Serverless on AWS

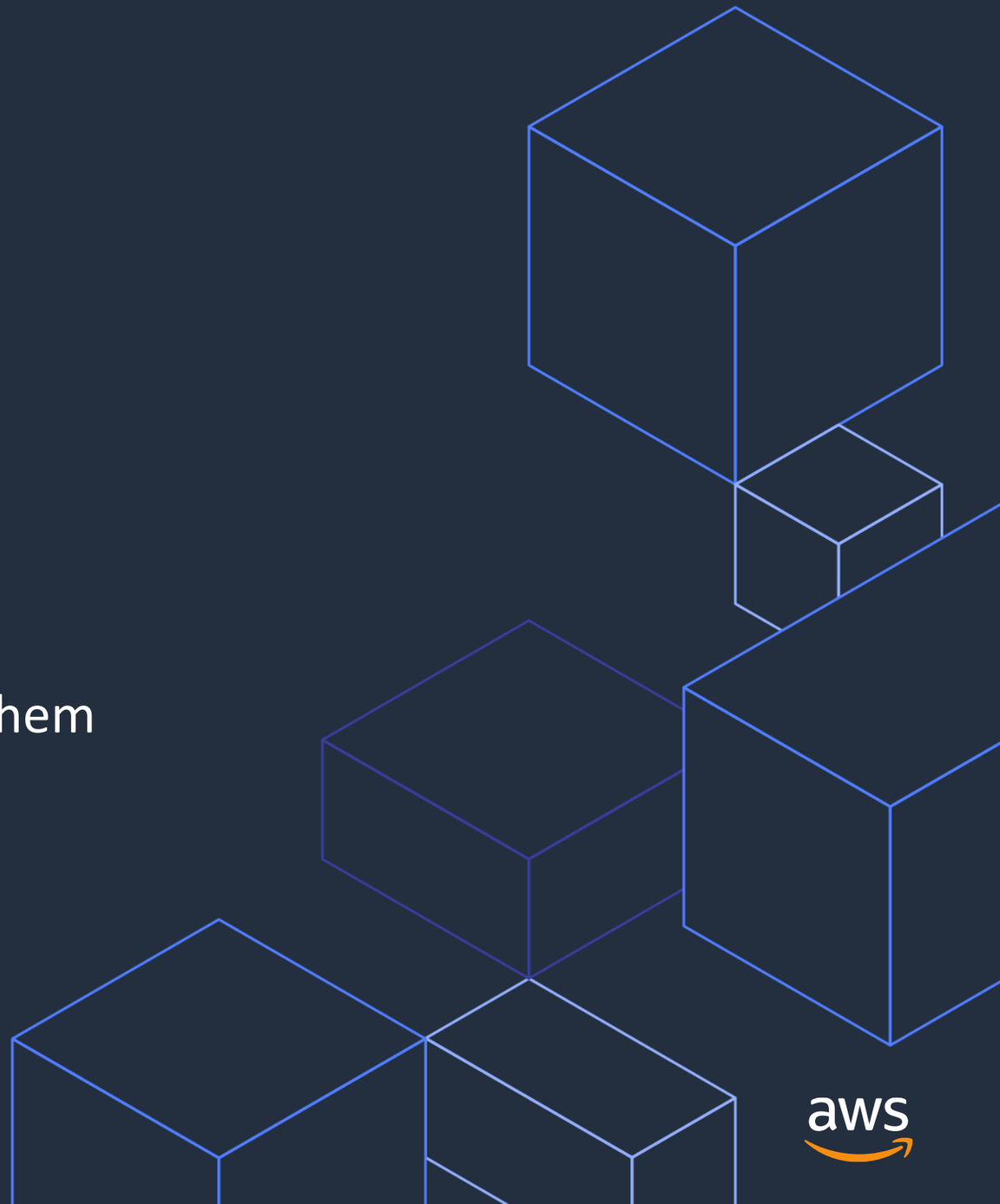
Immersion Day

Vamsi Vikash Ankam

2021

Building your second Lambda function

Or the mistakes we've all made … and how to fix them

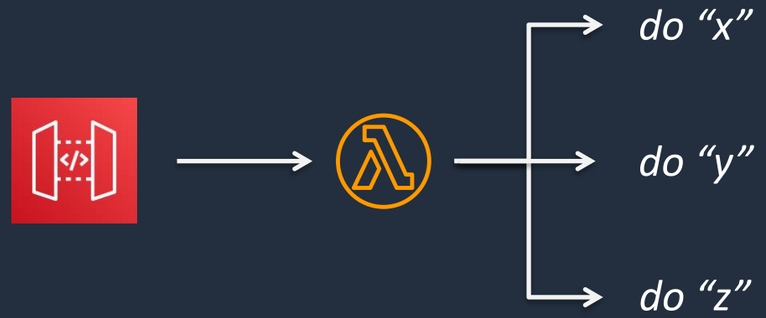


The “Lambda-lith” architecture

- *Should I write a single function with internal branching?*

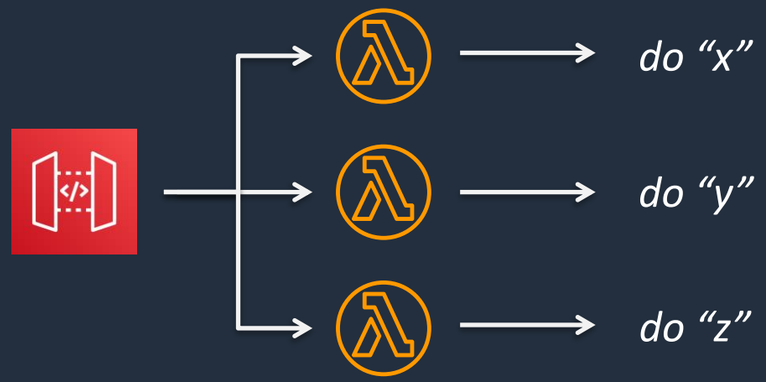
- Benefits:
  - Fewer functions to manage
  - Simpler migration path

- Challenges:
  - Large package size
  - Hard to enforce least privilege
  - Scaling team
  - Maintenance



Solution: The “Lambda-lith” architecture

- For net new, recommend single-purpose functions (or microservices)
- Benefits:
  - Easier to debug, trace errors
  - Enforce least privilege, one role per function
  - Simpler to test
  - Code reuse
  - Less surface area
- Tips:
  - Use naming conventions and tags to enhance discoverability
  - Don’t go *nano*



aws

Integrating with my relational database

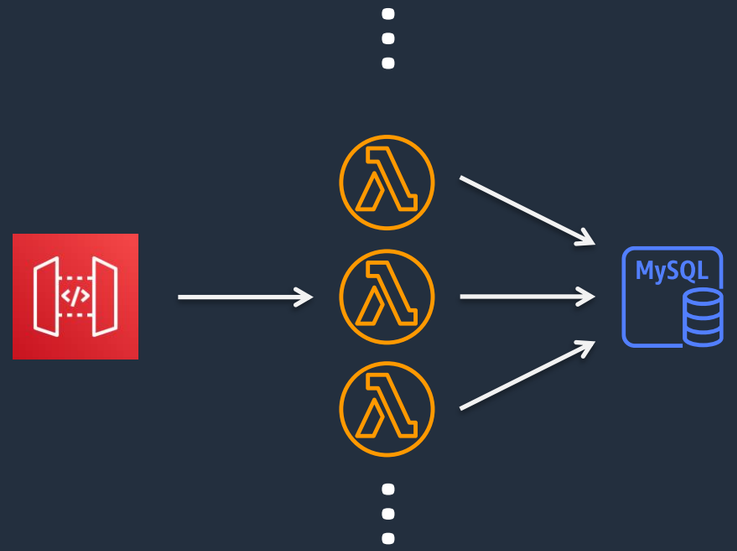
… or other constrained downstream resources

- *My experience / existing app is built with a relational database, will this work with Lambda?*

- Representative of two common issues:
  - Drowning downstream resources
  - Reusing expensive operations

- Lambda can scale up quickly, putting pressure on downstream resources

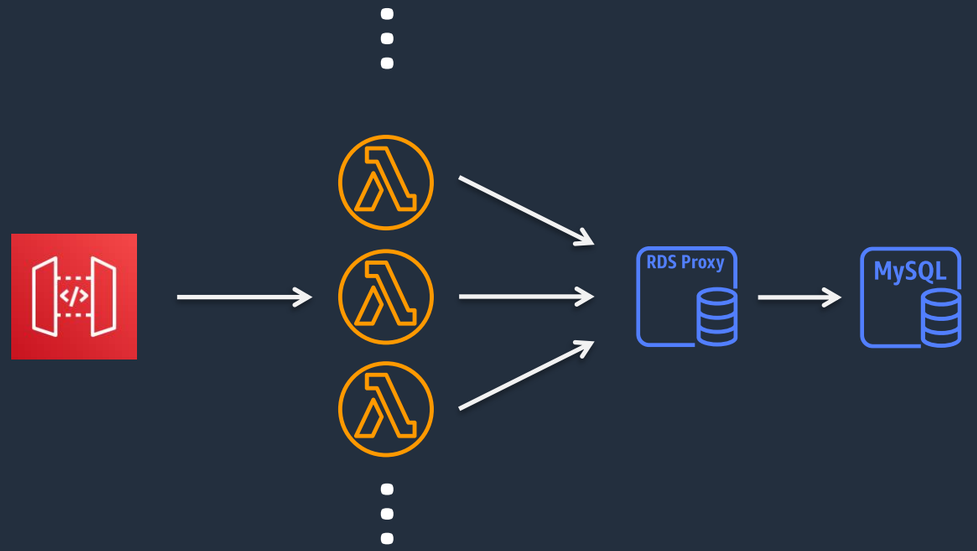
- Expensive operations, such as creating a database connection, take time
  - Time is proportional to cost
  - Impacts function cold start



aws

Solution: Pressure on downstream resources

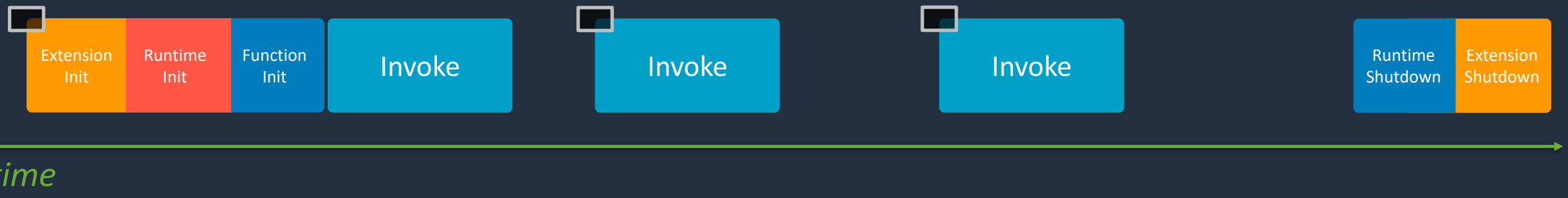
- Consider resource type: (1) relational database or (2) other
- Relational database options:
  - For MySQL, PostgreSQL, consider Amazon RDS Proxy for shared connection pool
  - Aurora Serverless supports Data API
- Other resource options:
  - Enable reserved concurrency
  - Insert a queue between function and resource



Solution: Managing expensive operations

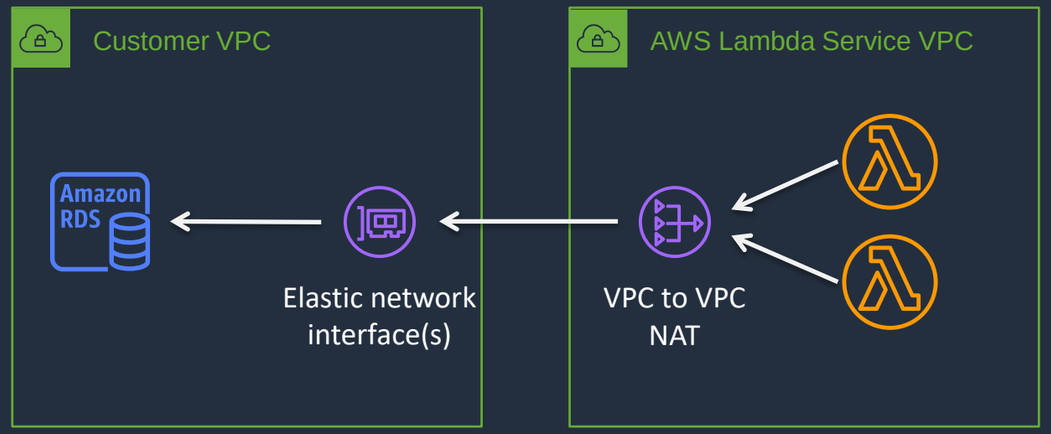
- Initialize clients and database connections outside the function handler
  - Consider caching static assets in `/tmp`
  - Avoid for user or other sensitive data
- Subsequent invocations can reuse these resources

```
import boto3

client = None

def handler(event):
  global client
  if not client:
    client = boto3.client("dynamodb")

  # business logic
```



To VPC or not VPC?

- *Should my Lambda function be VPC-enabled?*
  - Other resources are in a VPC (e.g. EC2, RDS)
  - Security wants to implement network security tools
- Lambda functions always run in VPCs owned by the Lambda service team
  - When VPC enabled, configured with access to your VPC via an ENI
- Lambda functions are invoked via an action with access controlled by AWS IAM



Solution: To VPC or not VPC?

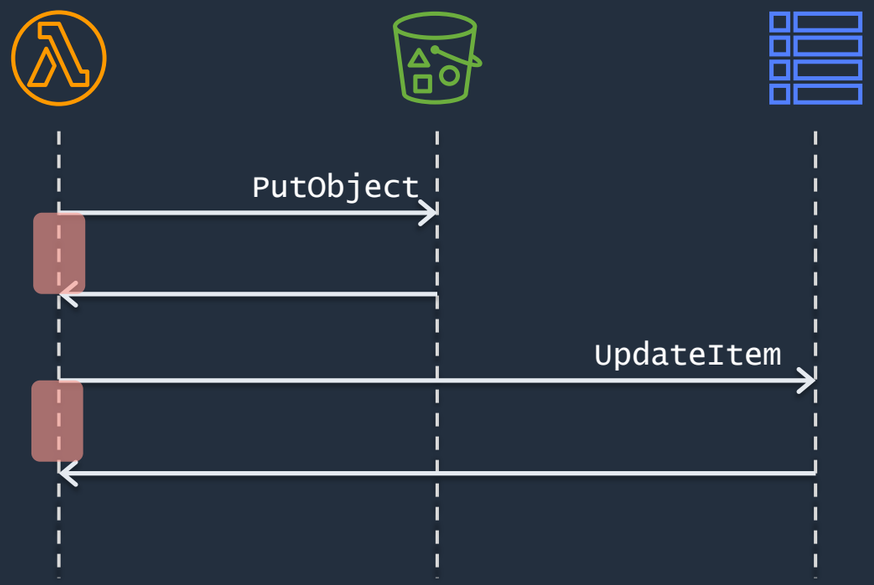
- Only if your function needs access to resources in the VPC
  - Includes access to on-prem via VPN or DX

- Tips for VPC functions:
  - Requires access to multiple AZs, select at least two subnets
  - Never target public subnets
  - Configure a NAT or NAT Gateway for internet access
  - Use VPC Endpoints for access to AWS services
  - ENIs can be exhausted, monitor usage

AWS services requiring VPC
Amazon ECS
Amazon EFS
Amazon ElastiCache
Amazon Elasticsearch Service
Amazon MSK
Amazon MQ
Amazon RDS
Amazon Redshift
AWS PrivateLink

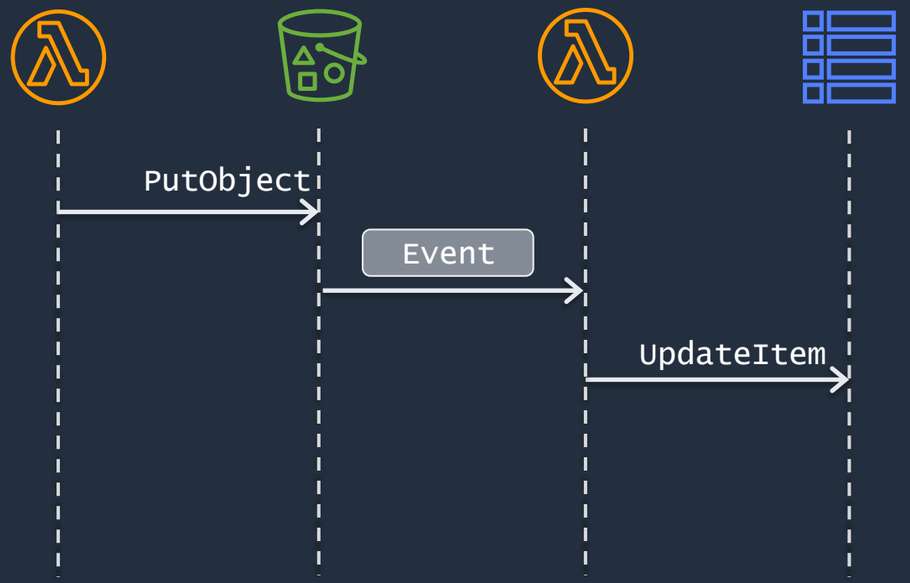
Waiting to wait

- *Should I perform synchronous work in my Lambda function?*
- Lambda charged by duration, per millisecond
  - Wait time = $$
- Functions calling other functions increases concurrency



Solution: Waiting to wait

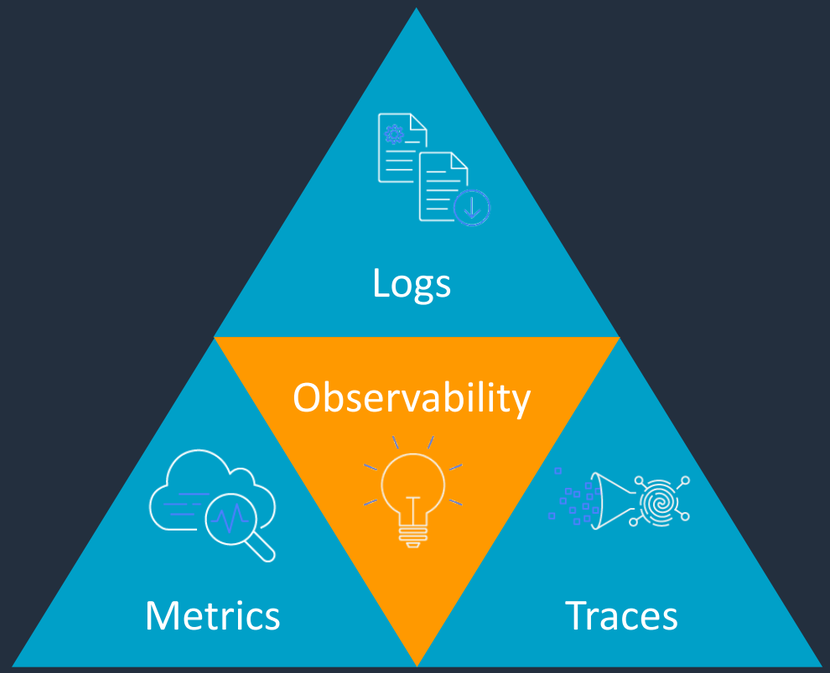
- Don’t wait (when possible)!
- Use fire and forget patterns to kick-off work, commit data, etc.
- Perform work in parallel
- Orchestrate multi-step process with AWS Step Functions
  - Pay for orchestration, not wait time



Observing my application

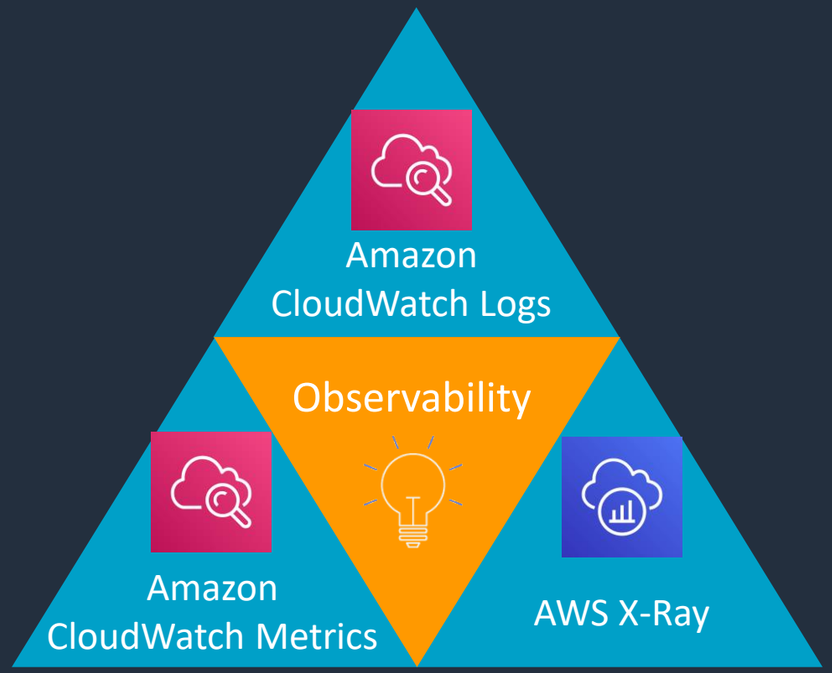
- *How do I manage logging and tracing in a serverless application?*
  - Serverless applications composed of numerous disparate, ephemeral services
  - Rely heavily on managed services

- Observability focuses on logs, metrics, and traces



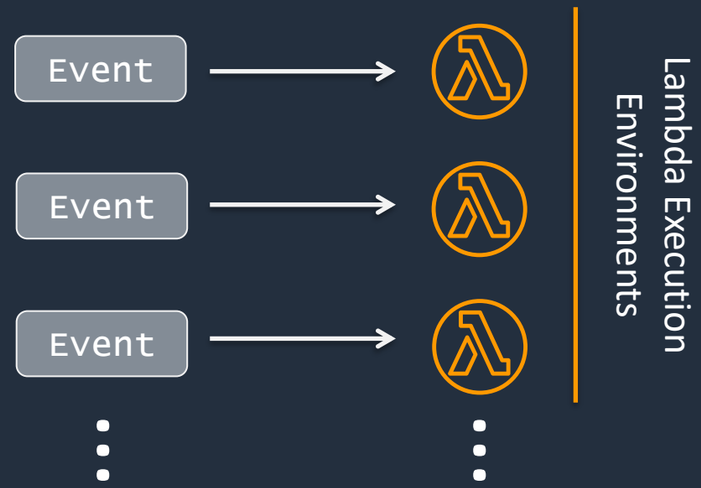
Solution: Observing my application

- Leverage AWS tools:
  - AWS CloudWatch
  - AWS X-Ray
- Tips:
  - Set a retention period for CloudWatch Logs
  - Use CloudWatch Embedded Metrics Format (EMF)
  - Add X-Ray permissions to execution role
- Lambda Extensions enables a broad array of partner tooling

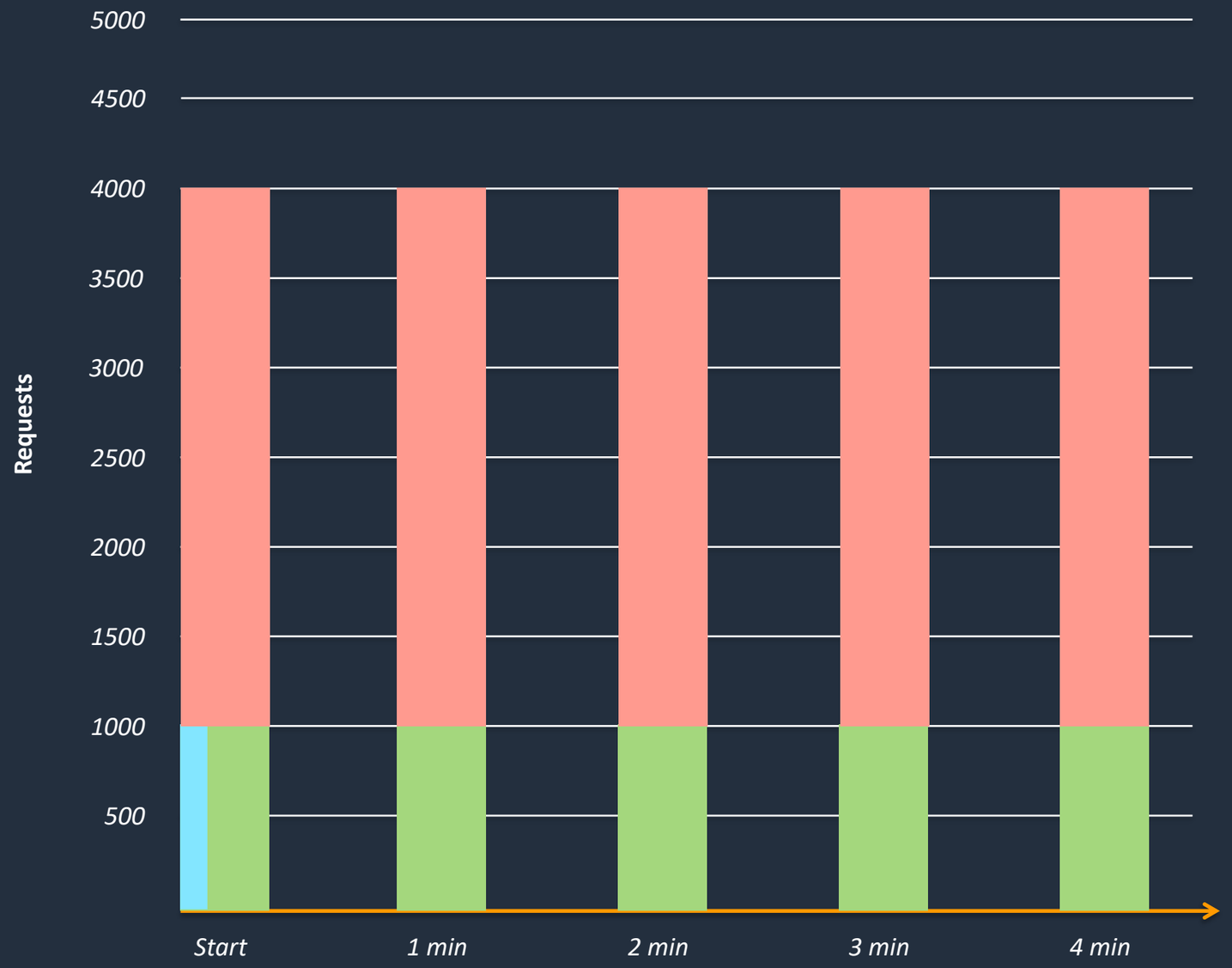


What’s concurrency? Should we raise it?

- *My team plans for X requests per second, what should I set concurrency to?*
- As demand increases, Lambda services increases concurrent executions
  - One function instance handles one request
  - Concurrency is a measure of concurrent executions
- Cumulative concurrency and burst are limited by AWS account per Region



Aside: Exploration of Lambda Concurrency & Burst Limits



Function

- Average duration: 1 second
- Requests / second: 4000
- Synchronous invocation

Account

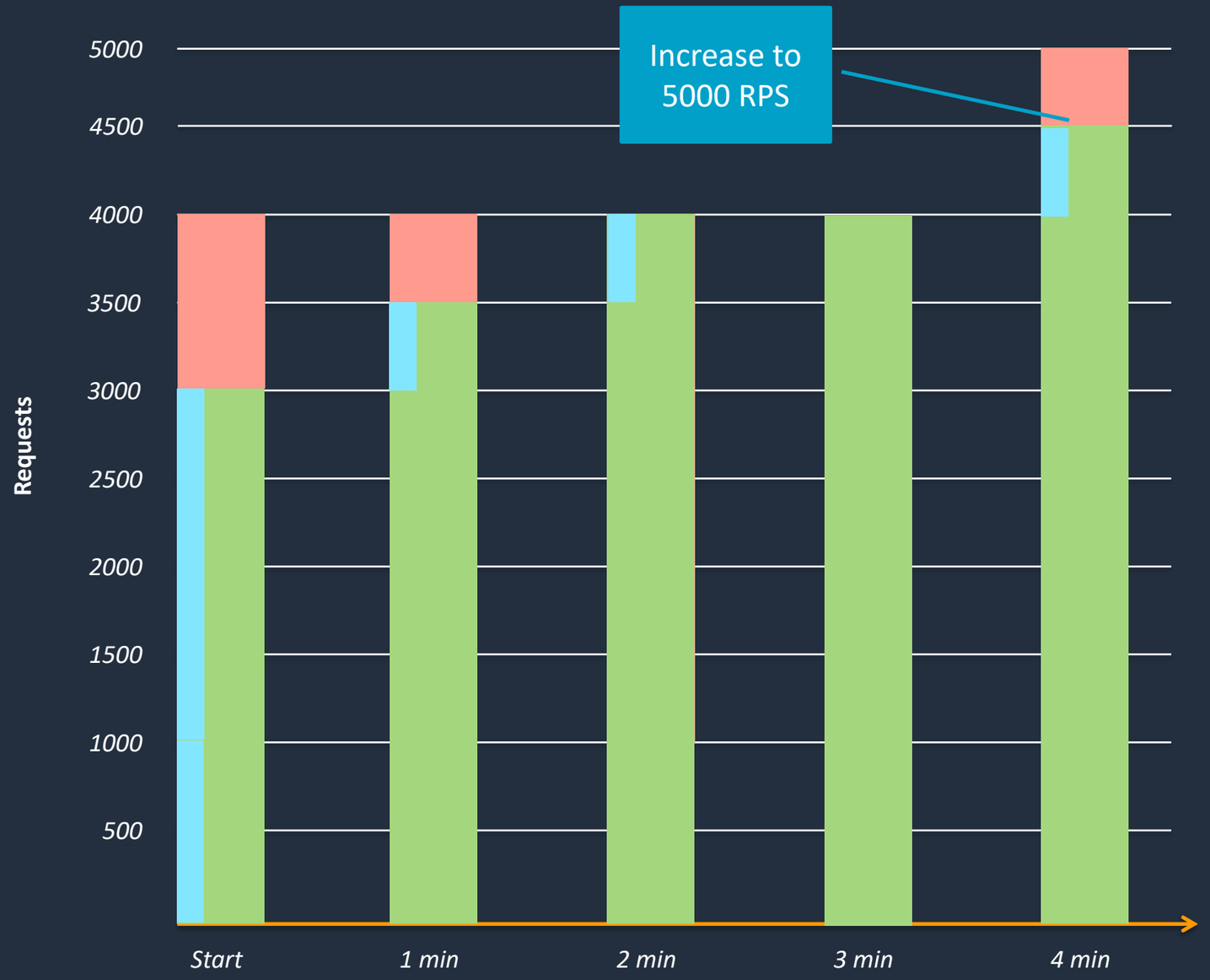
- N. Virginia (us-east-1)
  - Concurrency limit: 1000 (default)
  - Burst limit: 3000 (default)

Successful invocation

Cold start

Throttled invocation

Aside: Exploration of Lambda Concurrency & Burst Limits



Function

- Average duration: 1 second
- Requests / second: 4000
- Synchronous invocation

Account

- N. Virginia (us-east-1)
  - Concurrency limit: 5000 (default)
  - Burst limit: 3000 (default)

Successful invocation

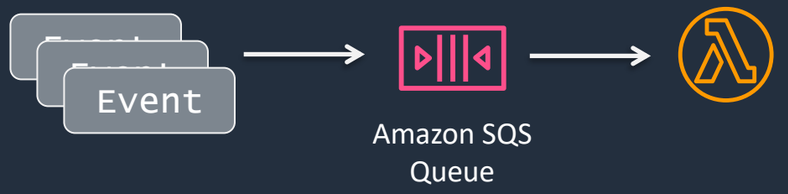
Cold start

Throttled invocation

aws

Solution: What's concurrency? Should we raise it?

- Understand Lambda scaling and concurrency
  - Concurrency is a function of duration and request rate
  - Load test to estimate duration
- Leverage Provisioned Concurrency for anticipated bursts of activity
  - Reserved Concurrency can be used to throttle, if needed
- Consider asynchronous processing to enable scaling



After the session: Keep Learning!

Architecting Serverless Solutions

https://www.aws.training/Details/eLearning?id=42594

AWS Lambda Foundations

https://www.aws.training/Details/eLearning?id=27197