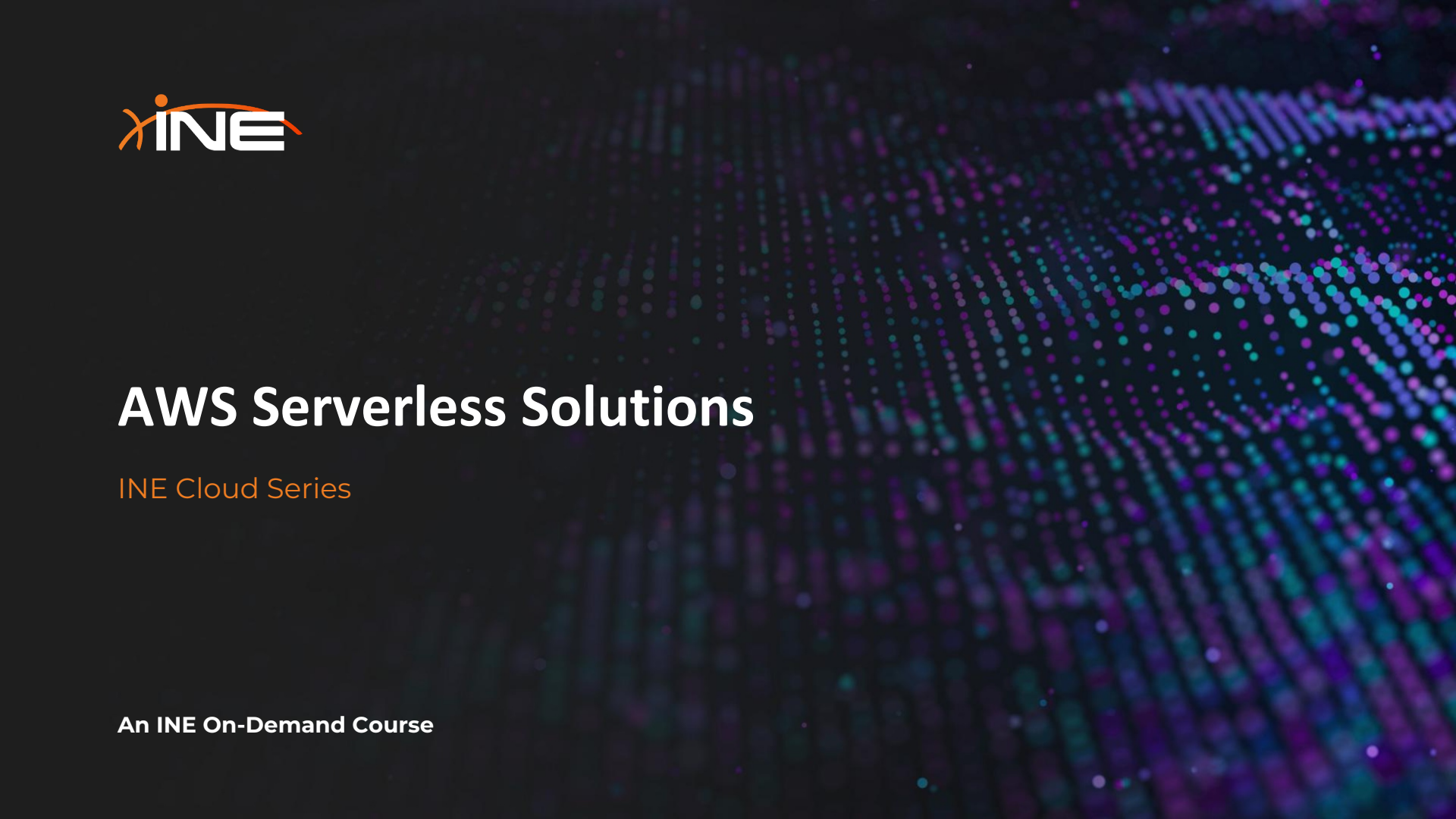


AWS Serverless Solutions

INE Cloud Series

An INE On-Demand Course

Brooks Seahorn

AWS Instructor

+ bseahorn@ine.com

+ @brooksseahorn

+ Search “Brooks Seahorn”

Topics Covered

- Introduction to serverless
- AWS Lambda
  - Best Practices & IaC
- Amazon EventBridge
- Amazon SNS & SQS
- Amazon API Gateway

Topics Covered

- Labs
  - Lab 1: Lambda
  - Lab 2: EventBridge
  - Lab 3: SNS
  - Lab 4: Capstone-Complete Solution

Serverless in AWS

INE Cloud Series

An INE On-Demand Course

Brooks Seahorn

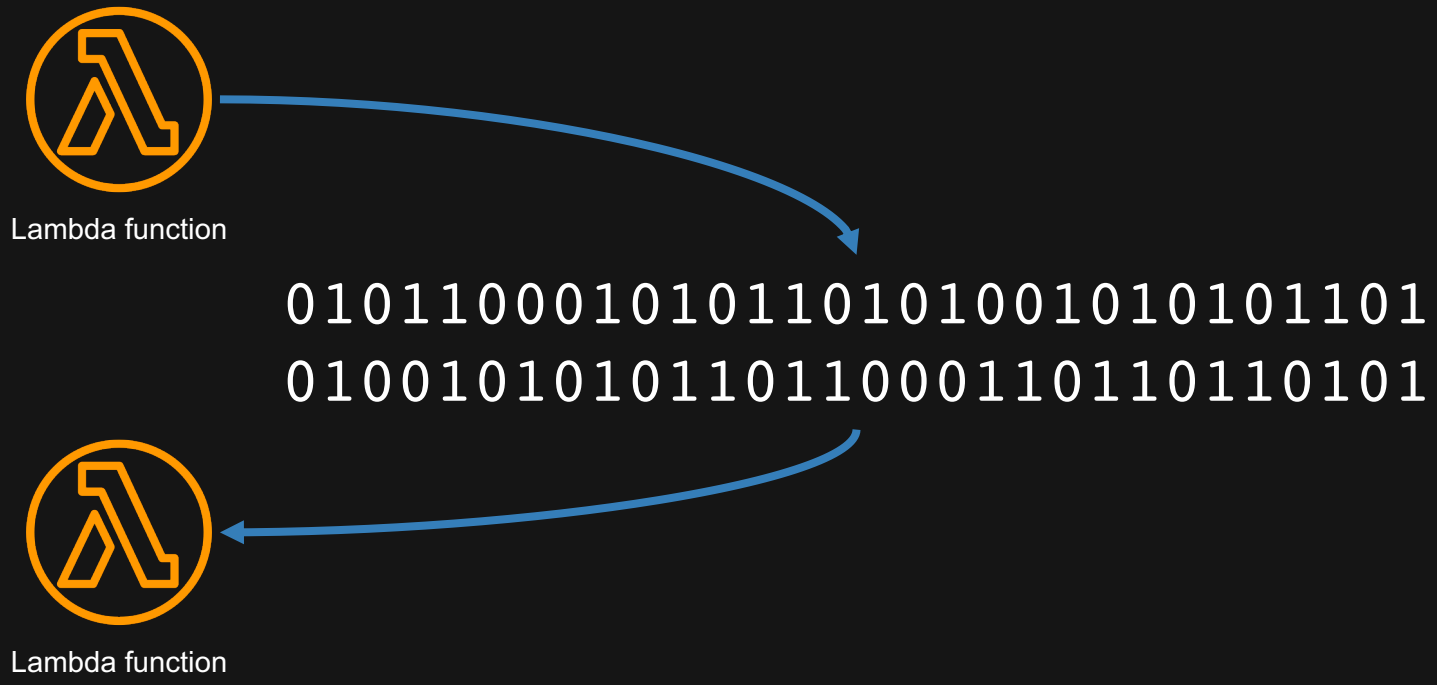
AWS Instructor

+ bseahorn@ine.com

+ @brooksseahorn

+ Search “Brooks Seahorn”

What is serverless


Lambda function
01011000101011010100101010110101001010101100011011011010
Lambda function

Amazon Simple Queue Service (Amazon SQS)

Standard: 14ms to just over 100ms
FIFO: 20ms to over 600ms

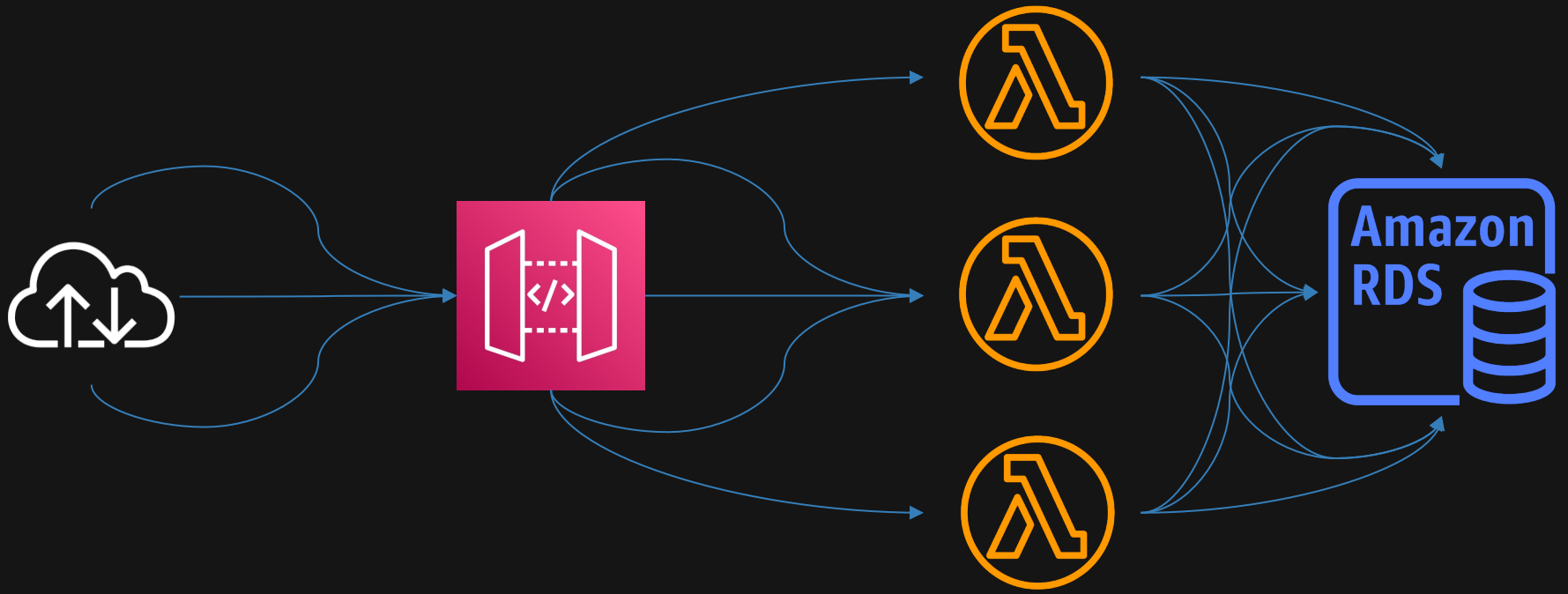
Amazon EventBridge

240ms to 800ms

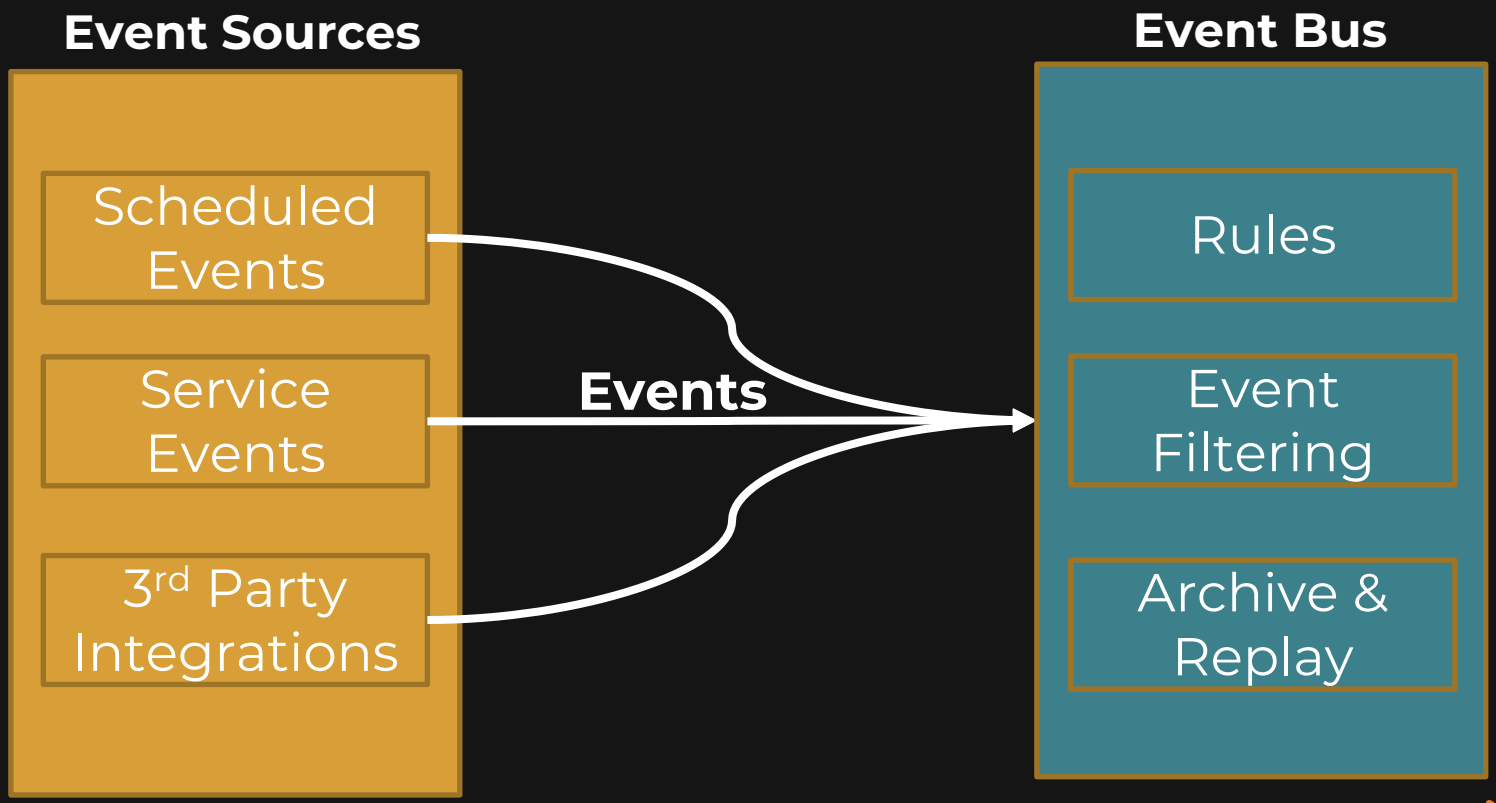
Amazon Kinesis

Standard data stream:
250ms to over 1400ms

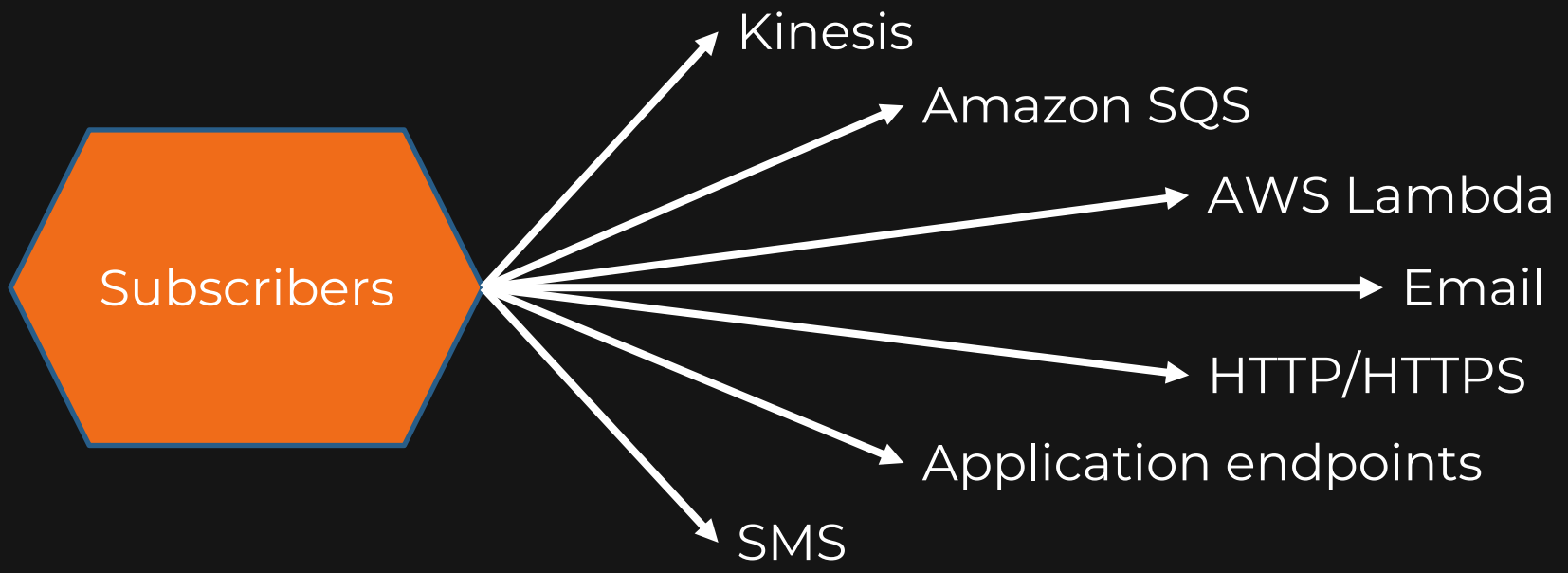
Amazon RDS Proxy



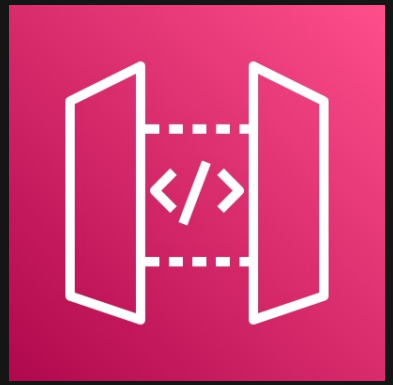
Amazon EventBridge



Subscribers

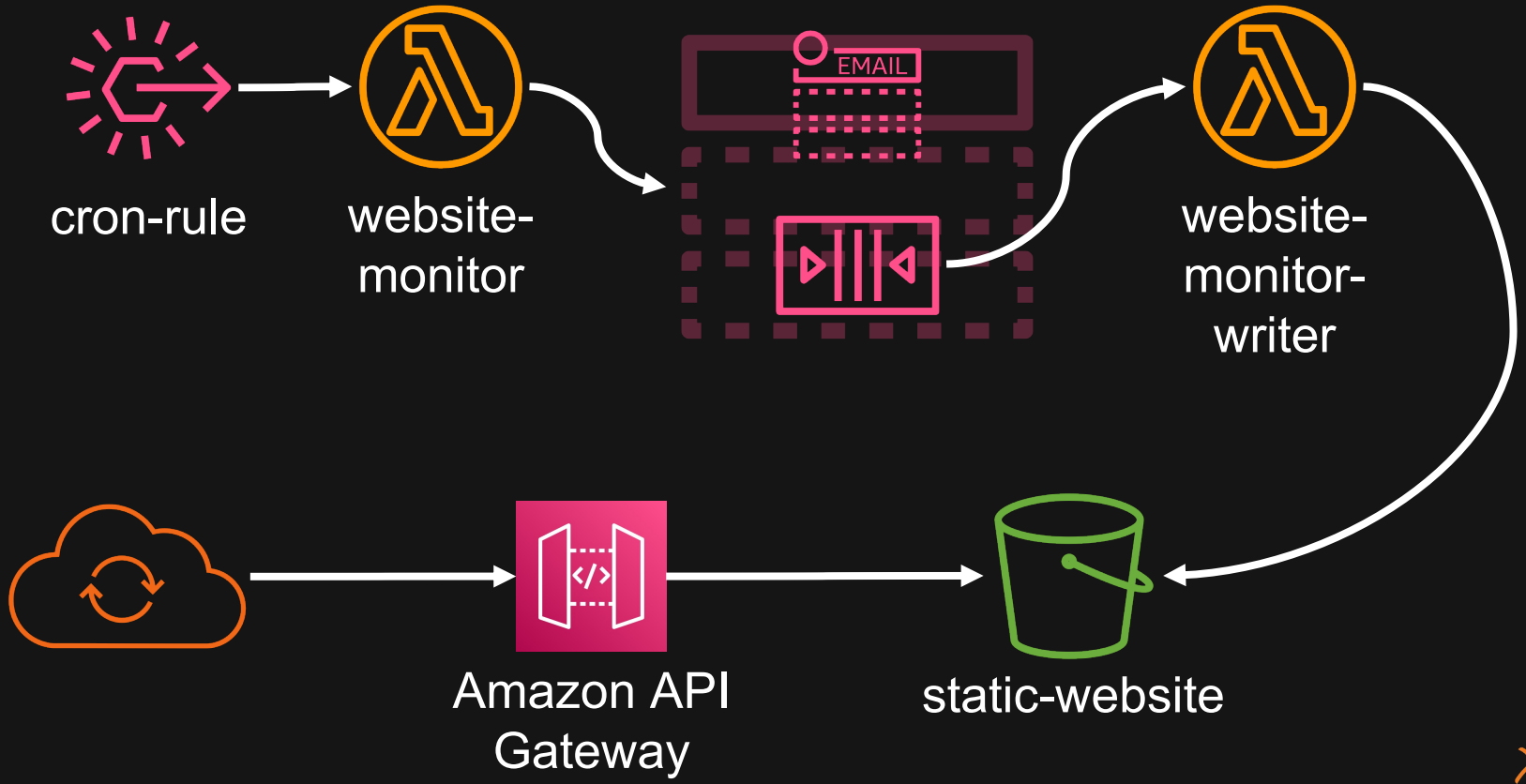


Amazon API Gateway



Amazon API Gateway

Website Monitoring System



Serverless in AWS Conclusion

INE Cloud Series

An INE On-Demand Course

Topics Covered

- Introduction to serverless
- AWS Lambda
- Amazon EventBridge
- Amazon SNS & SQS
- Amazon API Gateway
- Website Monitoring Solution

Recommendations

- Run Lab 4 again!
- Improve on that solution
- Review AWS' Serverless documentation
  - https://aws.amazon.com/serverless
- Review outside sources on serverless

Request from me to you

Leave a review

Thanks for watching!!!

A brief introduction to serverless

INE Cloud Series

An INE On-Demand Course

Topics

- What is serverless
- Real world serverless
- Serverless services in this course

What is serverless

What is serverless

What is serverless

01011000101011010100101010110101001010101101100011011010101

Real world serverless

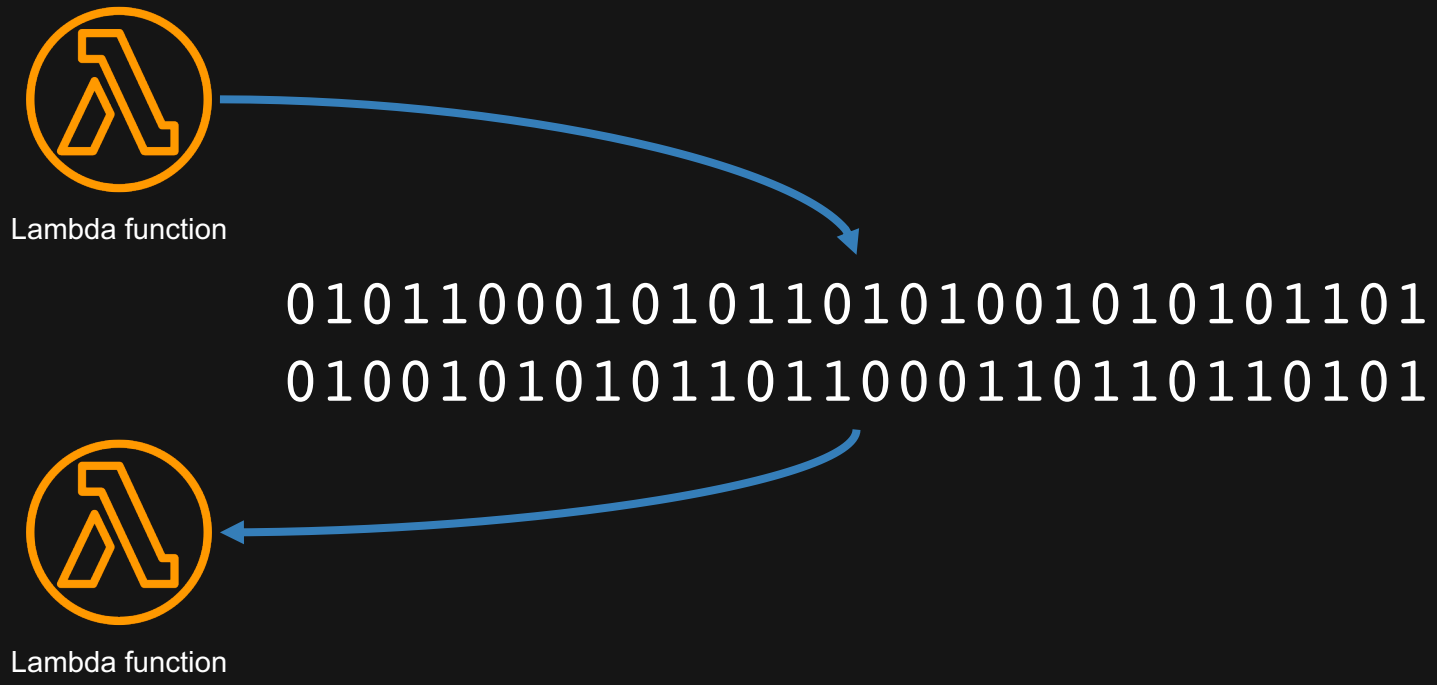
+ Small code base
+ Limited execution time
+ Managed architecture
+ Managed statefulness

```
use num::Complex;
use std::str::FromStr;
use image::ColorType;
use image::png::PNGEncoder;
use std::fs::File;
extern crate num_cpus;
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();

    if args.len() != 5 {
        eprintln!("Usage: {} FILE PIXELS UPPERLEFT LOWERRIGHT", args[0]);
        eprintln!("Example: {} mandel.png 1000x750 -1.20,0.35 -1,0.20", args[0]);
        std::process::exit(1);
    }

    let bounds = parse_pair(&args[2], 'x')
                    .expect("error parsing image dimensions");
    let upper_left = parse_complex(&args[3])
                    .expect("error parsing upper left corner point");
    let lower_right = parse_complex(&args[4])
                    .expect("error parsing lower right corner point");

    let mut pixels = vec![0; bounds.0 * bounds.1];

    // Commented out the following line.  It is the nonconcurent implementation.
    // render(&mut pixels, bounds, upper_left, lower_right);
    // The following is the concurrent implementation. It breaks the image
    // into horizontal bands that each are handled by a separate thread.
    let threads = num_cpus::get();
    println!("Running the set with {} thread(s).", threads );
    let rows_per_band = bounds.1 / threads + 1;
    {
        let bands: Vec<&mut [u8]> =
            pixels.chunks_mut(rows_per_band * bounds.0).collect();
        crossbeam::scope(|spawner| {
            for (i, band) in bands.into_iter().enumerate() {
                let top = rows_per_band * i;
                let height = band.len() / bounds.0;
```

```rust
use num::Complex;
use std::str::FromStr;
use image::ColorType;
use image::png::PNGEncoder;
use std::fs::File;
extern crate num_cpus;
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();

    if args.len() != 5 {
        eprintln!("Usage: {} FILE PIXELS UPPERLEFT LOWERRIGHT", args[0]);
        eprintln!("Example: {} mandel.png 1000x750 -1.20,0.35 -1,0.20", args[0]);
        std::process::exit(1);
    }

    let bounds = parse_pair(&args[2], 'x')
                 .expect("error parsing image dimensions");
    let upper_left = parse_complex(&args[3])
                 .expect("error parsing upper left corner point");
    let lower_right = parse_complex(&args[4])
                 .expect("error parsing lower right corner point");

    let mut pixels = vec![0; bounds.0 * bounds.1];

    // Commented out the following line.  It is the nonconcurent implementation.
    // render(&mut pixels, bounds, upper_left, lower_right);
    // The following is the concurrent implementation. It breaks the image
    // into horizontal bands that each are handled by a separate thread.
    let threads = num_cpus::get();
    println!("Running the set with {} thread(s).", threads );
    let rows_per_band = bounds.1 / threads + 1;
    {
        let bands: Vec<&mut [u8]> =
            pixels.chunks_mut(rows_per_band * bounds.0).collect();
        crossbeam::scope(|spawner| {
            for (i, band) in bands.into_iter().enumerate() {
                let top = rows_per_band * i;
                let height = band.len() / bounds.0;
```

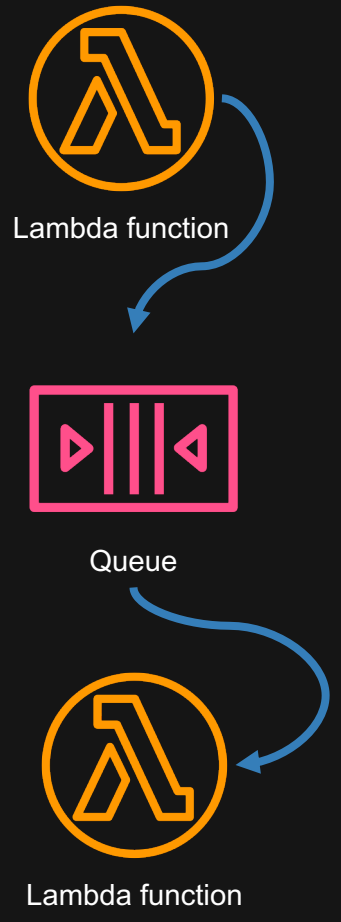
Lambda function

Lambda function


Lambda function
01011000101011010100101010110101001010101100011011011010
Lambda function


Lambda function
Queue
Lambda function

```rust
use num::Complex;
use std::str::FromStr;
use image::ColorType;
use image::png::PNGEncoder;
use std::fs::File;
extern crate num_cpus;
use std::env;

fn main() {
    let args: Vec<String> = env::args().collect();

    if args.len() != 5 {
        eprintln!("Usage: {} FILE PIXELS UPPERLEFT LOWERRIGHT", args[0]);
        eprintln!("Example: {} mandel.png 1000x750 -1.20,0.35 -1,0.20", args[0]);
        std::process::exit(1);
    }

    let bounds = parse_pair(&args[2], 'x')
                 .expect("error parsing image dimensions");
    let upper_left = parse_complex(&args[3])
                 .expect("error parsing upper left corner point");
    let lower_right = parse_complex(&args[4])
                 .expect("error parsing lower right corner point");

    let mut pixels = vec![0; bounds.0 * bounds.1];

    // Commented out the following line.  It is the nonconcurent implementation.
    // render(&mut pixels, bounds, upper_left, lower_right);
    // The following is the concurrent implementation. It breaks the image
    // into horizontal bands that each are handled by a separate thread.
    let threads = num_cpus::get();
    println!("Running the set with {} thread(s).", threads );
    let rows_per_band = bounds.1 / threads + 1;
    {
        let bands: Vec<&mut [u8]> =
            pixels.chunks_mut(rows_per_band * bounds.0).collect();
        crossbeam::scope(|spawner| {
            for (i, band) in bands.into_iter().enumerate() {
                let top = rows_per_band * i;
                let height = band.len() / bounds.0;
```



Increased management overhead

Increased latency

Amazon Simple Queue Service (Amazon SQS)

Standard: 14ms to just over 100ms
FIFO: 20ms to over 600ms

Amazon EventBridge

240ms to 800ms

Amazon Kinesis

Standard data stream:
250ms to over 1400ms

Real world serverless

- Small code base
- Limited time execution
- Managed architecture
- Managed statefulness

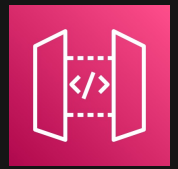
Real world serverless

- Serverless IS an architectural option
- Serverless IS NOT an idealized, perfected end state

Serverless Services in AWS

Amazon EventBridge

AWS Lambda



Amazon API Gateway

Amazon Simple Notification Service (Amazon SNS)

Amazon Simple Queue Service (Amazon SQS)

Project: Website Monitoring System

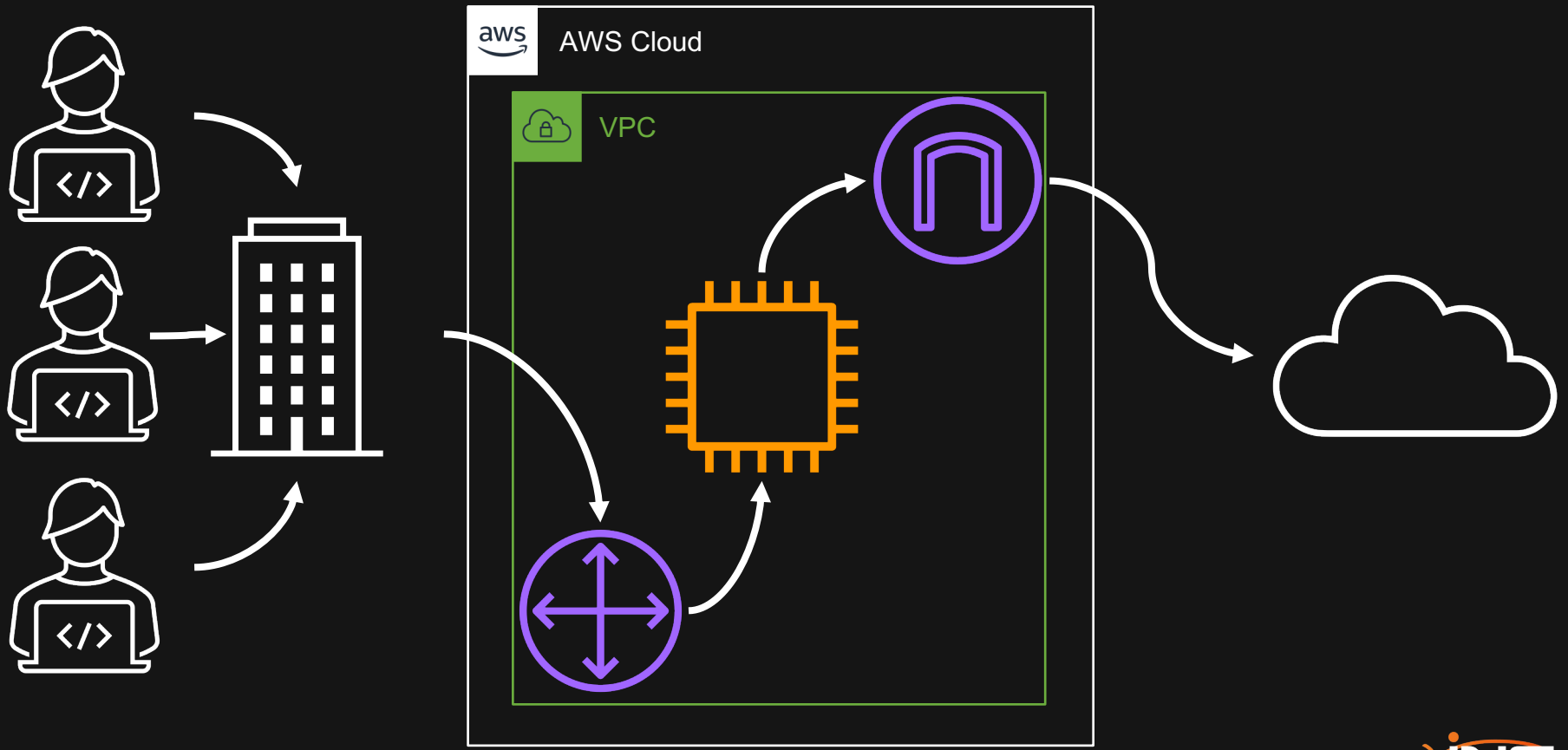
INE Cloud Series

An INE On-Demand Course


amazon.com®
overstock™
Rakuten

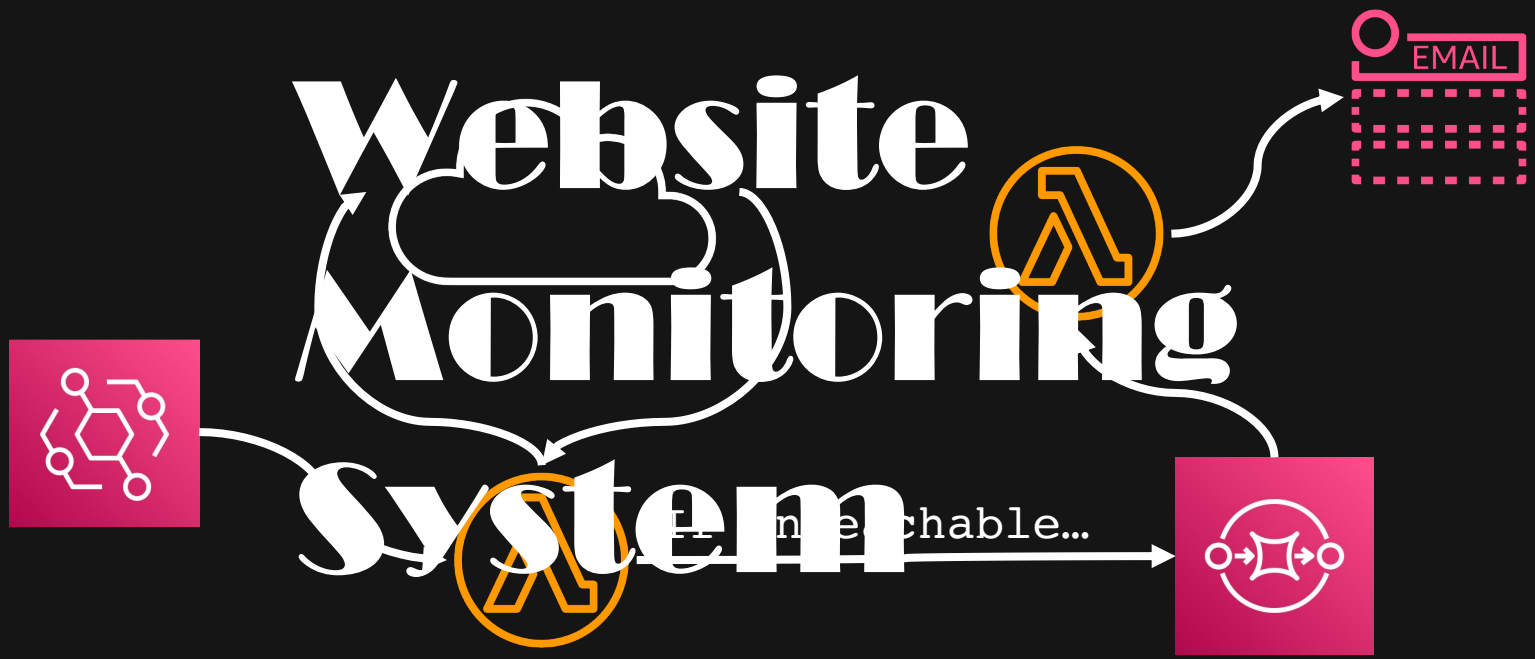

Pick two.
Low prices.
Fast delivery
amazon
203

Website is unreachable…


aws
AWS Cloud
VPC

Website is unreachable…

Website Monitoring System


Website
Monitoring
System
EMAIL
chable…

AWS Lambda

INE Cloud Series

An INE On-Demand Course

Topics

- AWS Lambda
- Usage
- Details
- Costs

AWS Lambda Usage

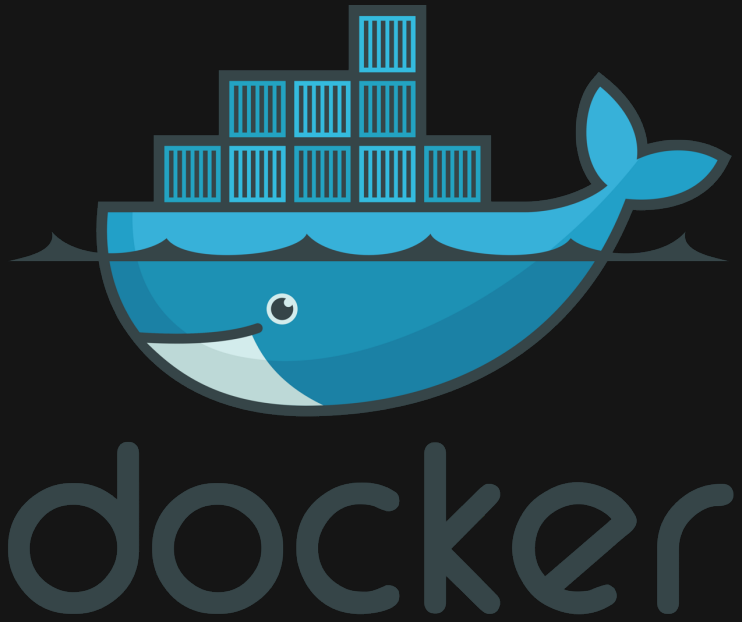
- Web applications
- Backends (web, application, and mobile)
- Event-driven architectures
- Extract-transform-load

AWS Lambda

- Serverless execution code
  - Container based execution
  - Managed VPC, multi-AZ deployment
- Trigger Lambda from many AWS services
  - Kinesis stream
  - Write to an S3 bucket
  - API Gateway
  - EventBridge scheduled call
  - Target from SQS

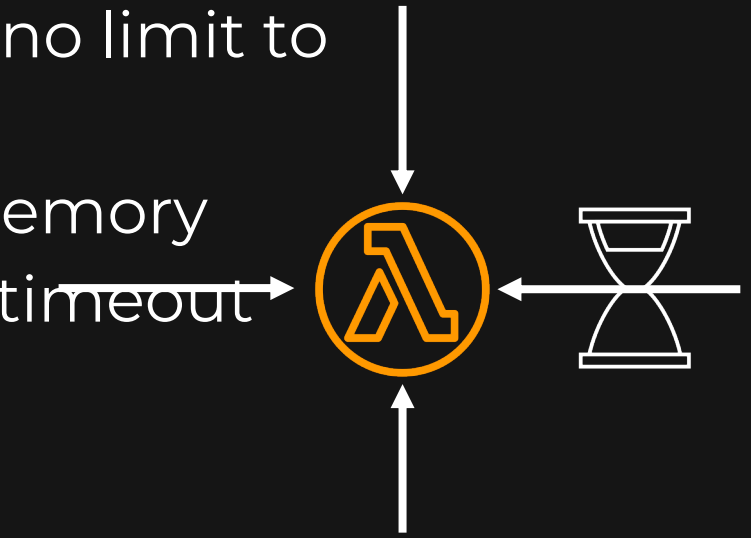
AWS Lambda

AWS Lambda



AWS Lambda

- Automatic scaling w/ no limit to capability
- 128MB to 10,240MB Memory
- 15-minute maximum timeout
- Ephemeral storage
  - 512MB to 10GB
  - Presented as `/tmp`

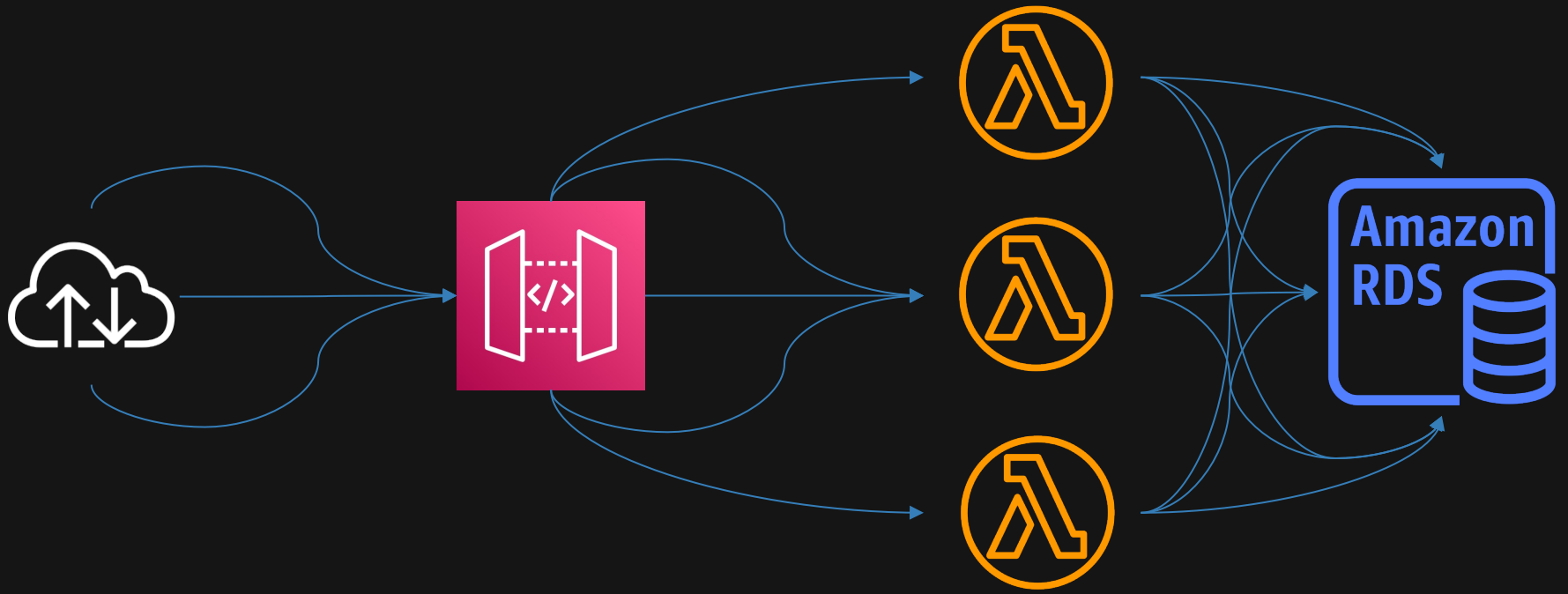


Check for ephemeral data!!!

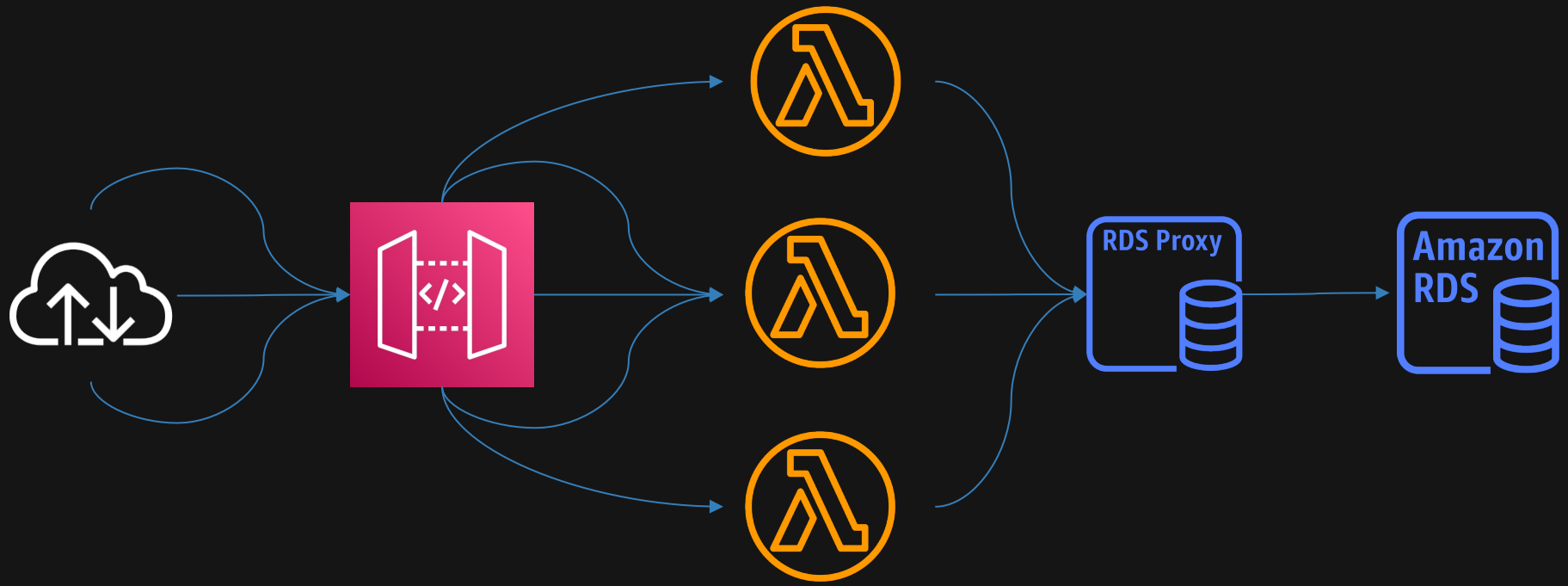
AWS Lambda

- Automatic scaling w/ no limit to capability
- 128MB to 10,240MB Memory
- 15-minute maximum timeout
- Ephemeral storage
  - 512MB to 10GB
  - Presented as `/tmp`
- Persistent storage
  - RDS connectivity via RDS Proxy

Amazon RDS Proxy



Amazon RDS Proxy



AWS Lambda

- Automatic scaling w/ no limit to capability
- Ephemeral storage
  - 512MB to 10GB
  - Presented as `/tmp`
- Persistent storage
  - RDS connectivity via RDS Proxy
  - Amazon EFS

AWS Lambda Cost

- Execution (128MB):
  - $0.0000000021 per 1ms
  - First 6 Billion GB-sec/month
- Ephemeral storage
  - $0.0000000309 for every GB-second

Conclusion

- AWS Lambda
- Usage
- Details
- Costs

AWS Lambda Demonstration

INE Cloud Series

An INE On-Demand Course

Demonstration

- Web site monitor
- Initial build
- Add custom code
- Create a functional test
- Monitor with CloudWatch

AWS Lambda Best/Worst Practices

INE Cloud Series

An INE On-Demand Course

Topics

- Best practices
- Bad practices

Best Practices – Code Organization

```
21  def lambda_handler(event, context):      You, 2 days ago • Add files via upload …
22      SITES = [item for item in temp_sites.split(",") if item]
23
24      for site in SITES:
25          NOW = str(datetime.now())
26          print('Checking ', site)
27          try:
28              req = Request(site, headers={'User-Agent': 'AWS Lambda'})
29              if not validate(str(urlopen(req).read())):
30                  raise Exception('Validation failed')
31          except:
32              print('Check failed!')
33              raise
34          else:
35              print('Check passed!' + NOW)
36              # return NOW
37          finally:
38              print(NOW)
```

Best Practices – Code Organization

```
20  def check_site(event, context):
21      SITES = [item for item in temp_sites.split(",") if item]
22
23      for site in SITES:
24          NOW = str(datetime.now())
25          print('Checking ', site)
26          try:
27              req = Request(site, headers={'User-Agent': 'AWS Lambda'})
28              if not validate(str(urlopen(req).read())):
29                  raise Exception('Validation failed')
30          except:
31              print('Check failed!')
32              raise
33          else:
34              print('Check passed!' + NOW)
35              # return NOW
36          finally:
37              print(NOW)
38
39  def lambda_handler(event, context):
40      check_site(event, context)
```

Best Practices – Performance

```
Function Logs
START RequestId: 8d0c1ad8-1caa-4314-b5ff-d1e4d6190c40 Version:
$LATEST
Checking  https://www.amazon.com
Check passed!2022-11-07 18:30:08.104088
2022-11-07 18:30:08.104088
Checking   https://www.ebay.com
Check passed!2022-11-07 18:30:08.291278
2022-11-07 18:30:08.291278
END RequestId: 8d0c1ad8-1caa-4314-b5ff-d1e4d6190c40
REPORT RequestId: 8d0c1ad8-1caa-4314-b5ff-d1e4d6190c40
       Duration: 875.16 ms Billed Duration: 876 ms    Memory
Size: 128 MB  Max Memory Used: 45 MB
```

Best Practices – Performance

```
Duration: 875.16 ms
Billed Duration: 876 ms
Memory Size: 128 MB
Max Memory Used: 45 MB
```

Best Practices – Performance

AWS Lambda Power Tuner

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

Best Practices – Other

- Reuse: Subsequent calls to your function can reuse resources
- Use environment variables
- Minimize deployment size
  - Use Lambda Layers
- Avoid duplicated event processing (idempotent code)
- SQS visibility timeout < invocation time

Best Practices – Other

- Use most restrictive permission
- Use versioning with Lambdas
- Remove unused Lambdas

Best Practices – Performance

- CloudWatch Metrics
  - **Invocations:** # times the function is called
  - **Errors:** Invocation # that error
  - **DestinaltionDeliveryFailures:** Lambda fails to send an event to a destination
  - **Throttles:** Rejected invocations
  - **Duration:** Processing time
  - **ConcurrentExecutions:** # of function instances processing

Worst Practices

- Creating a Lambda monolith
- Recursion (infinite loops with Lambda)
- Lambda to Lambda calls

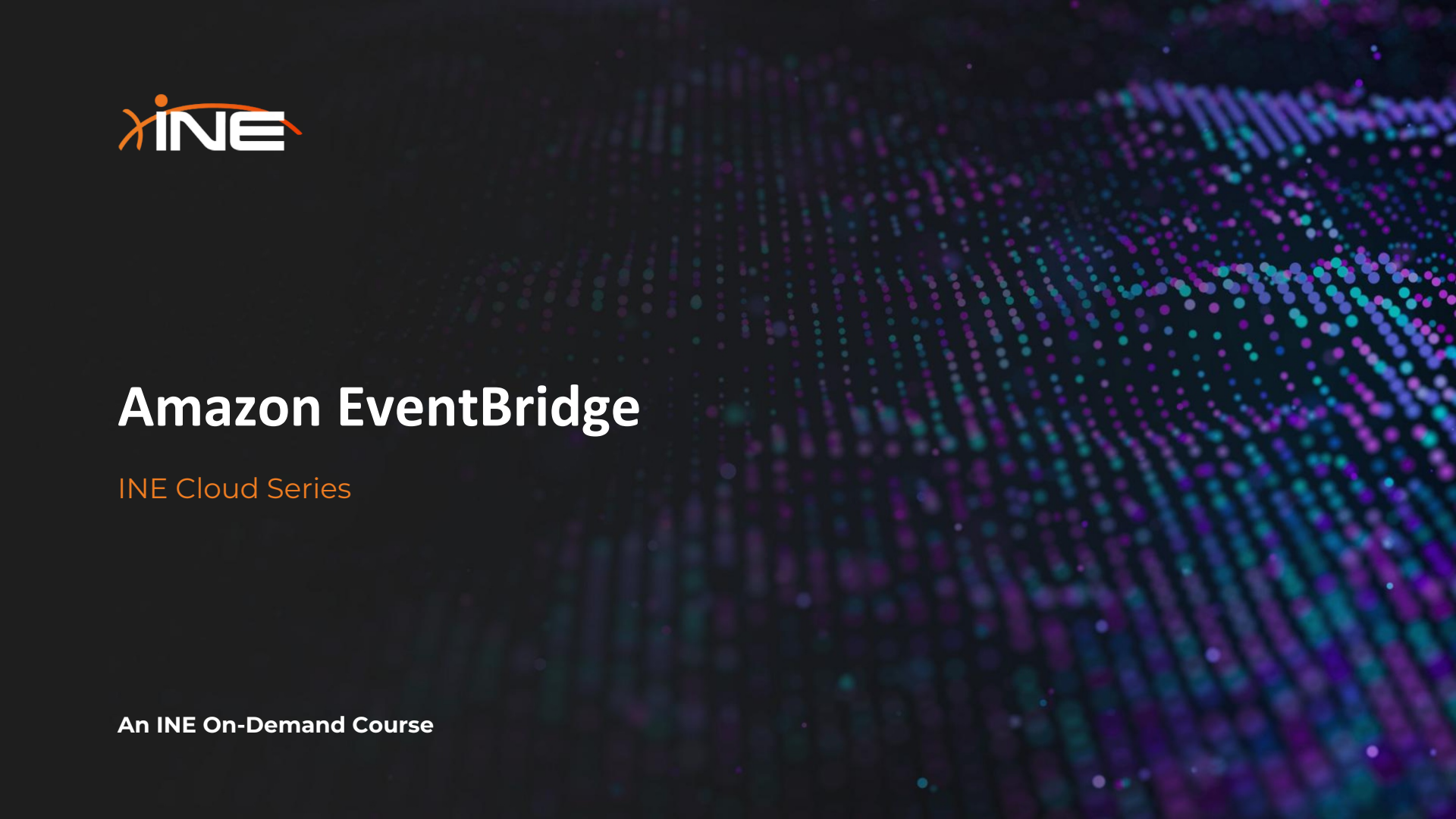
Amazon Lambda – Infrastructure as Code

INE Cloud Series

An INE On-Demand Course

Topics

- Security definitions
- Lambda definition
- Archive utility


Amazon EventBridge
INE Cloud Series
An INE On-Demand Course

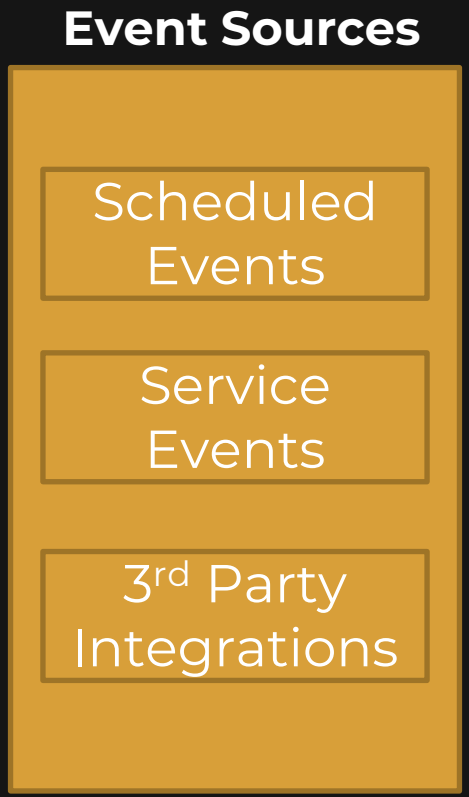
Topics

- Amazon EventBridge
- Architecture
- Demonstration

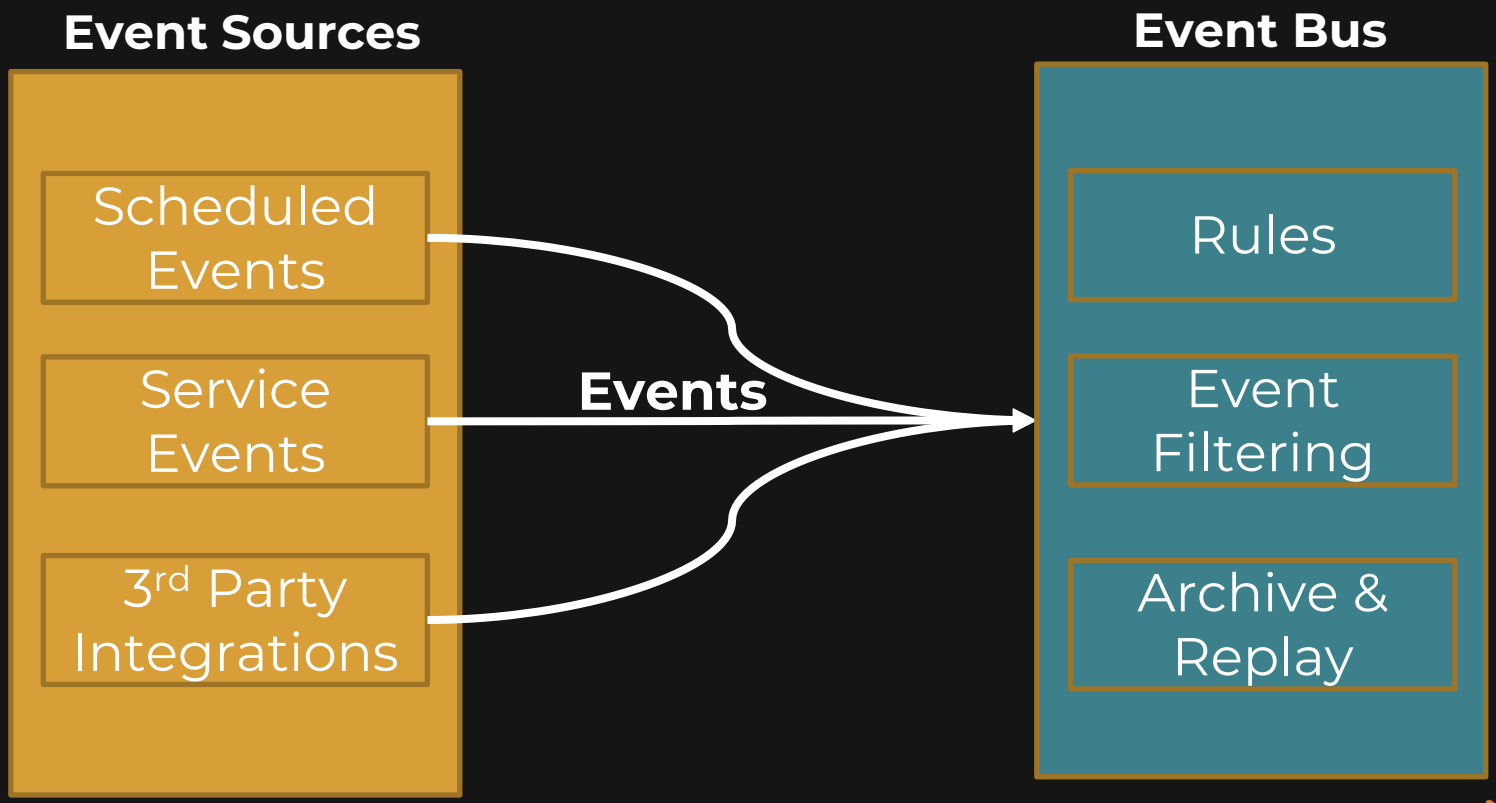
Amazon EventBridge

- Build event-driven architectures
- Connect SaaS (3rd party providers)
- Low code, low overhead

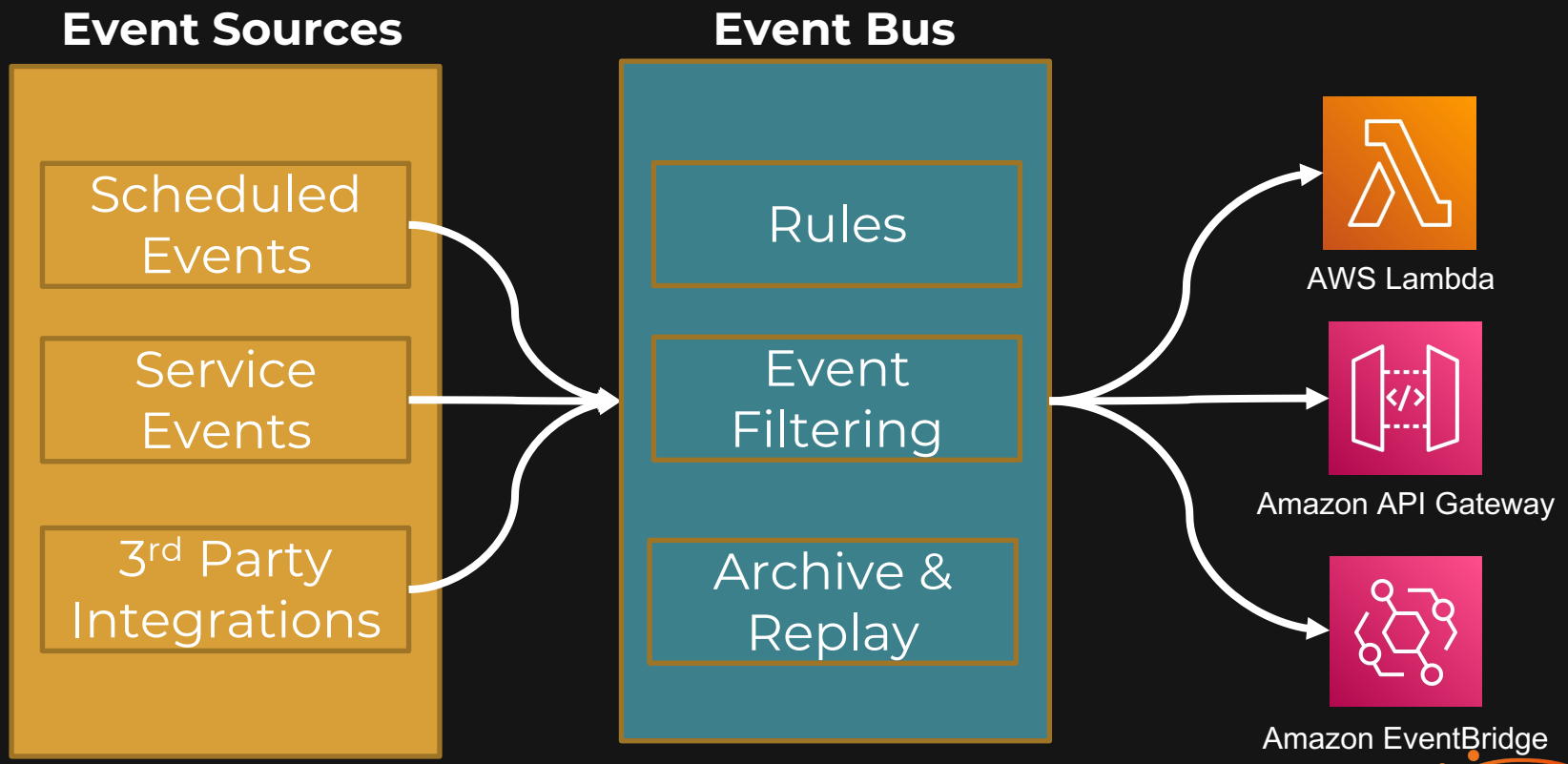
Amazon EventBridge



Amazon EventBridge



Amazon EventBridge



Demonstration

- Create a rule in EventBridge
- Call the Website Monitor on a schedule

Amazon EventBridge - Update

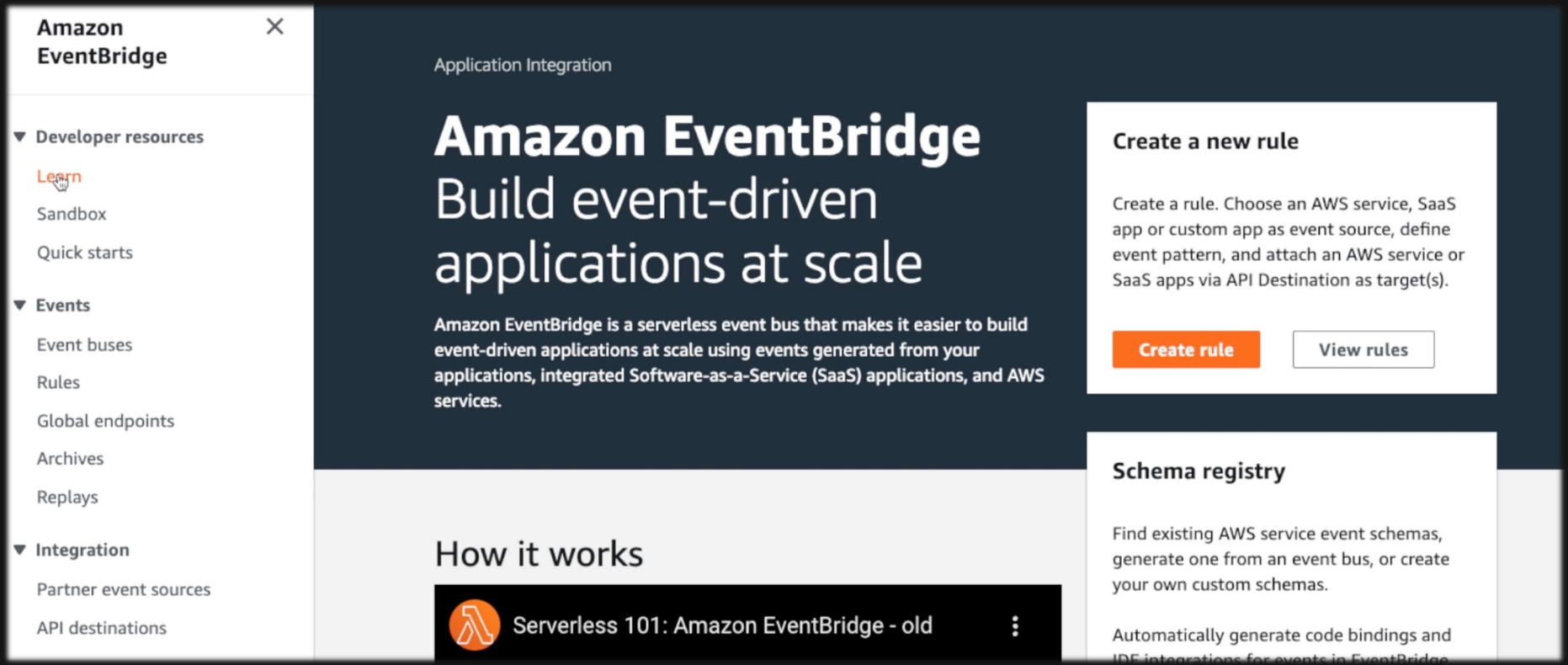
INE Cloud Series

An INE On-Demand Course

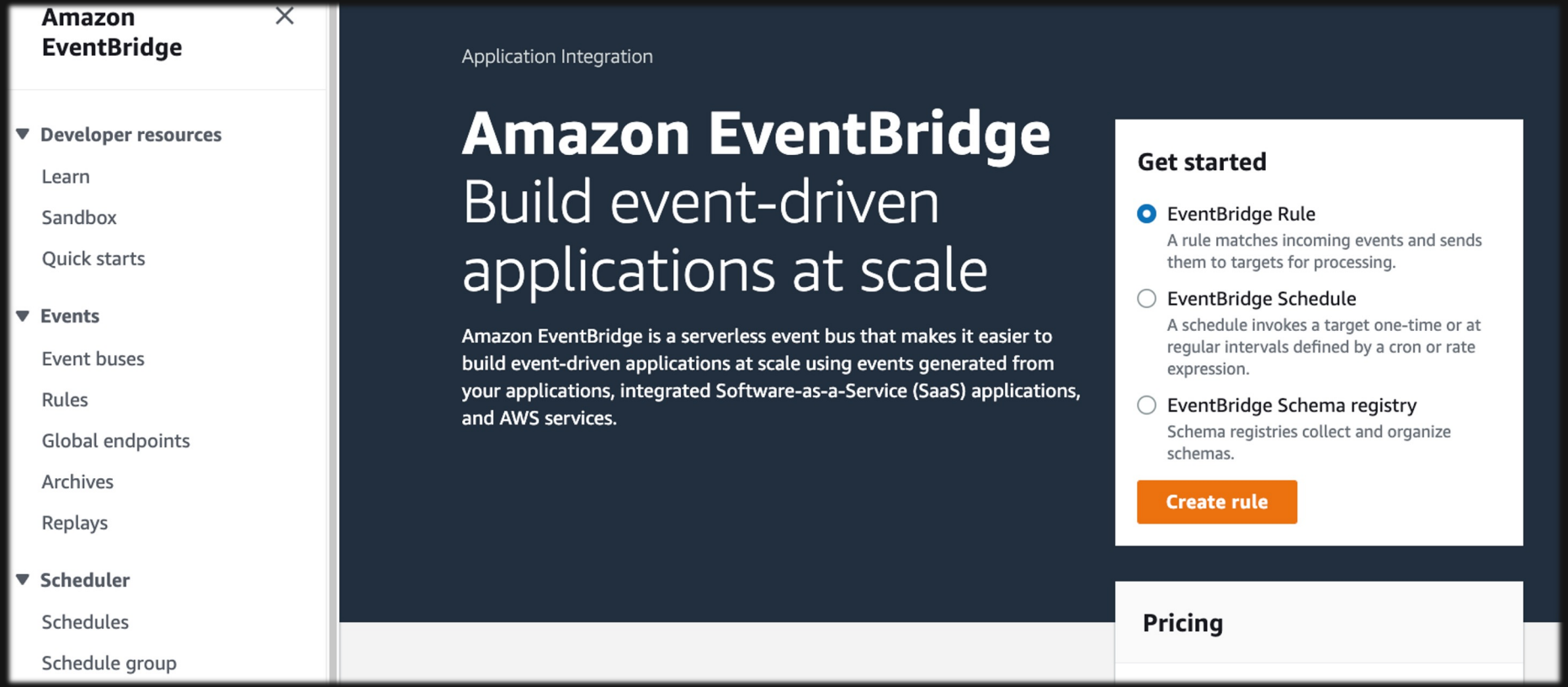
Topics

- November ‘22 Update

Amazon EventBridge



Amazon EventBridge



Conclusion

- Schedules are now 1st class citizens in the EventBridge Scheduler service
- Create services separate from the rules themselves

Amazon Simple Notification Service & Demonstration

INE Cloud Series

An INE On-Demand Course

Topics

- Amazon SNS
- Topic
- Subscriber
- Demonstration

Amazon SNS

- Amazon Simple Notification Service
- Pub/Sub, SMS, & mobile push
- App-2-App (A2A) & App-2-Person (A2P)

This is a message about something real important.

ECS!!!

Is Python performant?

Cloud networking…where does this stuff go?!?!?!

No no, you're crazy. Write everything in VBA.

Mechanical keyboards….great for Zoom calls. ;)

Mitchell!!!

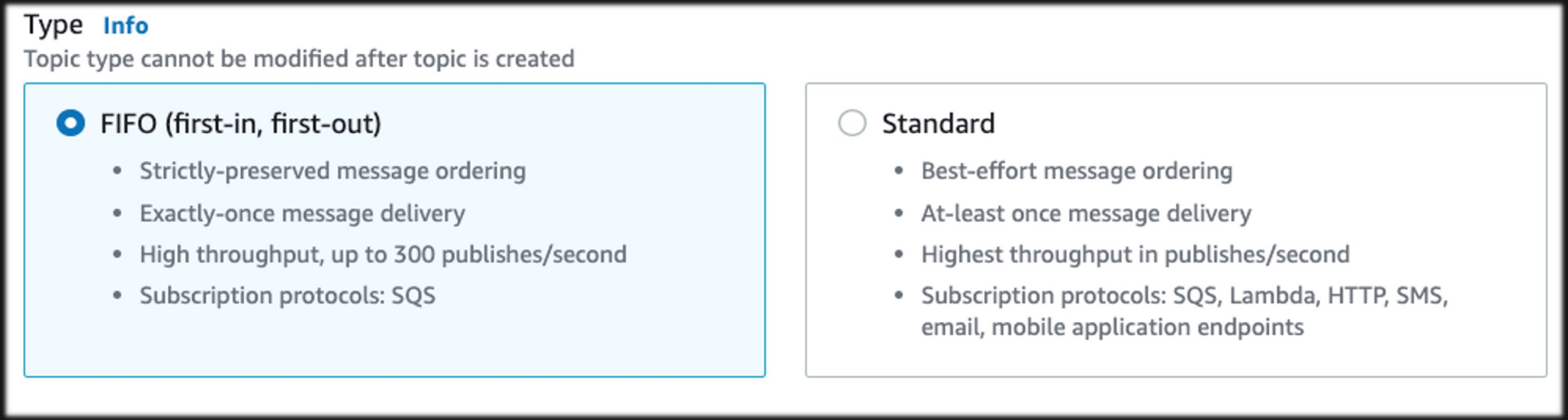
Amazon SNS

- Topic – Message Channel
- Subscriber – Delivery Endpoint

Topics

- Message channel
- Creating a queue: ordering, encryption, access, logging

Topics


Type Info
Topic type cannot be modified after topic is created
FIFO (first-in, first-out)
Strictly-preserved message ordering
Exactly-once message delivery
High throughput, up to 300 publishes/second
Subscription protocols: SQS
Standard
Best-effort message ordering
At-least once message delivery
Highest throughput in publishes/second
Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Topics

```
PublishRequest pubRequest =
PublishRequest.builder()
    .topicArn(topicArn)
    .subject(subject)
    .message(payload)
    .messageGroupId(groupId)
    .messageDeduplicationId(dedupId)
    .messageAttributes(attributes)
    .build();
```

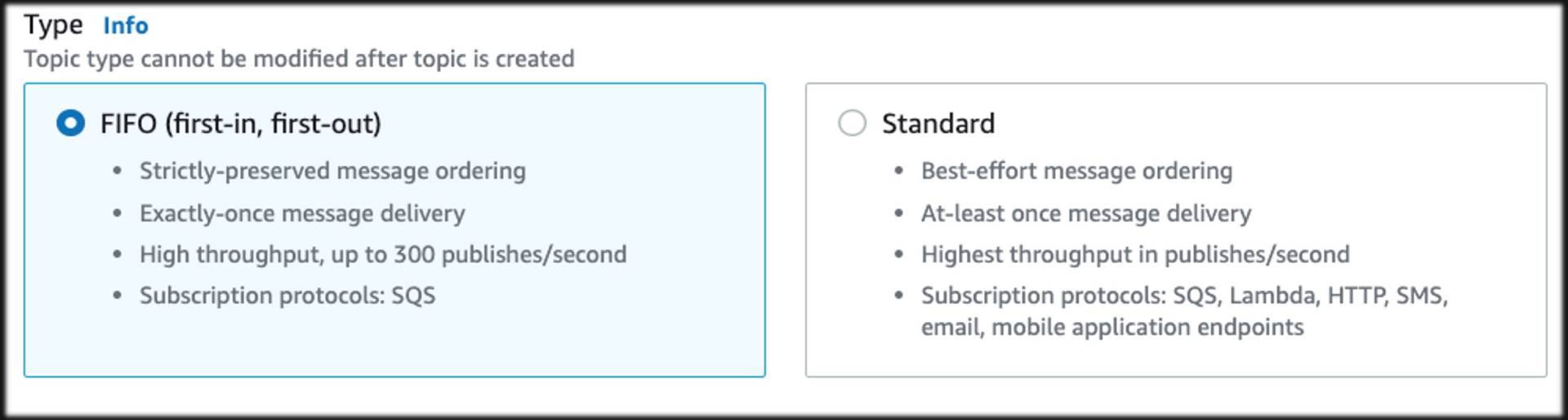
Topics

```
PublishRequest pubRequest =
PublishRequest.builder()
    .topicArn(topicArn)
    .subject(subject)
    .message(payload)
    .messageGroupId(groupId)
    .messageDeduplicationId(dedupId)
    .messageAttributes(attributes)
    .build();
```

Topics

```
PublishRequest pubRequest =
PublishRequest.builder()
    .topicArn(topicArn)
    .subject(subject)
    .message(payload)
    .messageGroupId(groupId)
    .messageDeduplicationId(dedupId)
    .messageAttributes(attributes)
    .build();
```

Topics


Type Info
Topic type cannot be modified after topic is created
FIFO (first-in, first-out)
Strictly-preserved message ordering
Exactly-once message delivery
High throughput, up to 300 publishes/second
Subscription protocols: SQS
Standard
Best-effort message ordering
At-least once message delivery
Highest throughput in publishes/second
Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

Optional Features

- Encryption
  - Automatic, only on queue
- Access policies
  - Who can publish/subscribe
- Data protection policies
- Filters
  - 200 per topic, 10,000 per account
- HTTP/S retry policies
- Delivery logging
- Tags

Optional Features

FIFO

- Encryption
- Access policies
- Delivery logging
- Tags

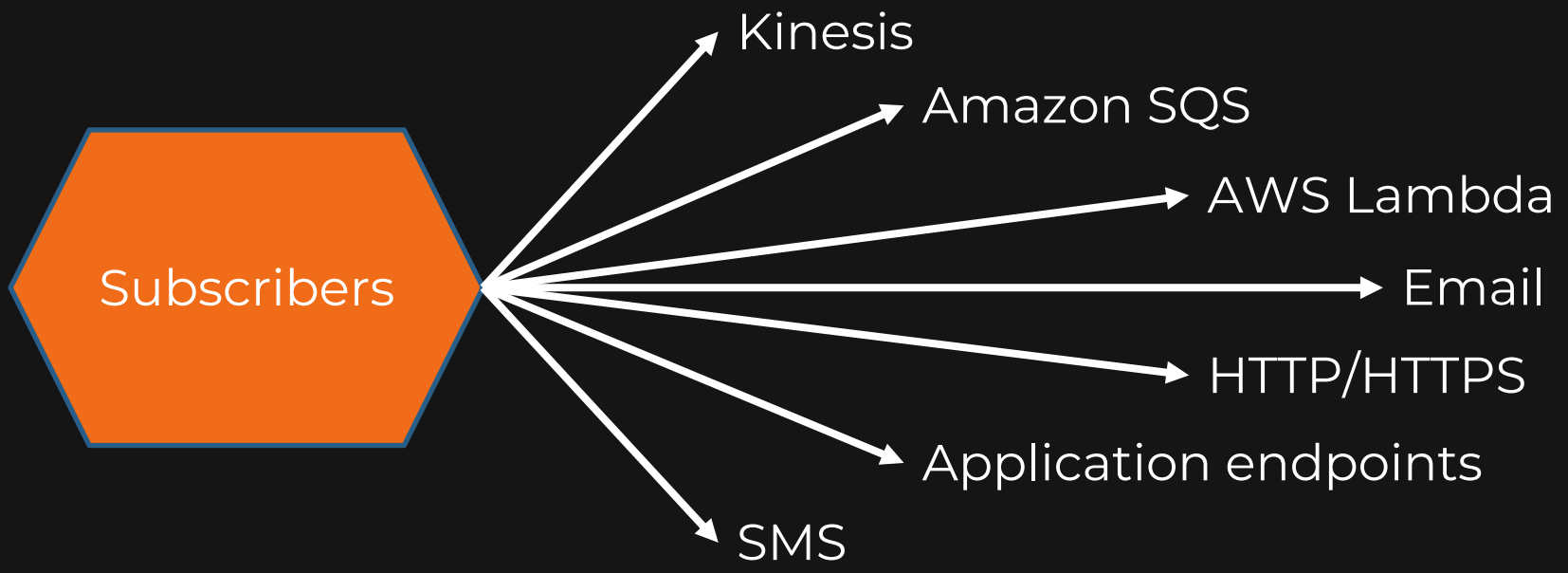
Standard

- Encryption
- Access policies
- Data protection policies
- HTTP/S retry policies
- Delivery logging
- Tags

Subscribers

- Tie topic to a delivery endpoint
- **Fan out** to subscriber endpoints

Subscribers



Conclusion

- Amazon SNS
- A2P or A2A, Pub/Sub model
- Topic & Subscriber
- FIFO & Standard
- Options

Amazon Simple Queue Service & Demonstration

INE Cloud Series

An INE On-Demand Course

Topics

- Amazon SQS
- Features
- Visibility Timeout
- Demonstration

Amazon Simple Queue Service

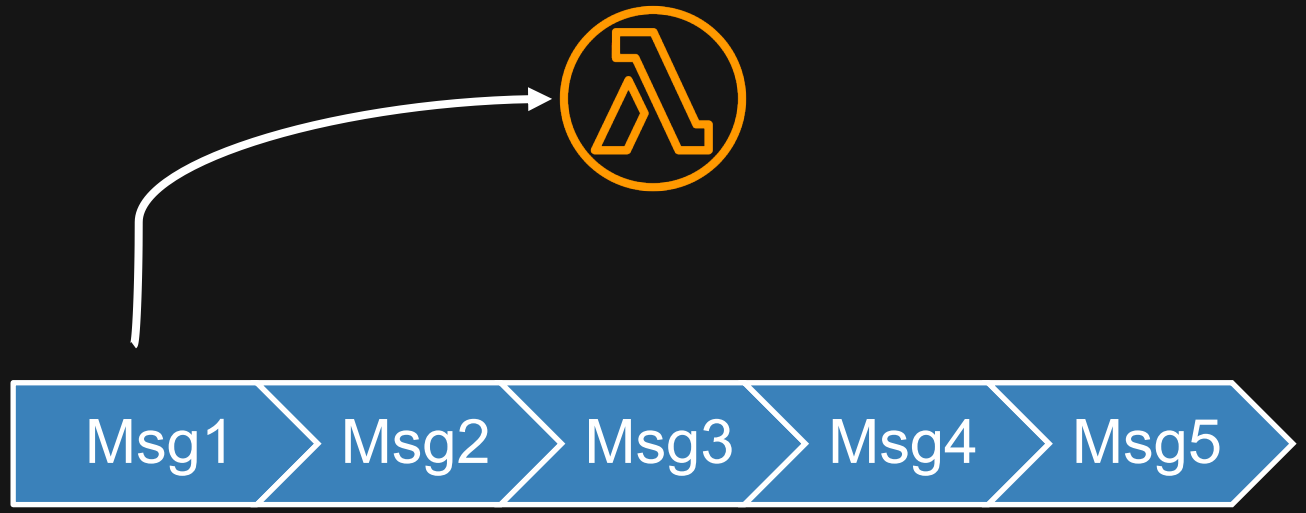
- Decouple infrastructure
- ”Linkage” between processing elements

Amazon Simple Queue Service

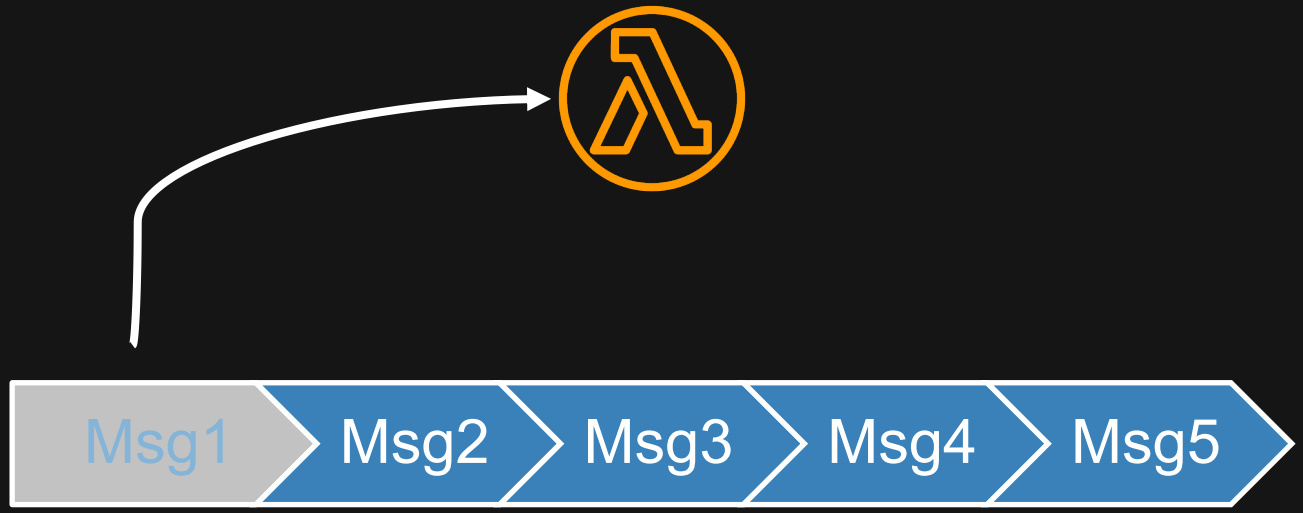
- Managed queue
- Message size up to 256KB
  - Each 64KB is billed as 1 request (***...a 256KB message is billed as 4 requests.)***
- Up to 14-day message retention
- Server-side encryption & dead-letter queue
  - SSE & DLQ
- Message “locking” to stop duplicate processing

SQS “Message locking”

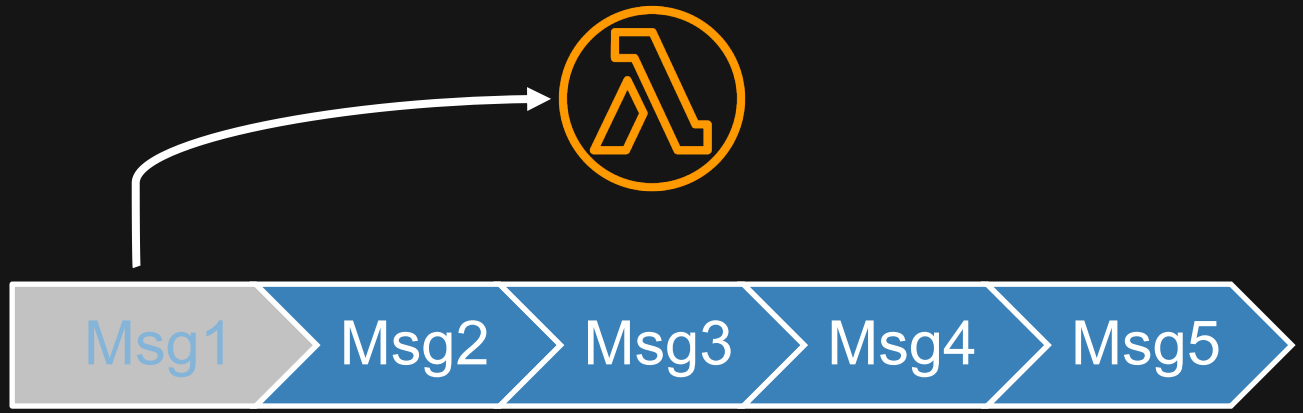
SQS “Message locking”



SQS “Message locking”

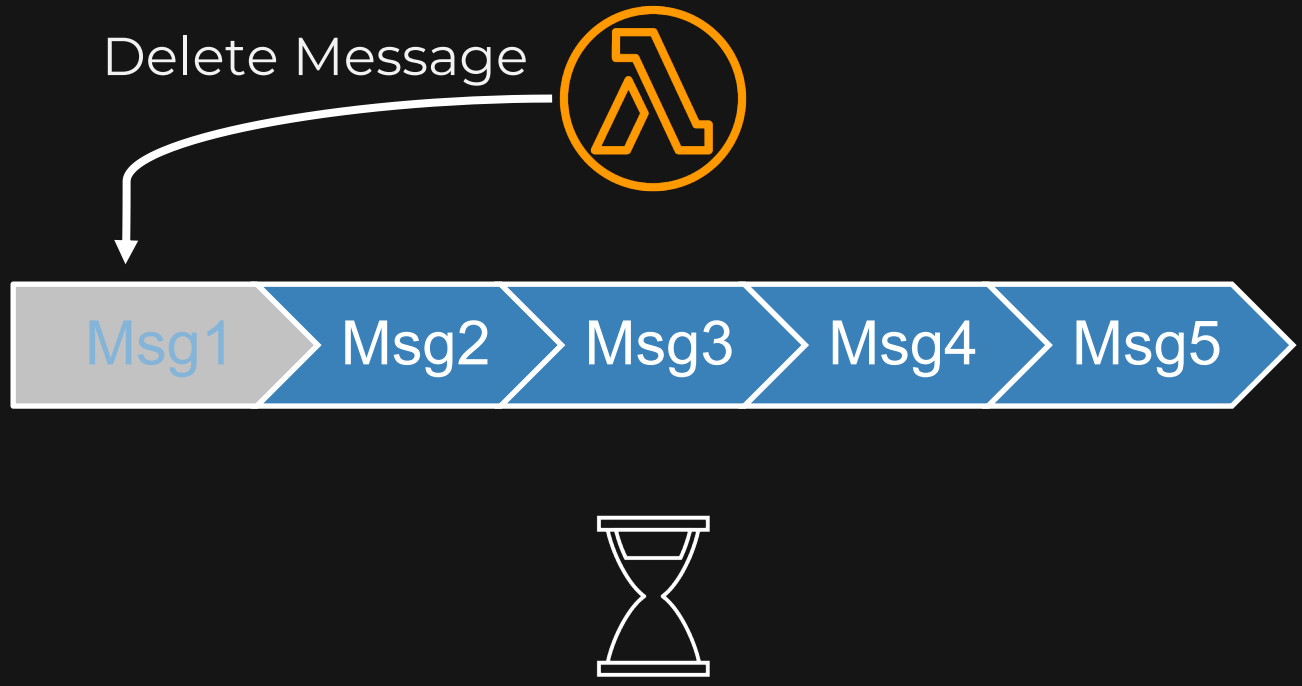


SQS “Message locking”



Visibility Timeout

SQS “Message locking”



SQS “Message locking”

Visibility Timer

SQS Visibility Timeout

Visibility Timer

Maximum time for processing before we retry with another process?

Conclusion

- Amazon SQS
- Managed, decoupling
- FIFO & Standard
- Dead letter queue
- Visibility timeout


Amazon API Gateway
INE Cloud Series
An INE On-Demand Course

Topics

- Why you need an API gateway
- Understanding API
- Amazon API Gateway

The missing link

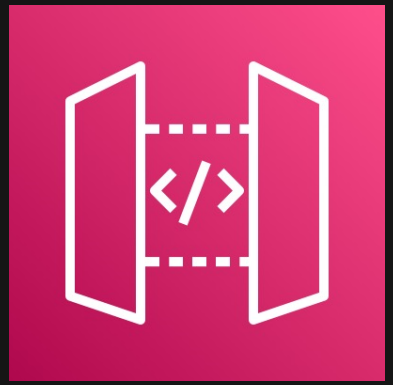
The missing link

The missing link

What is an API then?

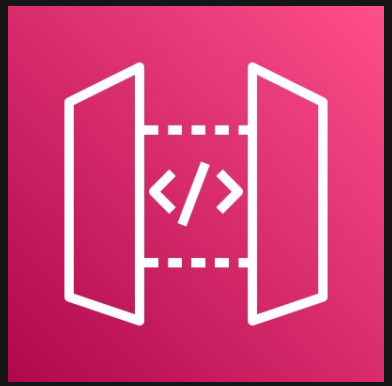
Photo by Tim Douglas : https://www.pexels.com/photo/couple-of-barista-coworkers-standing-in-street-near-cafe-6205486/
Photo by Andrea Piacquadio: https://www.pexels.com/photo/shallow-focus-photo-of-woman-smiling-while-holding-pen-and-paper-3801649
Photo by Ron Lach : https://www.pexels.com/photo/group-of-people-sitting-by-the-table-beside-man-in-black-suit-8921559/
Photo by Timur Saglambilek: https://www.pexels.com/photo/man-wearing-black-apron-near-two-silver-metal-cooking-pot-66639/

Amazon API Gateway



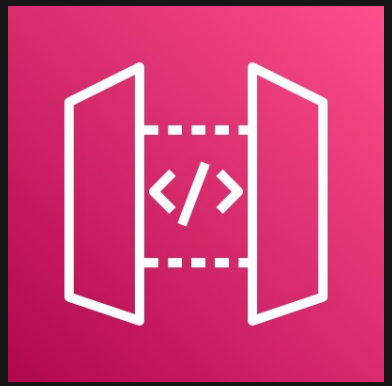
Amazon API Gateway

Amazon API Gateway



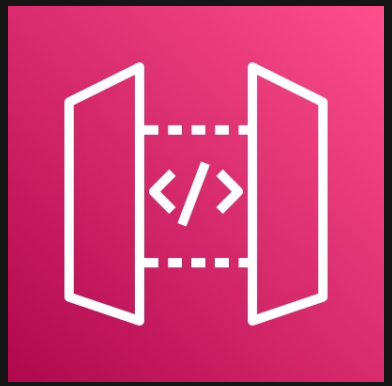
- Managed service
- Endpoint types
  - HTTP API
  - Restful API
  - WebSocket API
    - Real-time, 2-way communications
- VPC Private Endpoints

Amazon API Gateway



- Location:
  - Regional
  - Edge-Optimized
    - Think ***CloudFront***
  - Private API
    - Think ***VPC***

Amazon API Gateway



- Deploy multiple versions of you API
- Performant w/ throttling
- Integrates w/ CloudWatch, IAM, & Amazon Cognito
- Cost: $0.90/million requests

Demonstration

- The PetStore API
  - Understanding standard API
  - Testing
  - Log analysis
- Quick build

Conclusion

- API Gateway
- Service particulars
- Endpoints
- Configuration

Serverless Lab 4: Complete Architecture Build

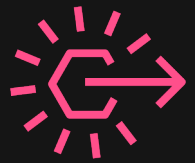
INE Cloud Series

An INE On-Demand Course

LAB 4 – Complete Architecture

Amazon
EventBridge

LAB 4 – Complete Architecture



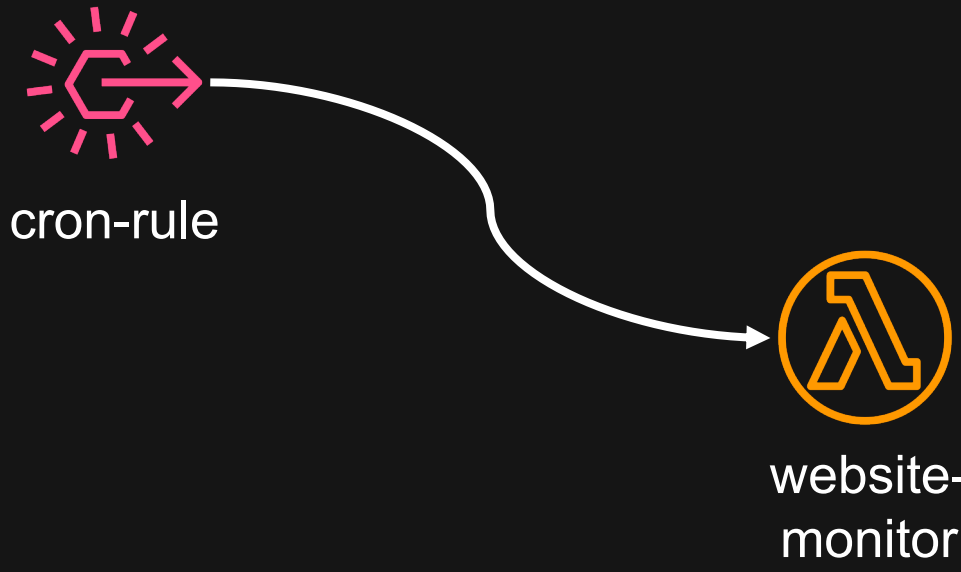
cron-rule

LAB 4 – Complete Architecture

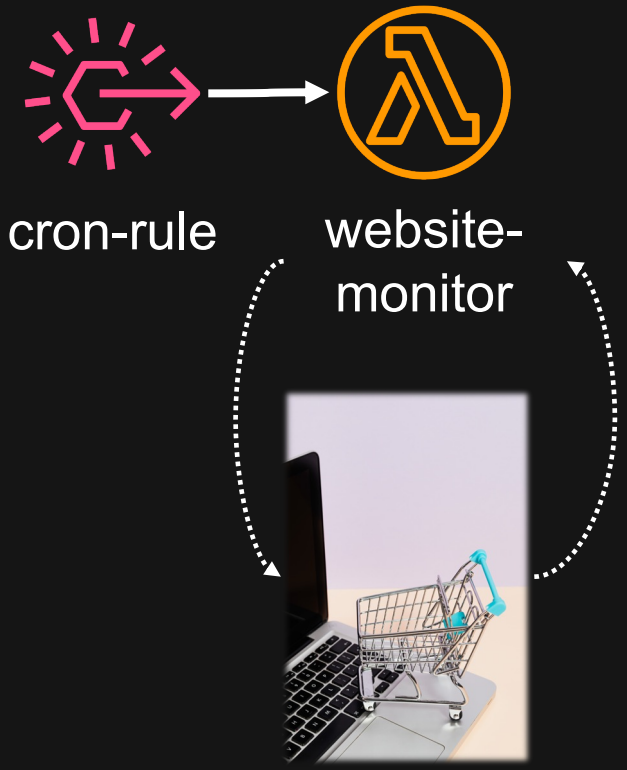
cron-rule

AWS Lambda

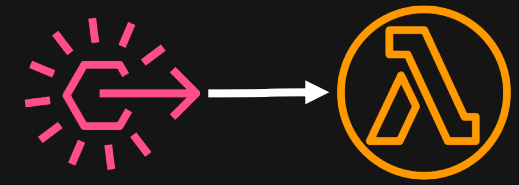
LAB 4 – Complete Architecture



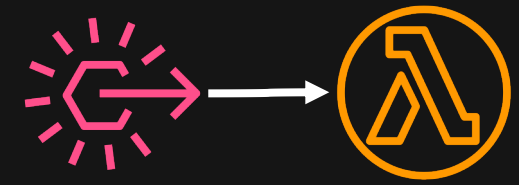
LAB 4 – Complete Architecture



LAB 4 – Complete Architecture



LAB 4 – Complete Architecture



cron-rule

website-monitor

Amazon Simple Notification Service (Amazon SNS)

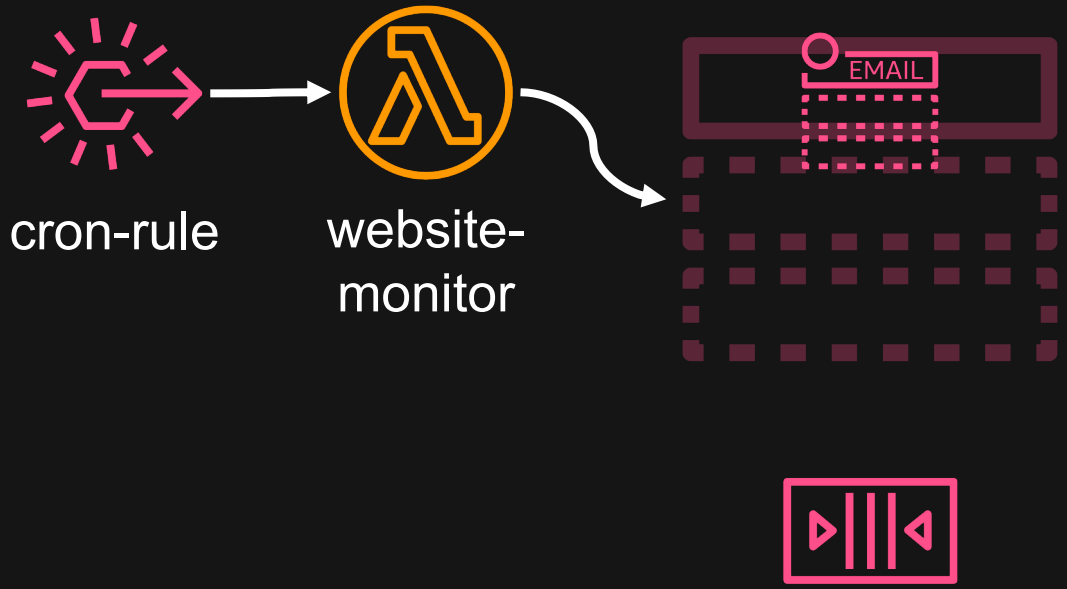
LAB 4 – Complete Architecture

LAB 4 – Complete Architecture

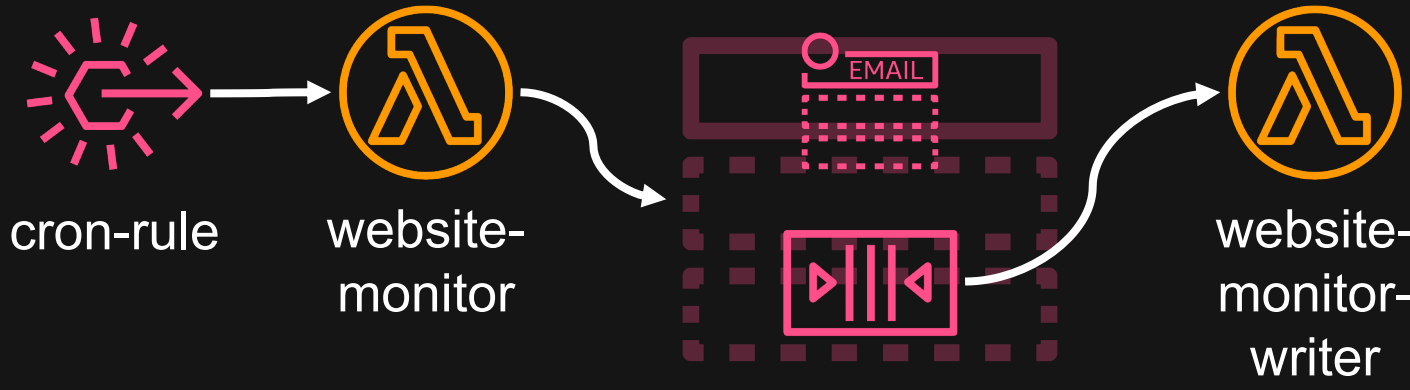
LAB 4 – Complete Architecture

Amazon Simple Queue Service (Amazon SQS)

LAB 4 – Complete Architecture

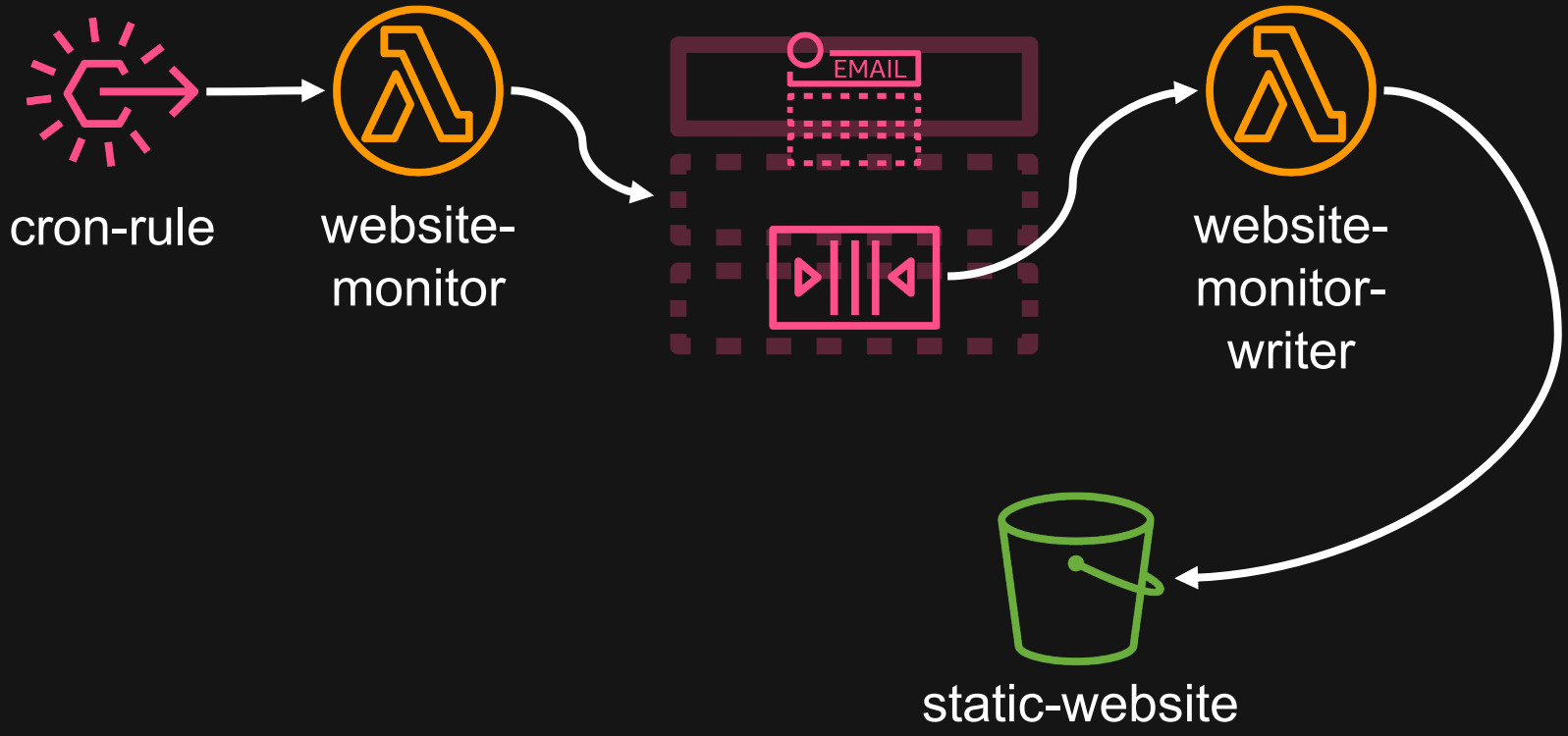


LAB 4 – Complete Architecture

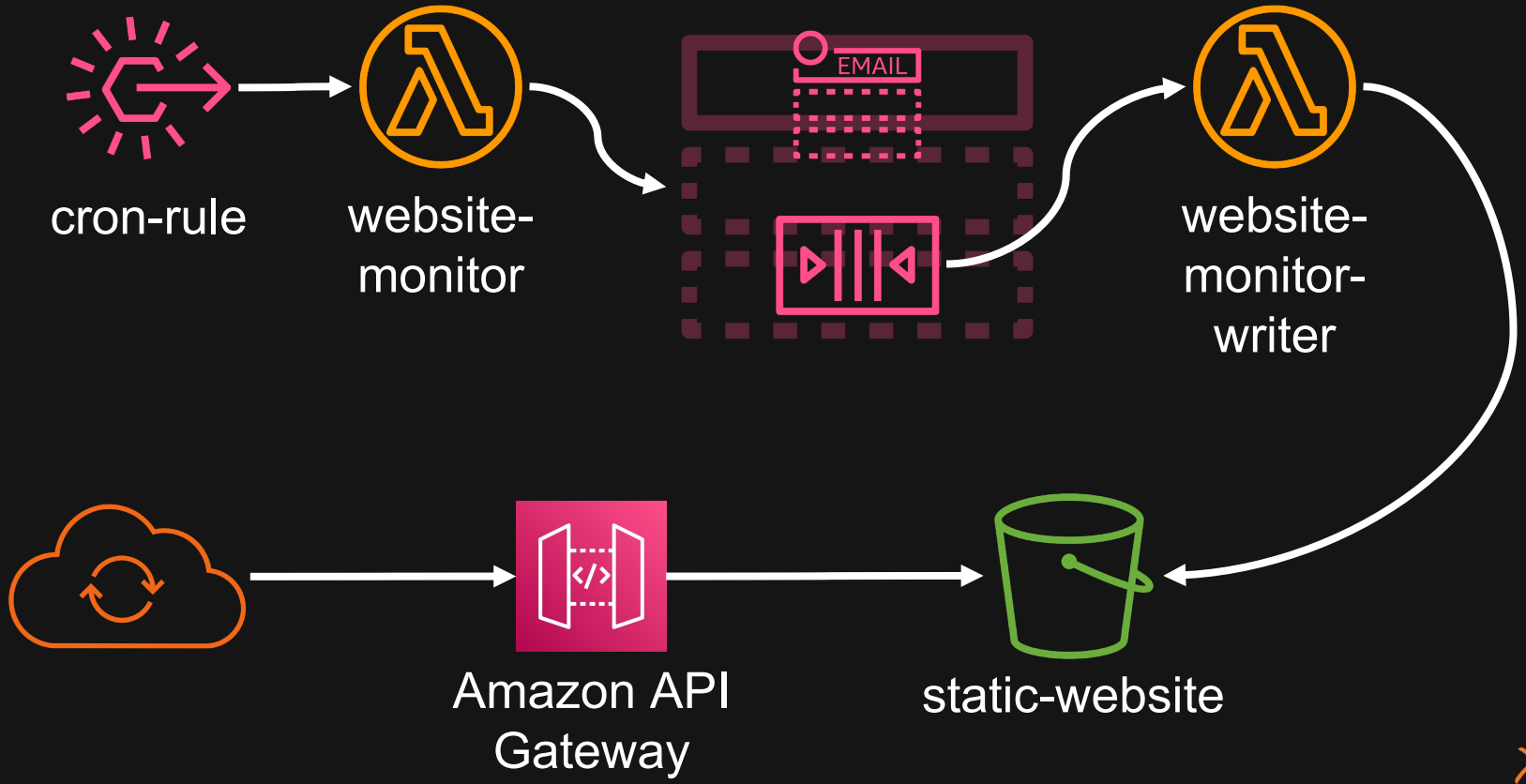


LAB 4 – Complete Architecture

LAB 4 – Complete Architecture



LAB 4 – Complete Architecture



Lab 4: Serverless Capstone Project

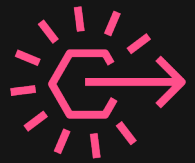
INE Cloud Series

An INE On-Demand Course

LAB 4 – Complete Architecture

Amazon
EventBridge

LAB 4 – Complete Architecture



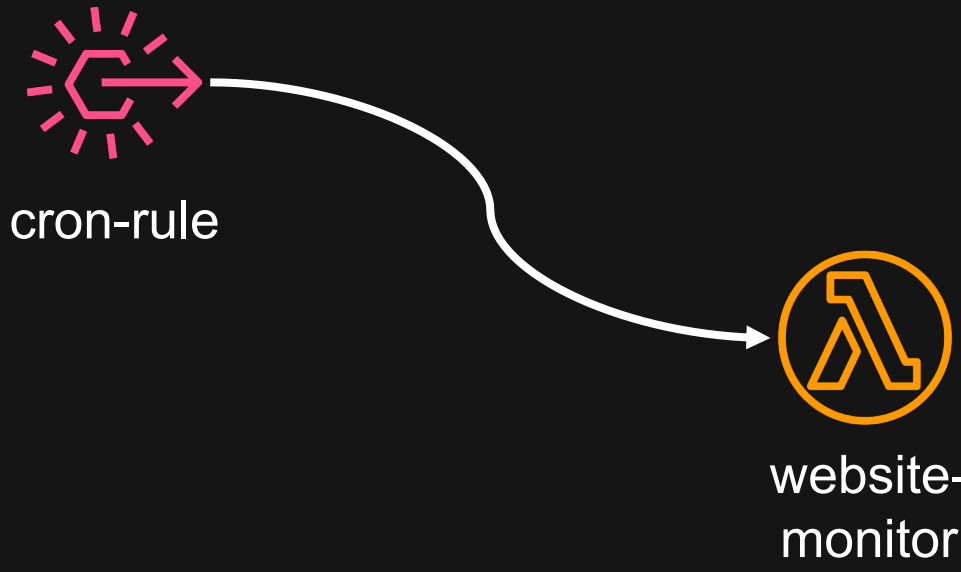
cron-rule

LAB 4 – Complete Architecture

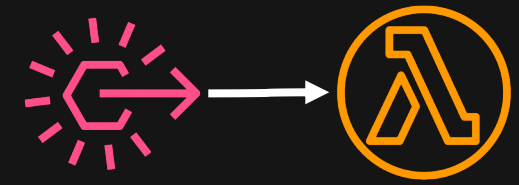
cron-rule

AWS Lambda

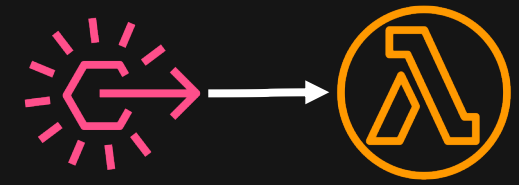
LAB 4 – Complete Architecture



LAB 4 – Complete Architecture



LAB 4 – Complete Architecture



cron-rule

website-monitor

Amazon Simple Notification Service (Amazon SNS)

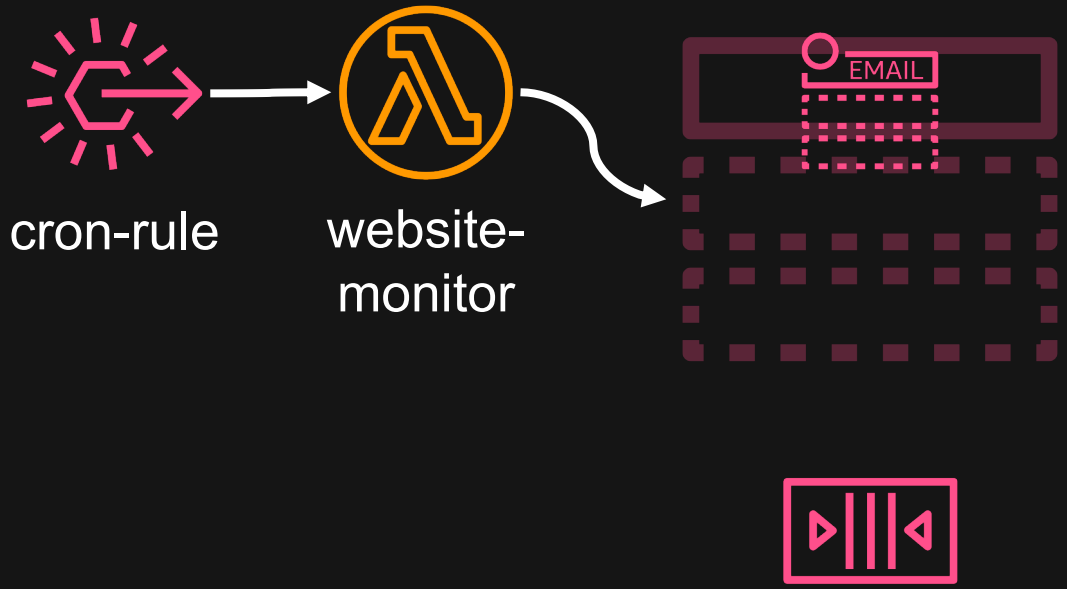
LAB 4 – Complete Architecture

LAB 4 – Complete Architecture

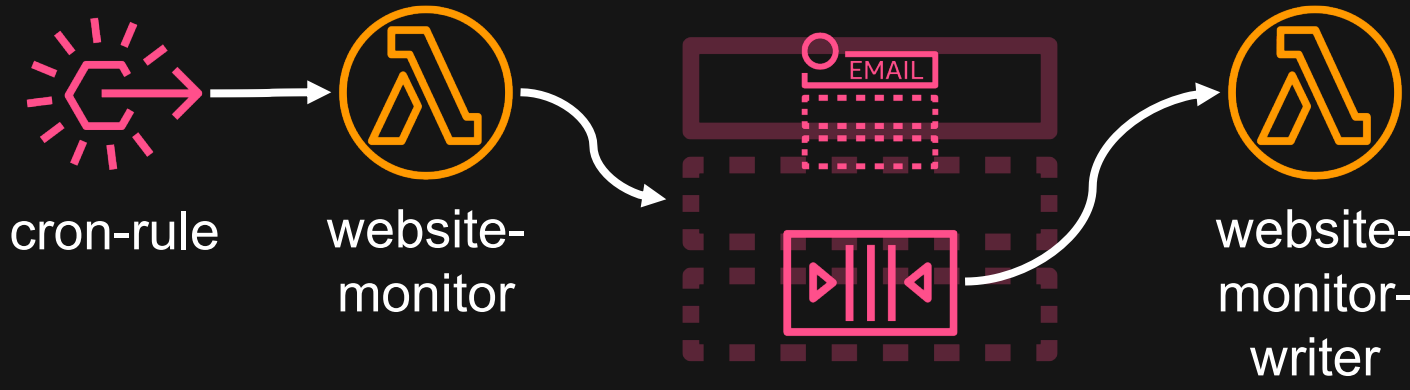
LAB 4 – Complete Architecture

Amazon Simple Queue Service (Amazon SQS)

LAB 4 – Complete Architecture

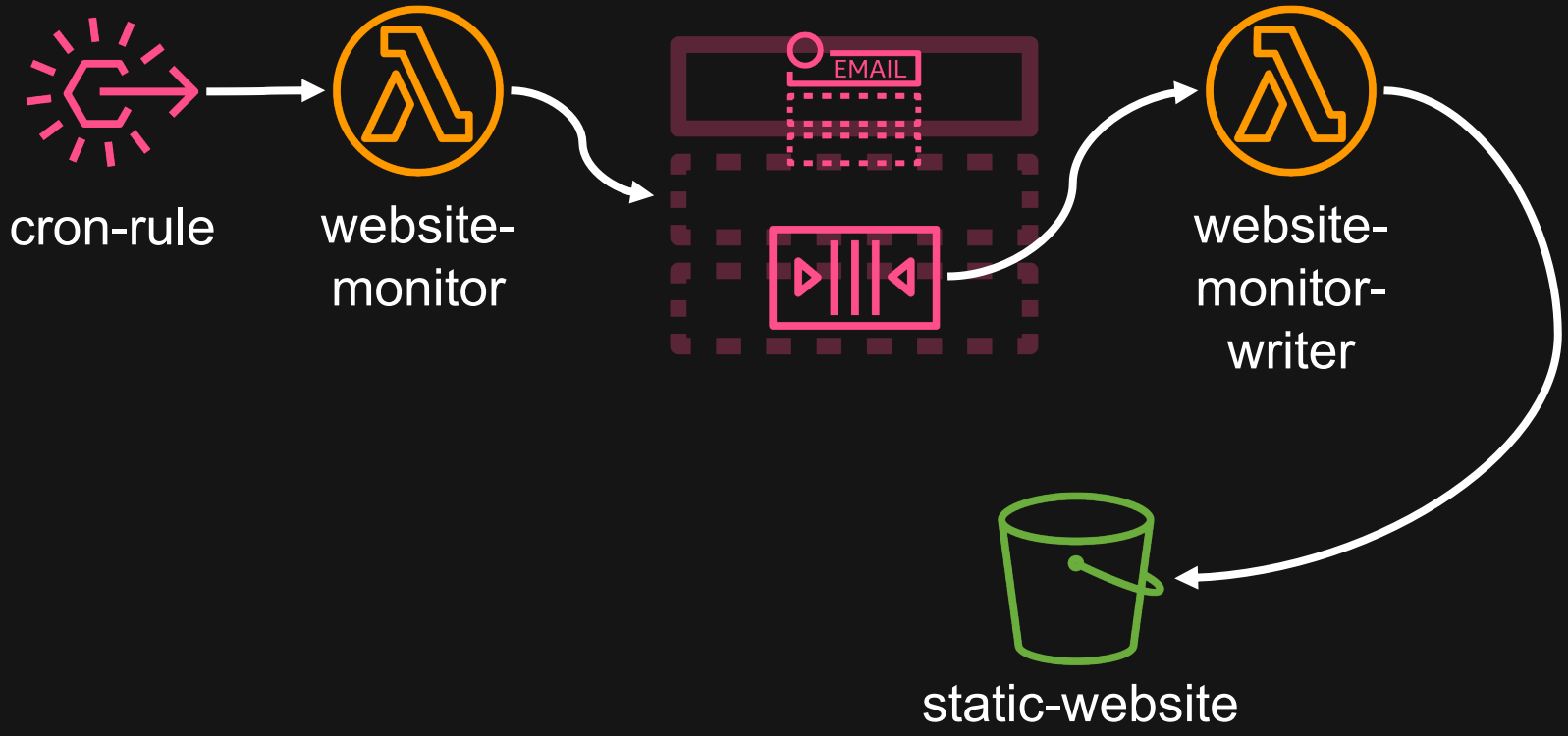


LAB 4 – Complete Architecture

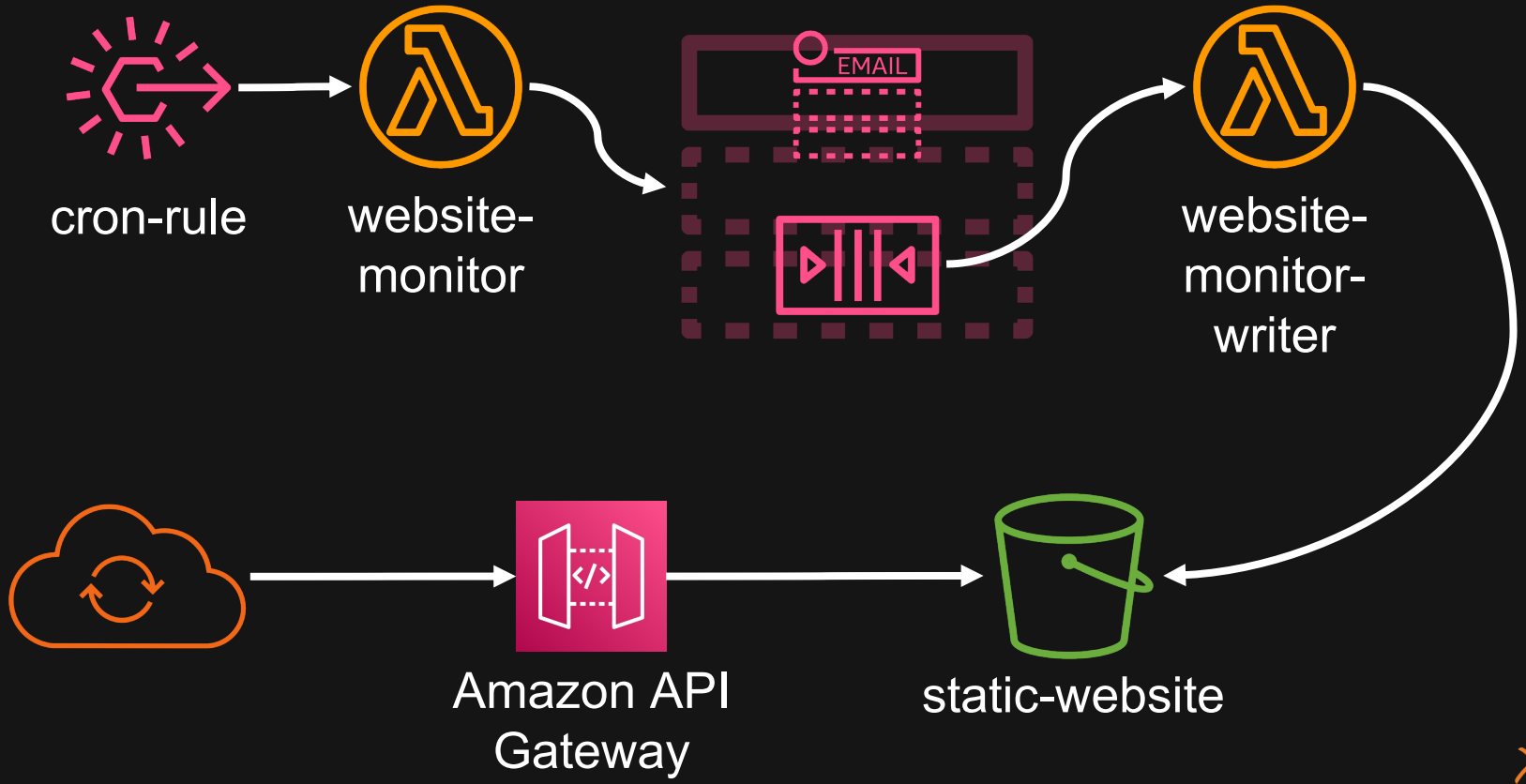


LAB 4 – Complete Architecture

LAB 4 – Complete Architecture



LAB 4 – Complete Architecture



Lab 4: Serverless Capstone Project
Part 8: Code Review

INE Cloud Series

An INE On-Demand Course