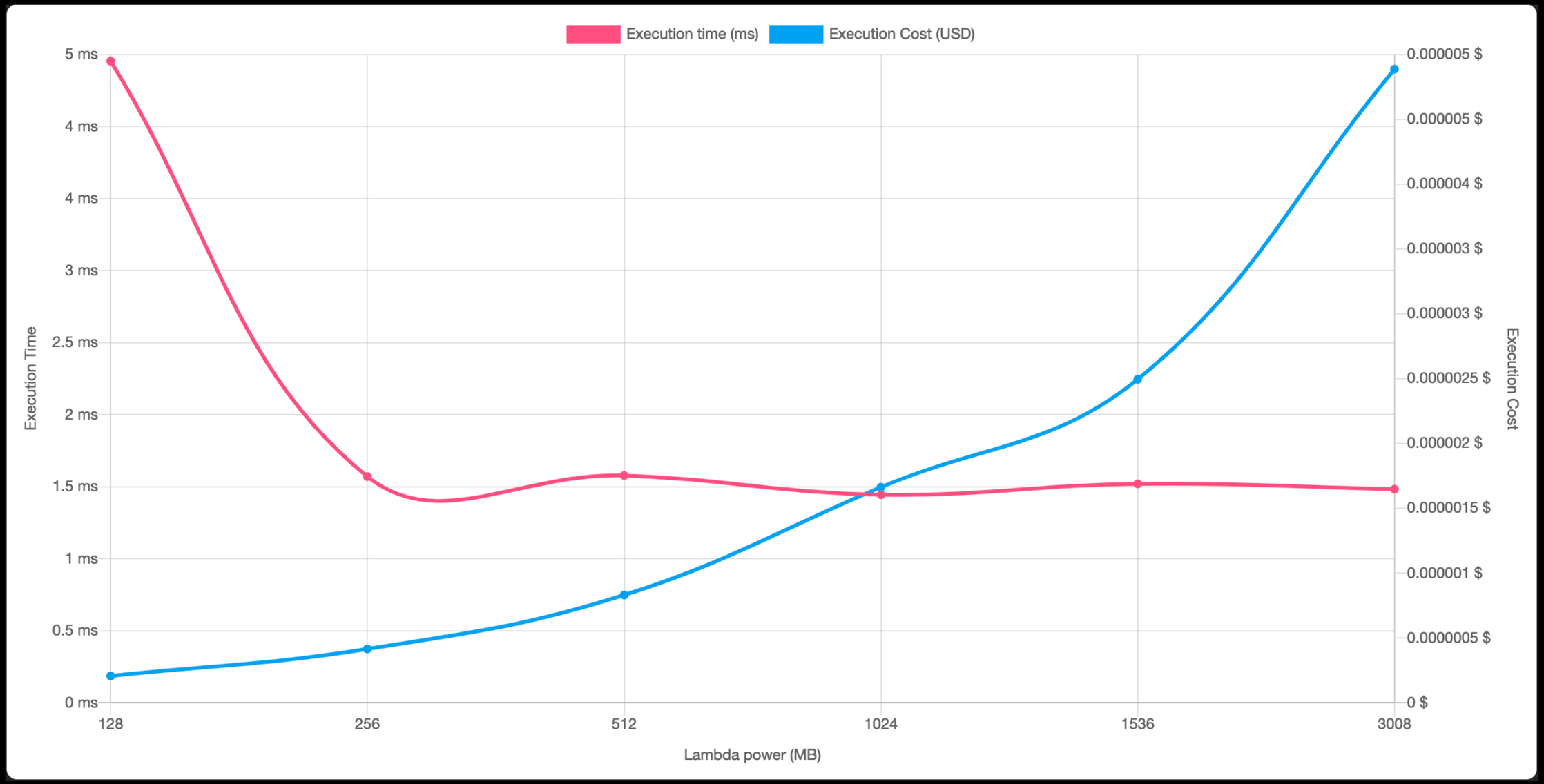
github.com/alexcasalboni/aws-lambda-power-tuning

Real-world examples and patterns

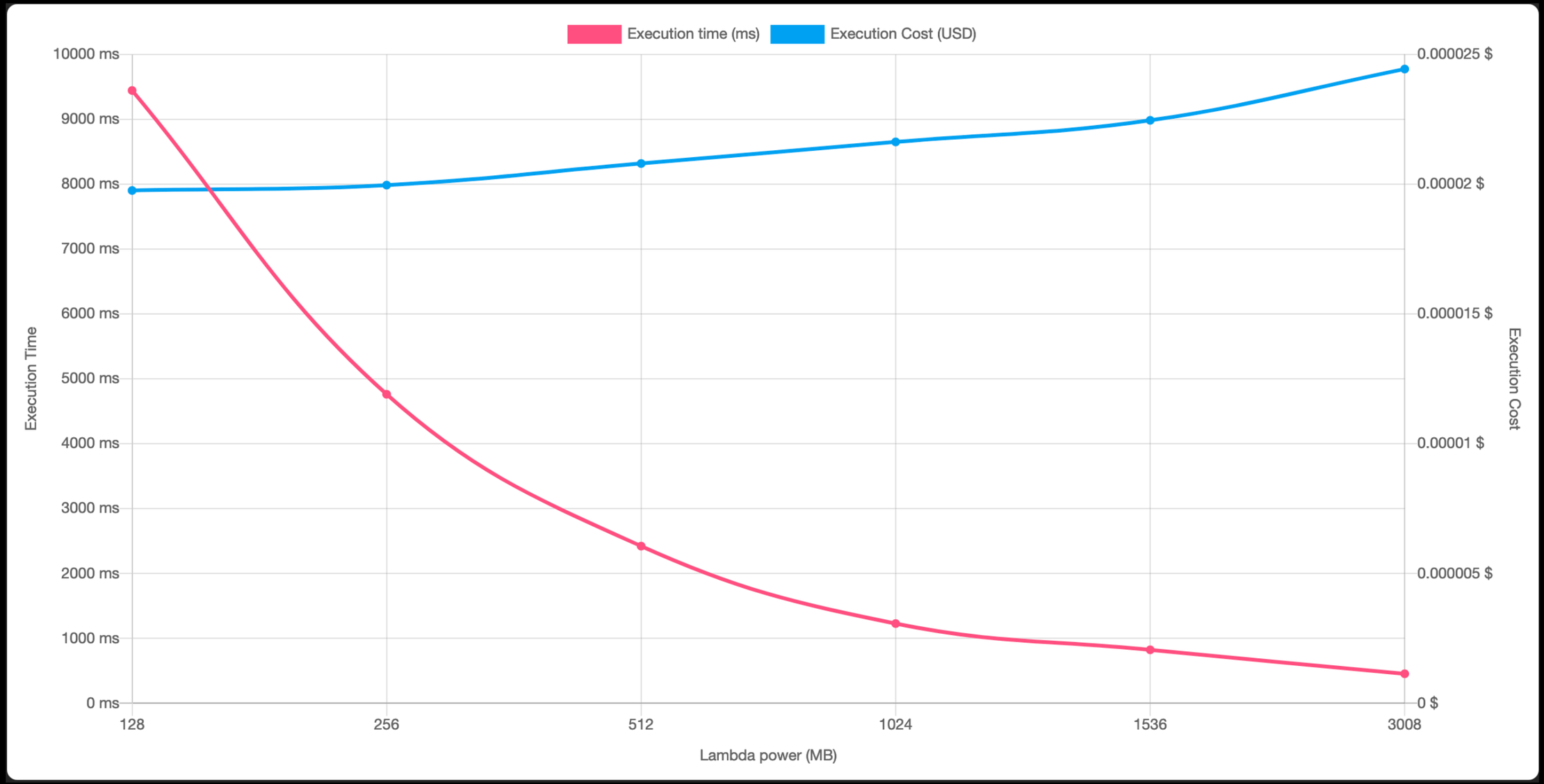


© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

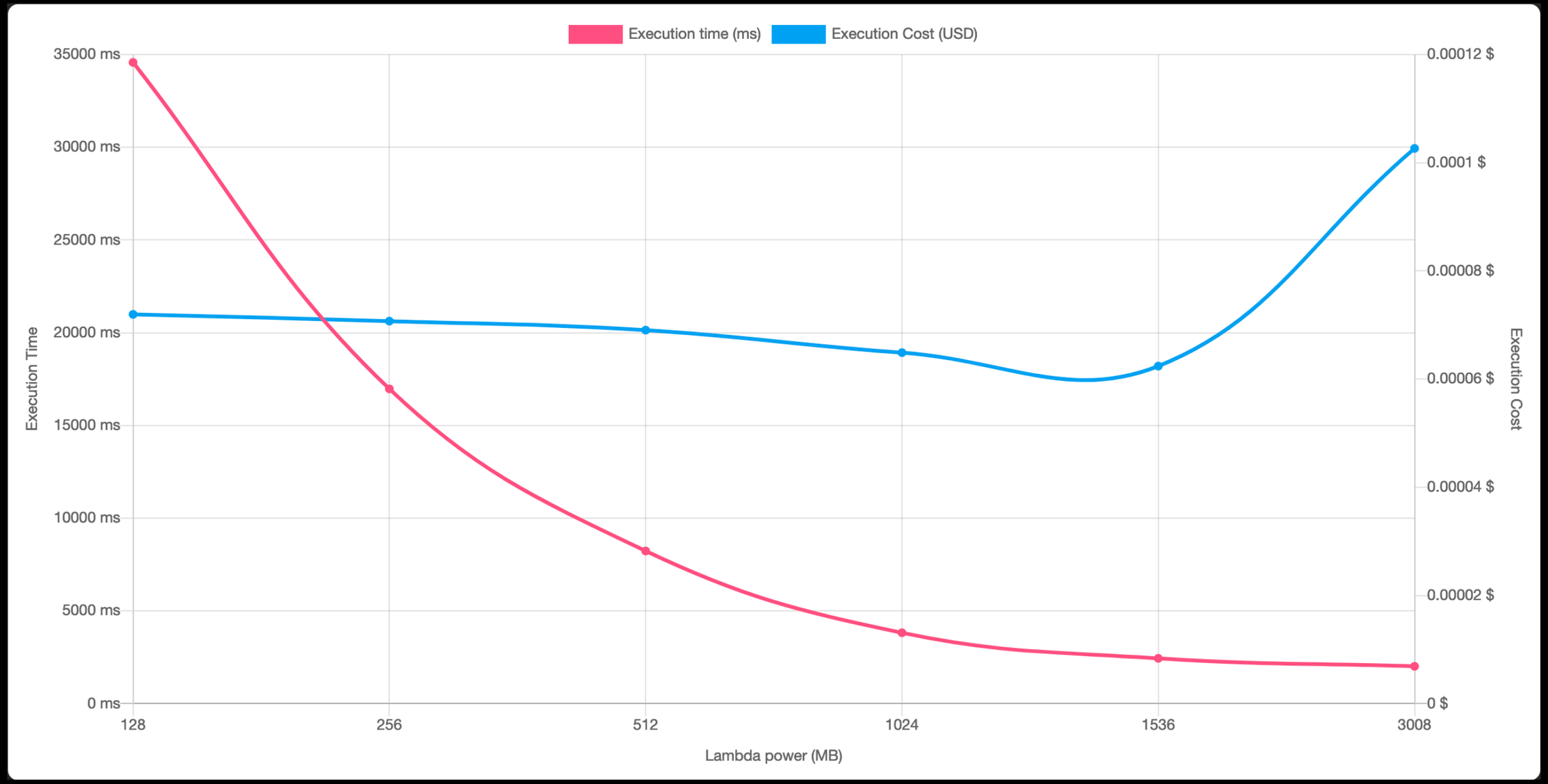
No-op: Trivial data manipulation <100 ms



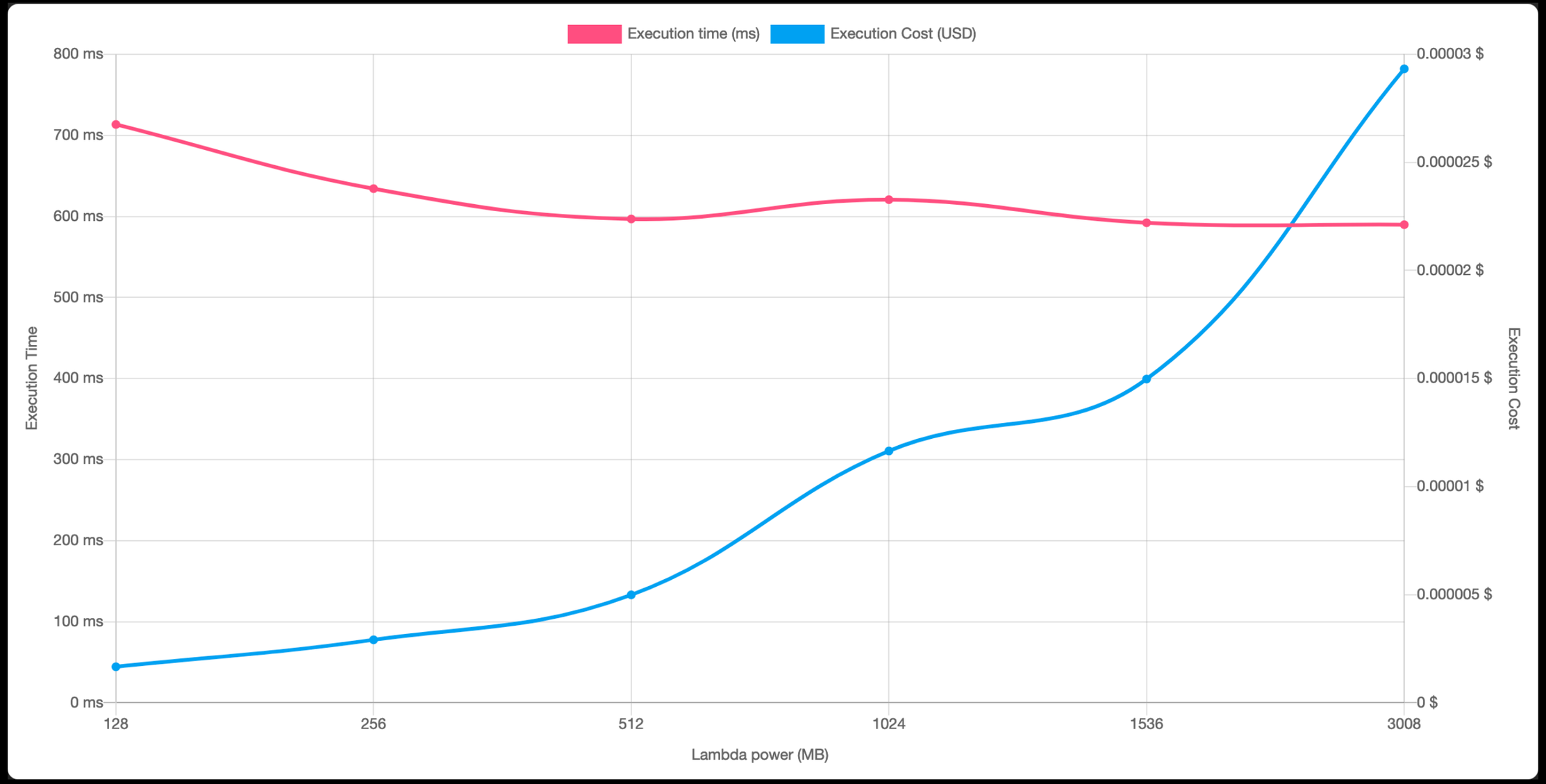
CPU-bound: Numpy – inverting a large matrix



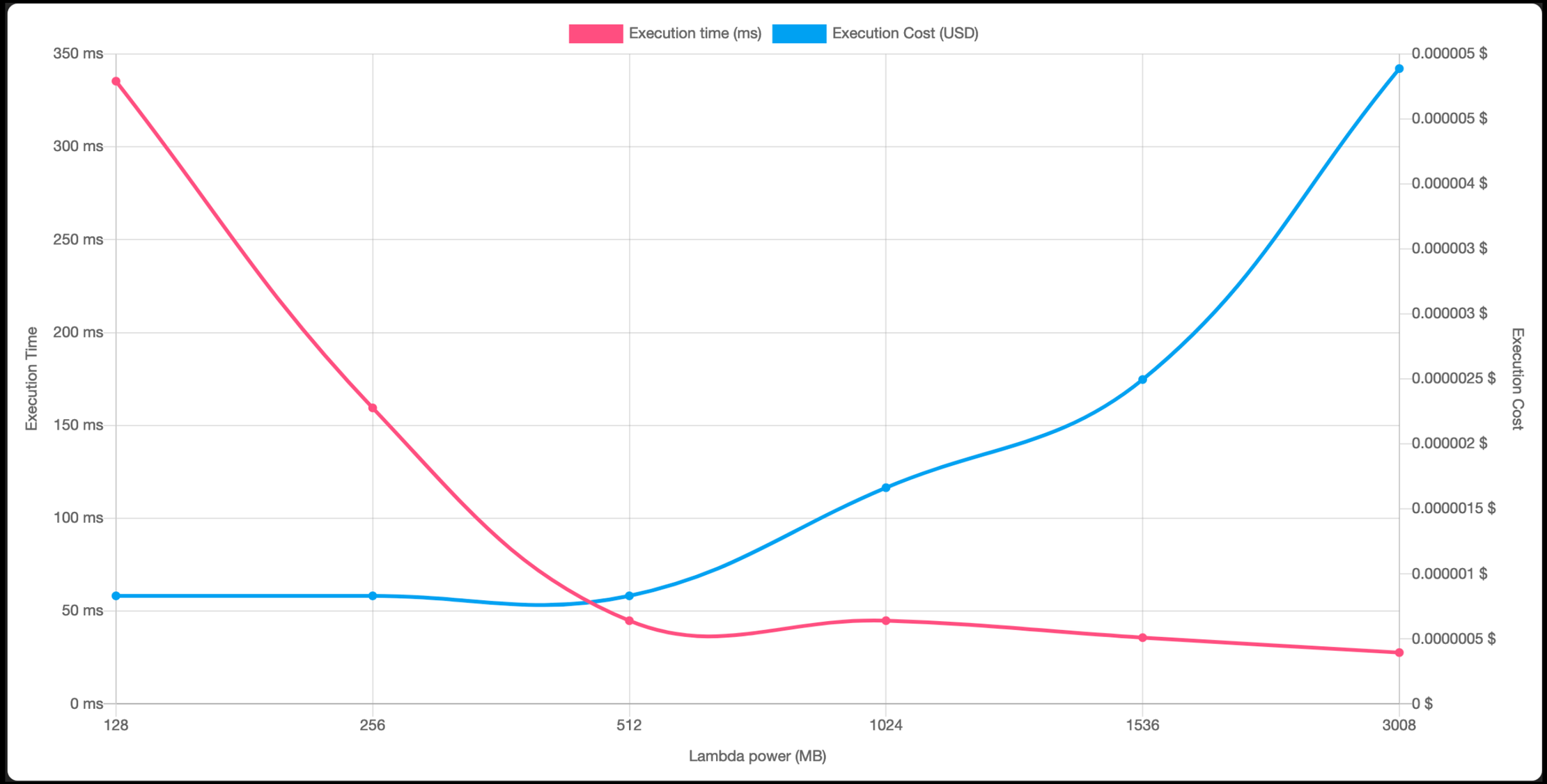
CPU-bound: Prime numbers



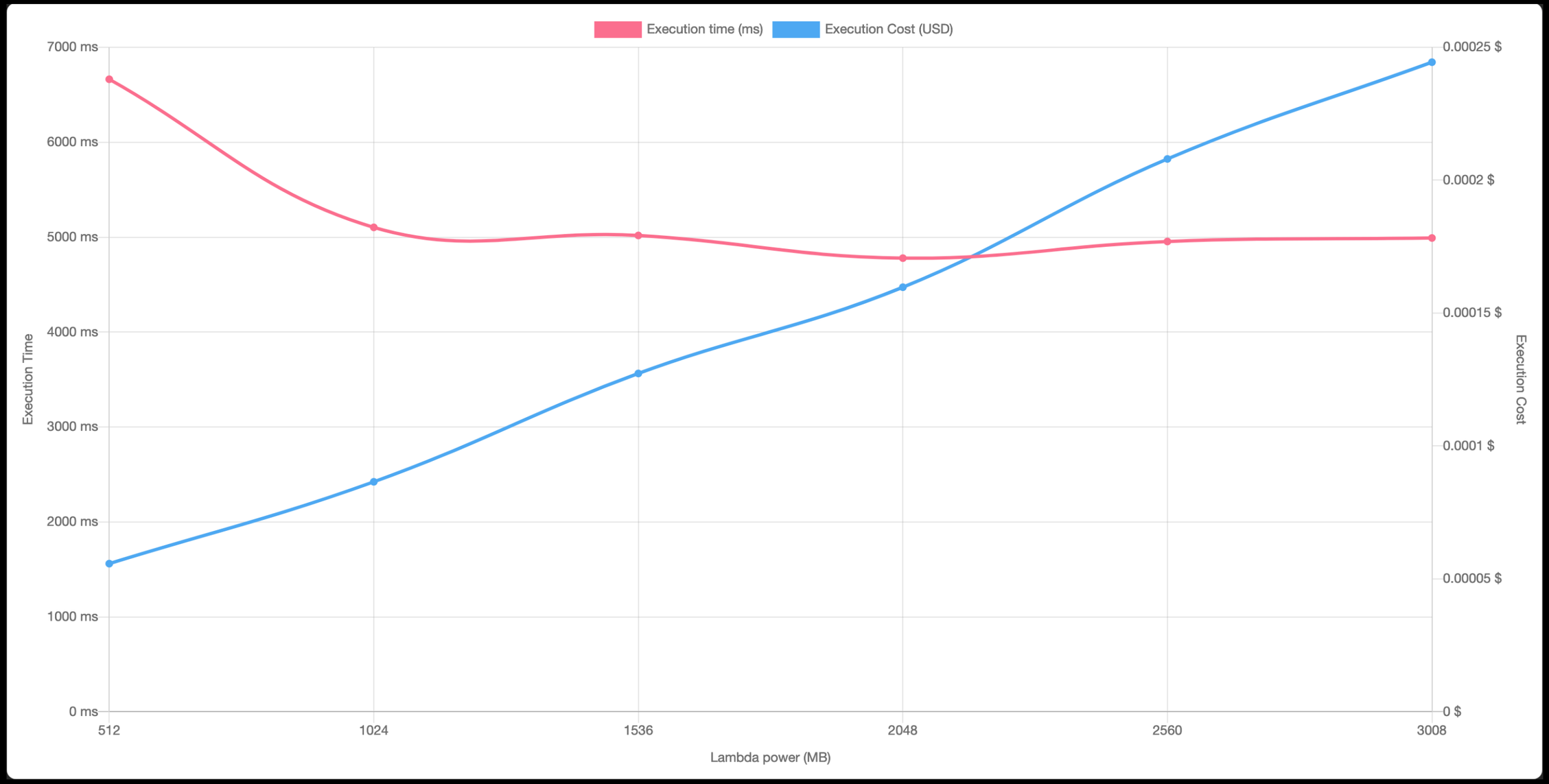
Network-bound: Third-party API call



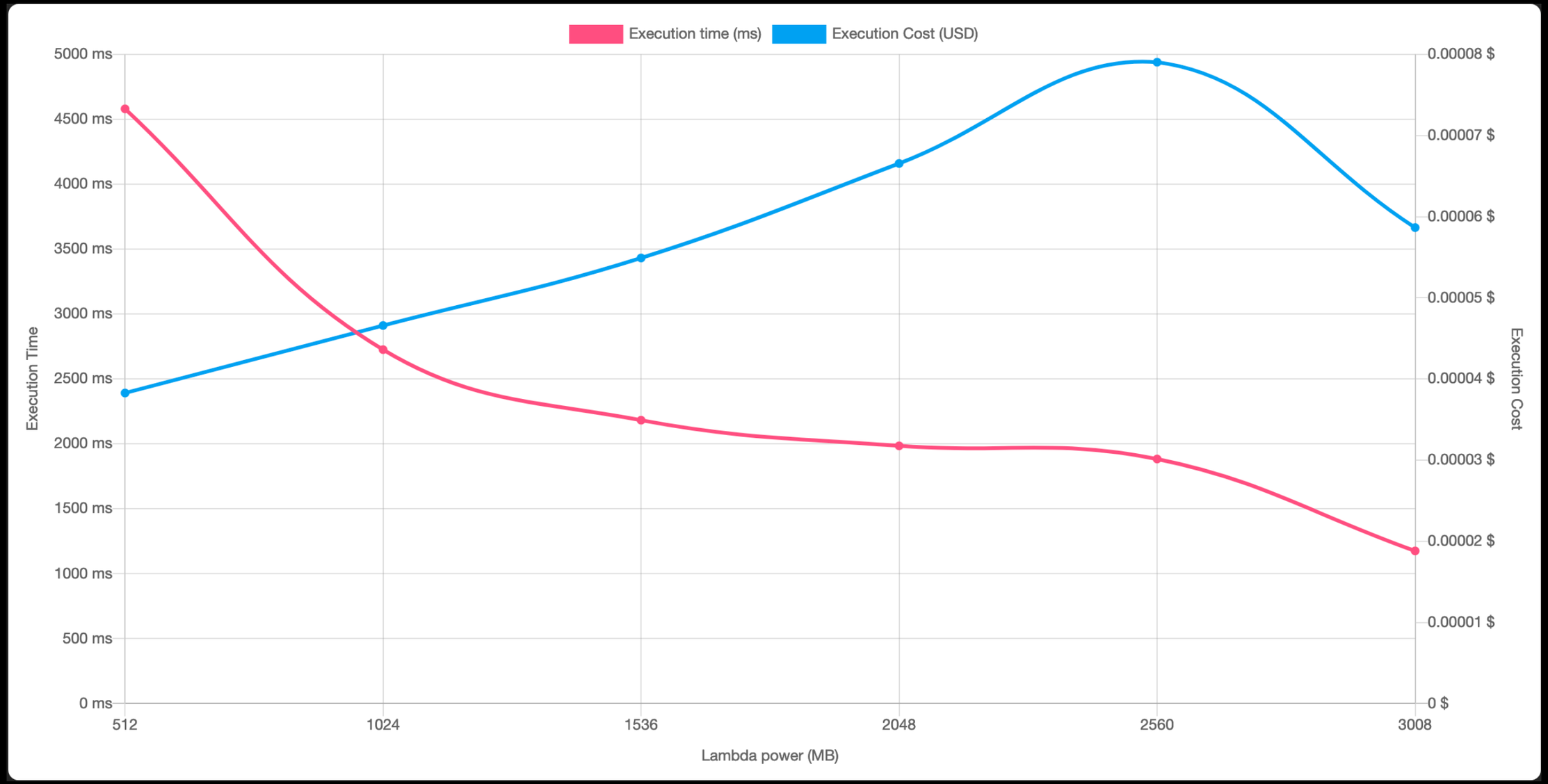
Network-bound: 3x DDB queries



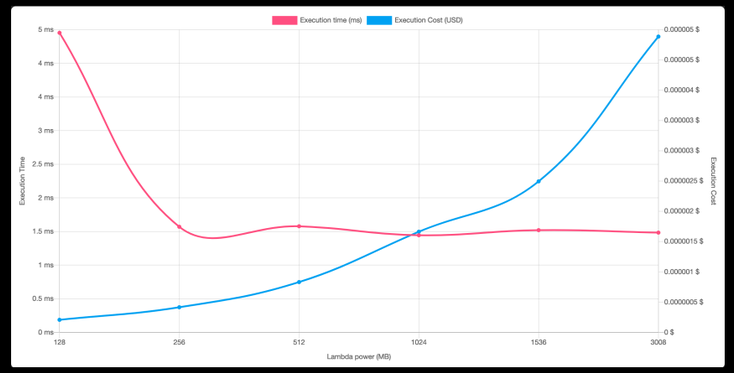
Network-bound: S3 download – 150 MB

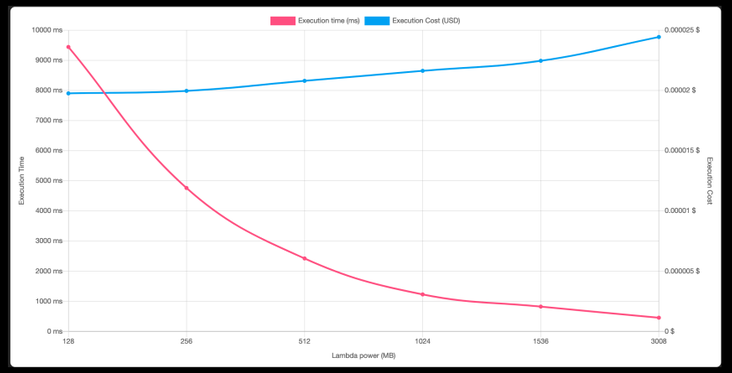


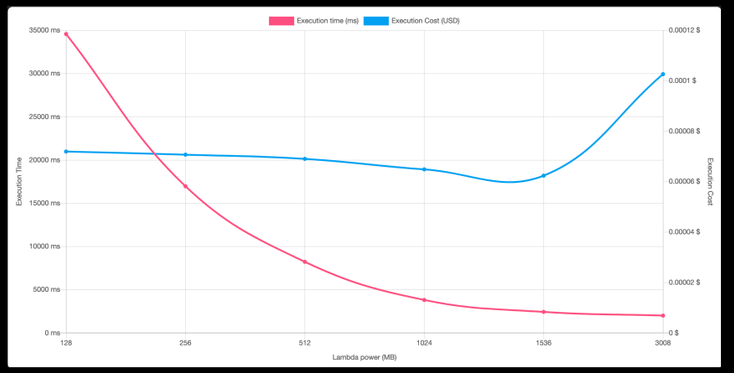
Network-bound: S3 download multithread

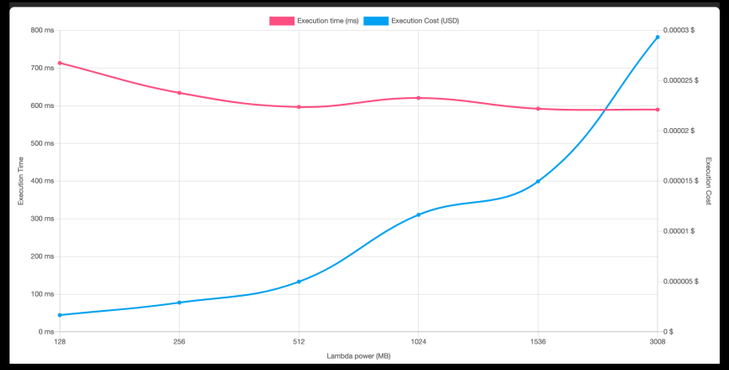


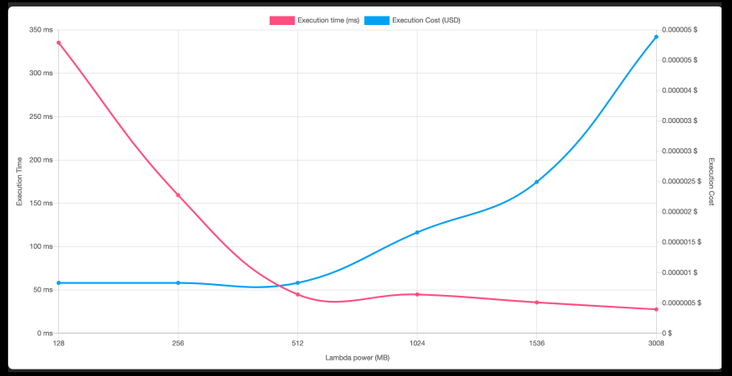
Cost / performance patterns

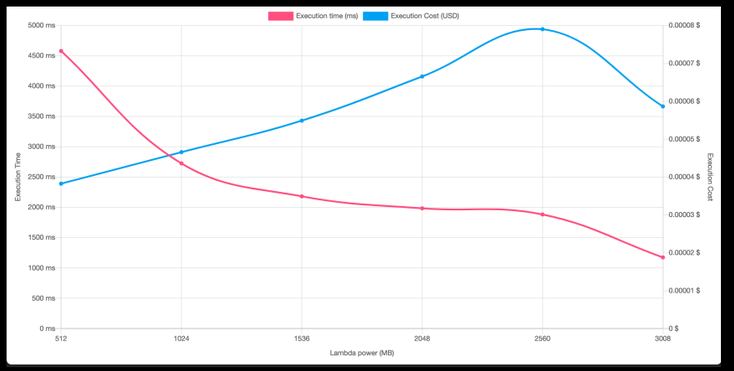












github.com/alexcasalboni/aws-lambda-power-tuning

Takeaways



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Takeaways

Optimization best practices (what are you trying to optimize?)

Avoid cold starts with provisioned concurrency

Memory 👉 power

Think in terms of workload categories and cost / performance patterns

Visualize optimal trade-offs with /alexcasalboni/aws-lambda-power-tuning

Thank you!

Alex Casalboni

@alex_casalboni

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.