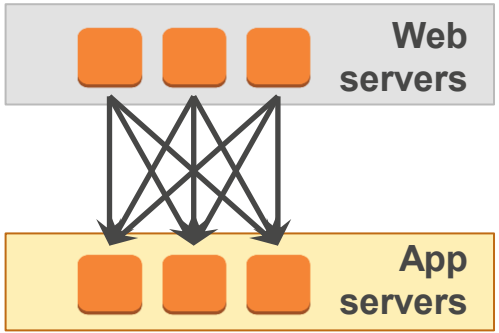
Best Practice: Loosely Couple Your Components

Design architectures with independent components.

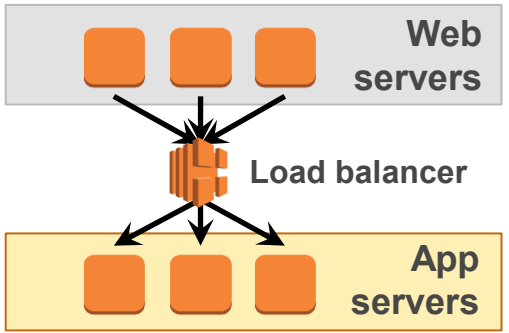
Reduce interdependencies so that the change or failure of one component does not affect other components.

Anti-pattern



Web servers **tightly coupled** to app servers

Best practice

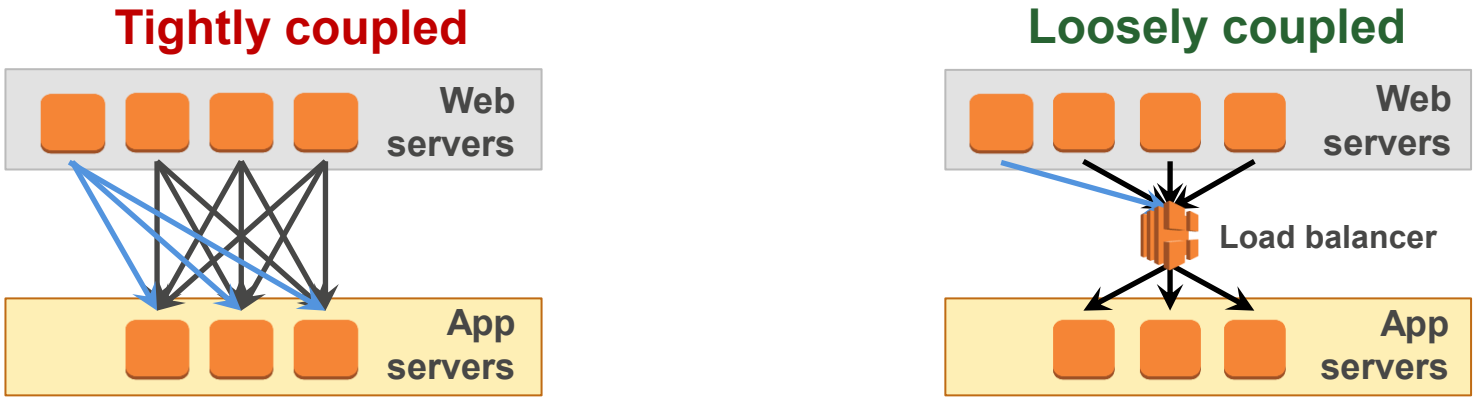


Decoupled with a load balancer

Decoupling

The more loosely your system is coupled:

- The more easily it scales.



Decoupling

The more loosely your system is coupled:

- The more easily it scales.
- The more fault-tolerant it can be.

Loose Coupling Strategies

Best Practice: Design Services, Not Servers

Leverage the breadth of AWS services; don't limit your infrastructure to servers.

Managed services and serverless architectures can provide greater reliability and efficiency in your environment.

Anti-pattern

- Simple applications run on persistent servers.
- Applications communicate directly with one another.
- Static web assets are stored locally on instances.
- Back-end servers handle user authentication and user state storage.

Best practice

- Serverless solution is provisioned at time of need.
- Message queues handle communication between applications.
- Static web assets are stored externally, such as on Amazon S3.
- User authentication and user state storage are handled by managed AWS services.

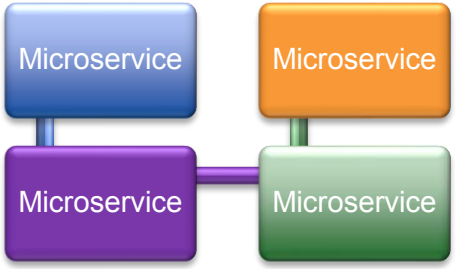
Implement A Service-Oriented Architecture (SOA)

Service-Oriented Architecture:

An architectural approach in which application components provide **services** to other components via a communications protocol.

Services are self-contained units of functionality.

Microservices Architectures And Decoupling



Microservices:

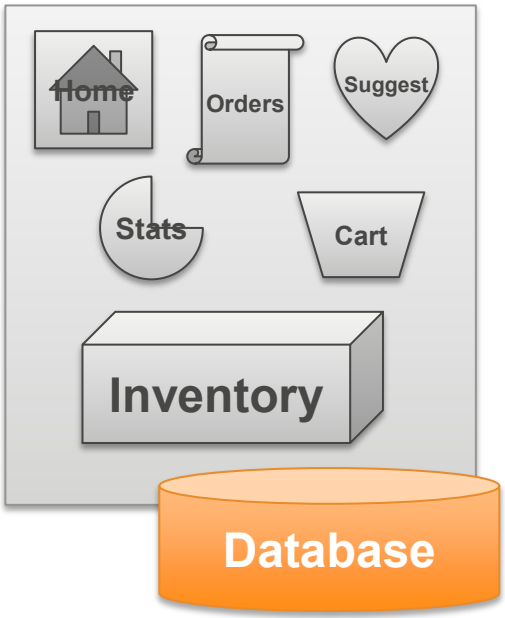
Small, independent processes within an SOA.

Each process is focused on doing one small task.

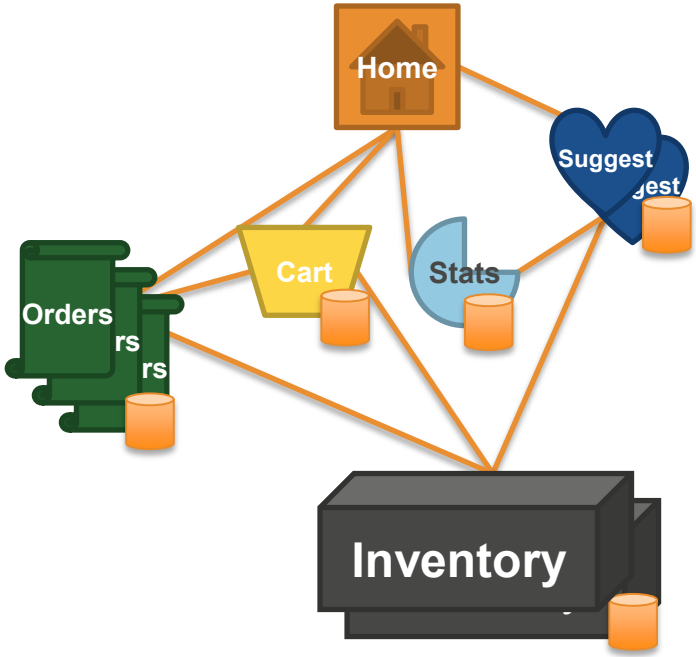
Processes communicate to each other using language-agnostic APIs.

Comparing Architectural Styles

Traditional app architectures are monolithic:

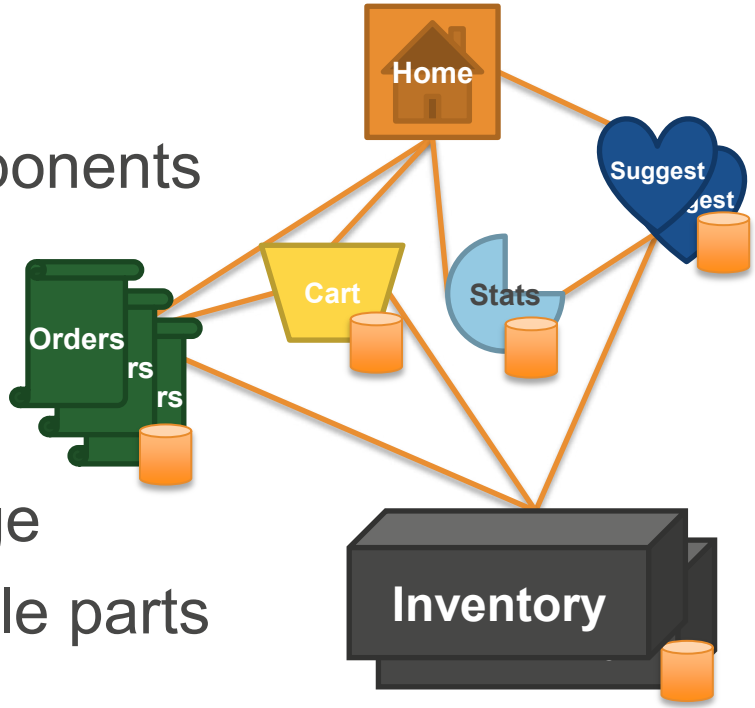


Microservice-based architectures are loosely coupled:



Microservices

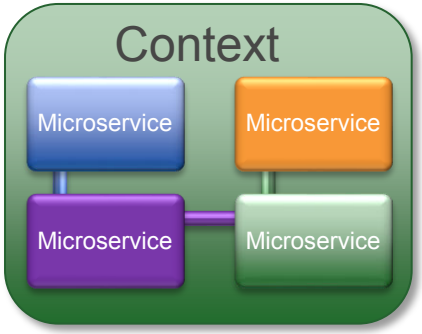
- Split features into individual components
- Have smaller parts to iterate on
- Have a reduced test surface area
- Benefit from a lower risk of change
- Use individual horizontally scalable parts



Microservices Concepts: Bounding

Each business domain can be divided into contexts.

- Contexts are made up of microservices.
- Each context has its own functions, objects, and language.
- Contexts can comprise single operations like:
  - Transaction handling
  - User registration, authentication, tracking
  - Content publishing or syndication
  - Catalog, warehouse management



Best Practices (1/2)

Change components without breaking them:

- The interface is a contract.
- Modifying capabilities should not affect consumers.

Use a simple API:

- Lowers the cost of using your service.
- More complexity means more resistance to change.
- The less you share, the less will break.
- Allows you to hide the details.

Best Practices (2/2)

Keep it technology-agnostic:

- Enable change: be ready to embrace the next evolution
- Be tech-omnivorous

Design with failure in mind:

- “Everything fails, all the time.”

Monitor your environment:

- Not just the infrastructure
- Extracting health status means collecting it from services

What can be used to easily and reliably communicate between components?

Amazon Simple Queue Service (SQS)

Amazon Simple Queue Service (SQS)

Amazon SQS is a fully managed message queueing service. Transmit any volume of messages at any level of throughput without losing messages or requiring other services to be always available.

Messages

- Generated by one component to be consumed by another.
- Can contain 256 KB of text in any format.

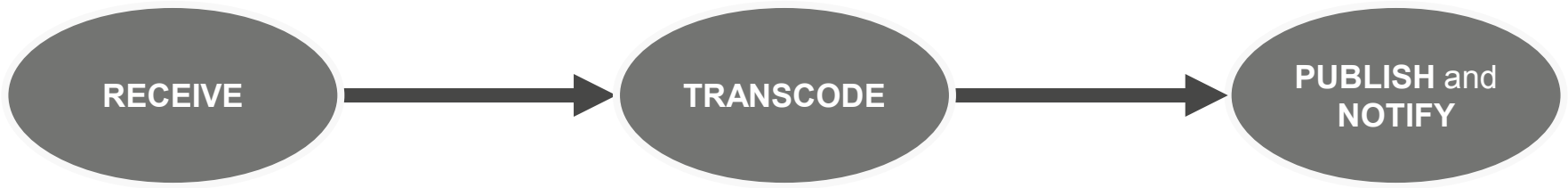
Amazon SQS

- Ensures delivery of each message at least once.
- Supports multiple readers and writers on the same queue.
- Does not guarantee first in, first out.

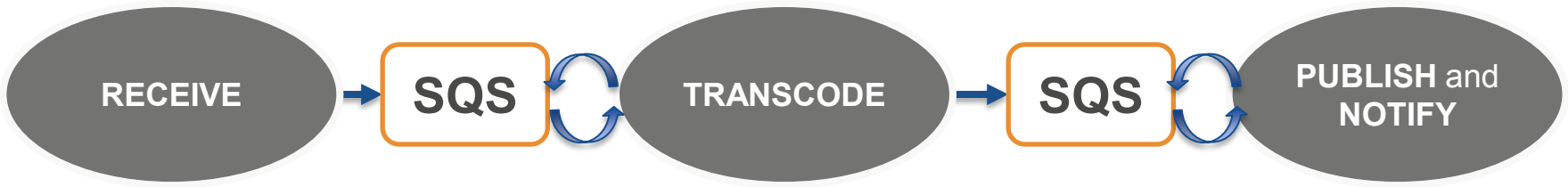
Queues

- Repository for messages awaiting processing.
- Acts as a buffer between the components which produce and receive data.

Tightly Coupled Systems

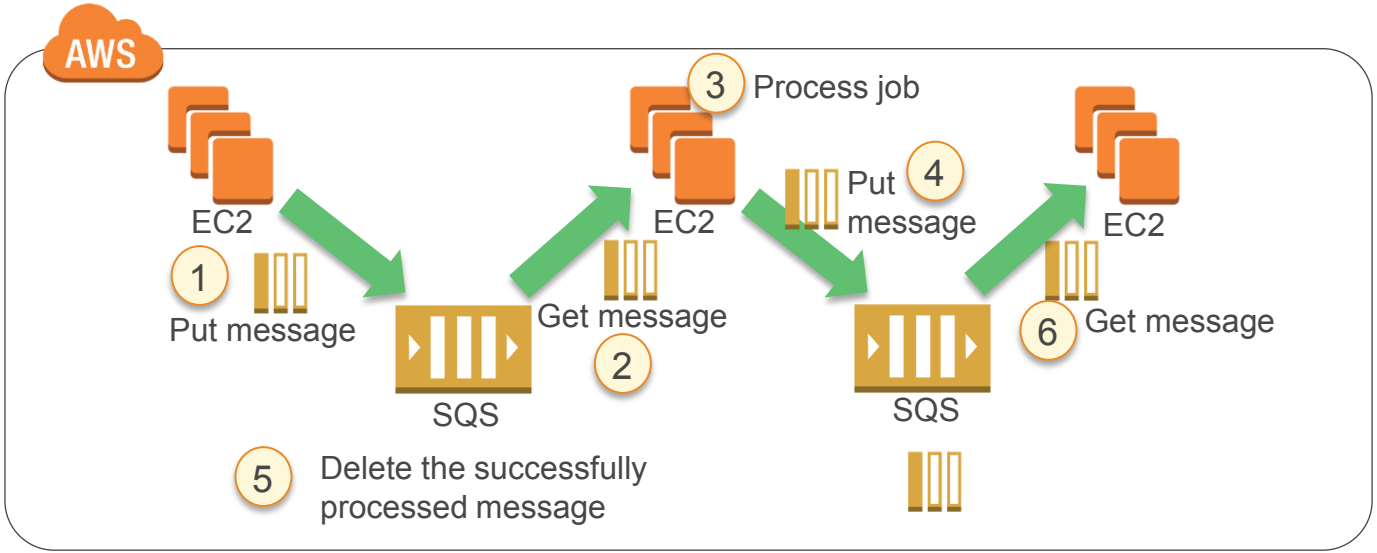


Loosely Coupled Systems



Loose Coupling With Amazon SQS

The queuing chain pattern enables asynchronous processing:



Why Choose Amazon Simple Queue Service (SQS)?

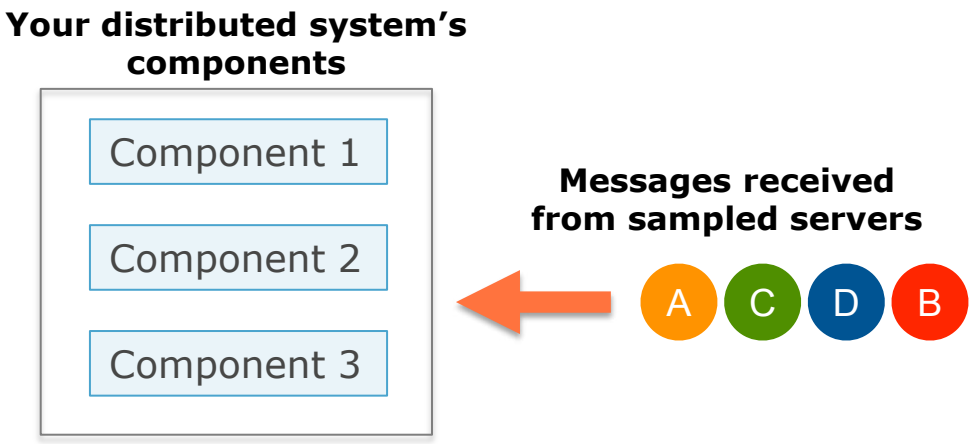
- Extremely scalable
  - Potentially millions of messages.
- Extremely reliable
  - All messages are stored redundantly on multiple servers and in multiple data centers.
- Simple to use
  - Messages get sent in, messages get pulled out.
- Simultaneous read/write
- Secure
  - API credentials are needed.

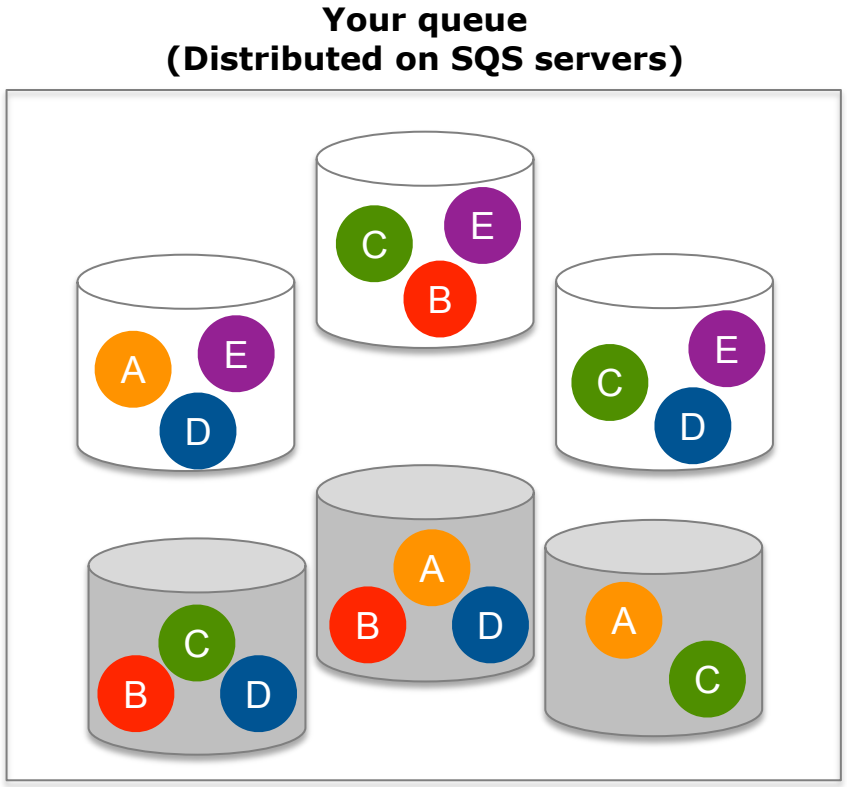
Understand The Properties Of Distributed Queues

Message Order

At-Least-Once Delivery

Message Sample





Amazon SQS Key Feature: Visibility Timeout

Visibility timeout prevents multiple components from processing the same message.

- When a message is received, it becomes “locked” while being processed. This prevents it from being processed by other computers.
- The component that receives the message processes it and then deletes it from the queue.
- If the message processing fails, the lock will expire and the message will be available again (fault tolerance).

Amazon SQS Key Feature: Dead Letter Queues

A dead letter queue is a queue of messages that **could not be processed**.

Why use a dead letter queue?

- It can sideline and isolate the unsuccessfully processed messages.

Note: A dead letter queue must reside in the same account and AWS region as the other queues that use the dead letter queue.

Resource-based Permissions: Sharing A Queue

Shared queues

- Queues can be shared with other AWS accounts and anonymously.
- A **permission** gives access to another person to use your queue in some particular way.
- A **policy** is the actual document that contains the permissions you granted.

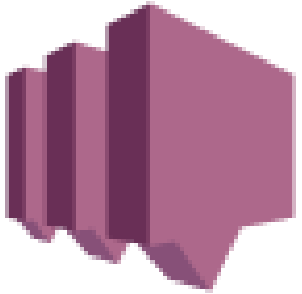
Who pays for shared queue access?

- The queue **owner** pays for shared queue access.

Amazon SQS Use Cases

- Work queues
- Buffering batch operations
- Request offloading
- Fan-out
- Auto Scaling

Other decoupling services on AWS:



Amazon Simple Notification Service (SNS)

Amazon DynamoDB

Amazon API Gateway

AWS Lambda

Amazon Simple Notification Service (SNS)

Amazon SNS enables you to set up, operate, and send notifications to subscribing services other applications.

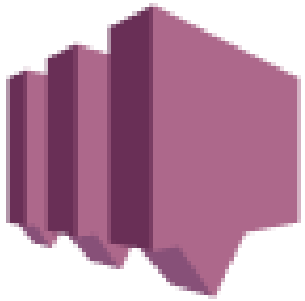
- Messages published to topic.
- Topic subscribers receive message.

Subscriber types:

- Email (plain or JSON)
- HTTP/HTTPS
- Short Message Service (SMS) clients (USA only)
- Amazon SQS queues
- Mobile push messaging
- AWS Lambda Function

Characteristics Of Amazon SNS

- Single published message
- Order is not guaranteed
- No recall
- HTTP/HTTPS retry
- 256 KB max per message

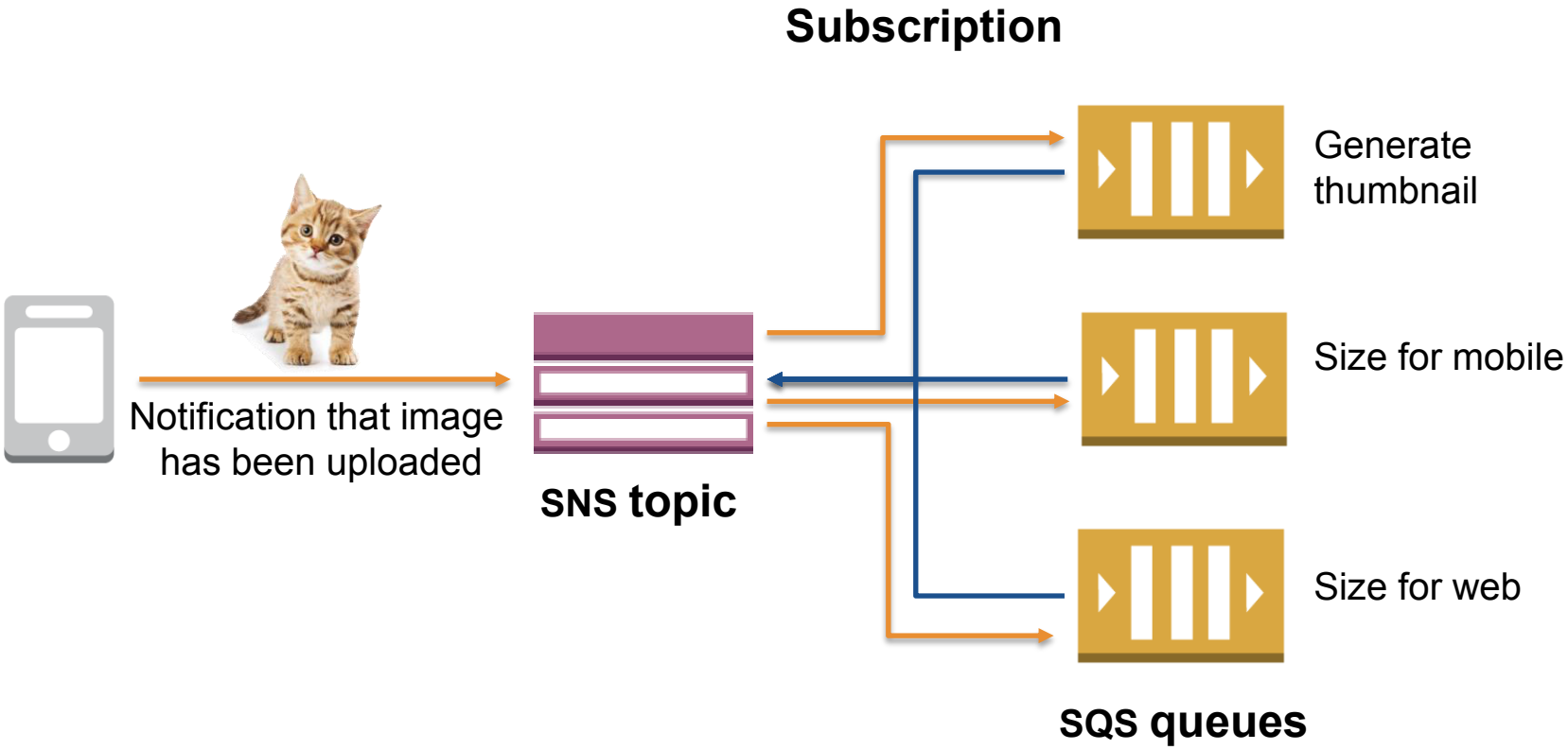


How Is Amazon SNS Different From Amazon SQS?

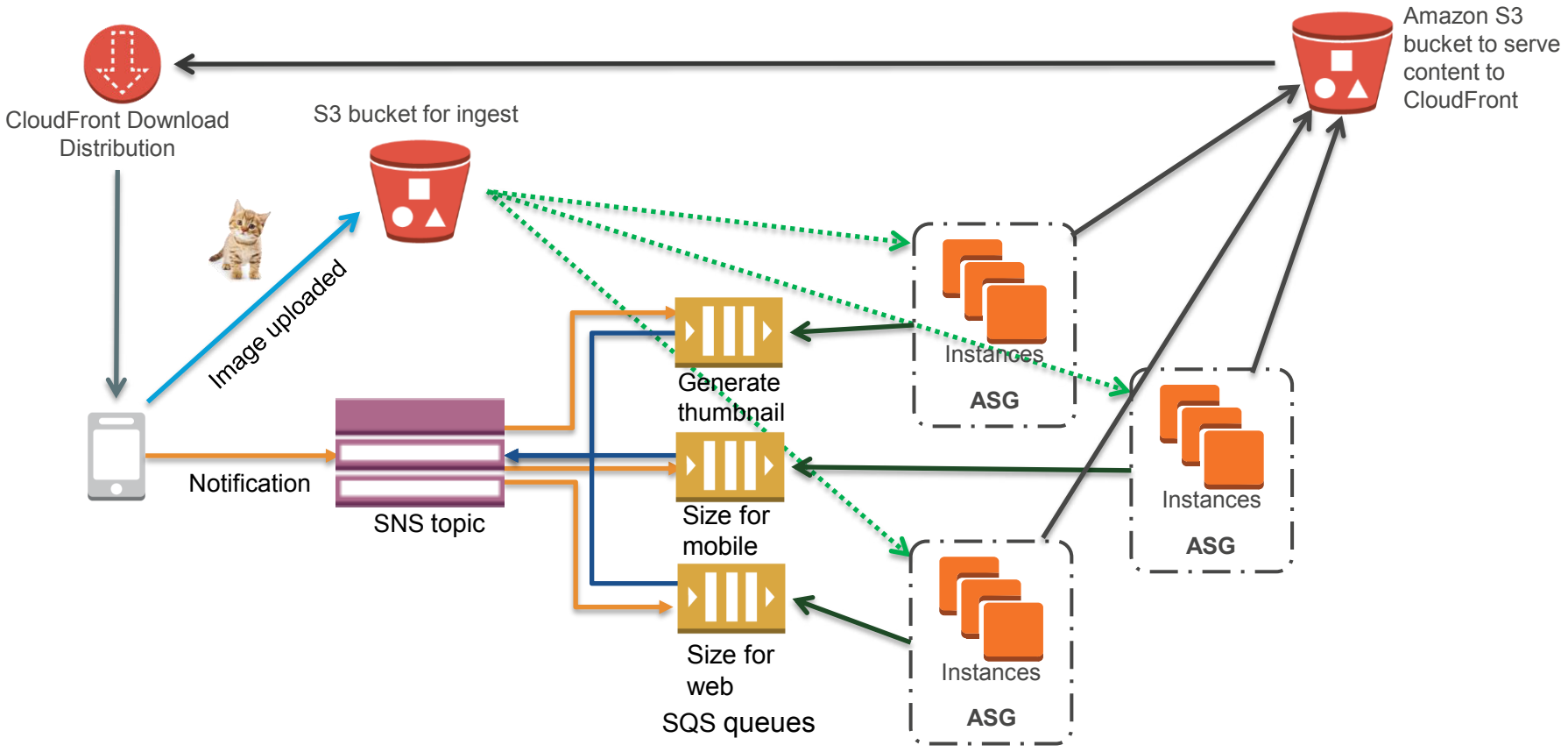
Amazon SQS and Amazon SNS are both messaging services within AWS.

	Amazon SNS	Amazon SQS
Message persistence	No	Yes
Delivery mechanism	Push (Passive)	Poll (Active)
Producer/consumer	Publish/subscribe	Send/receive

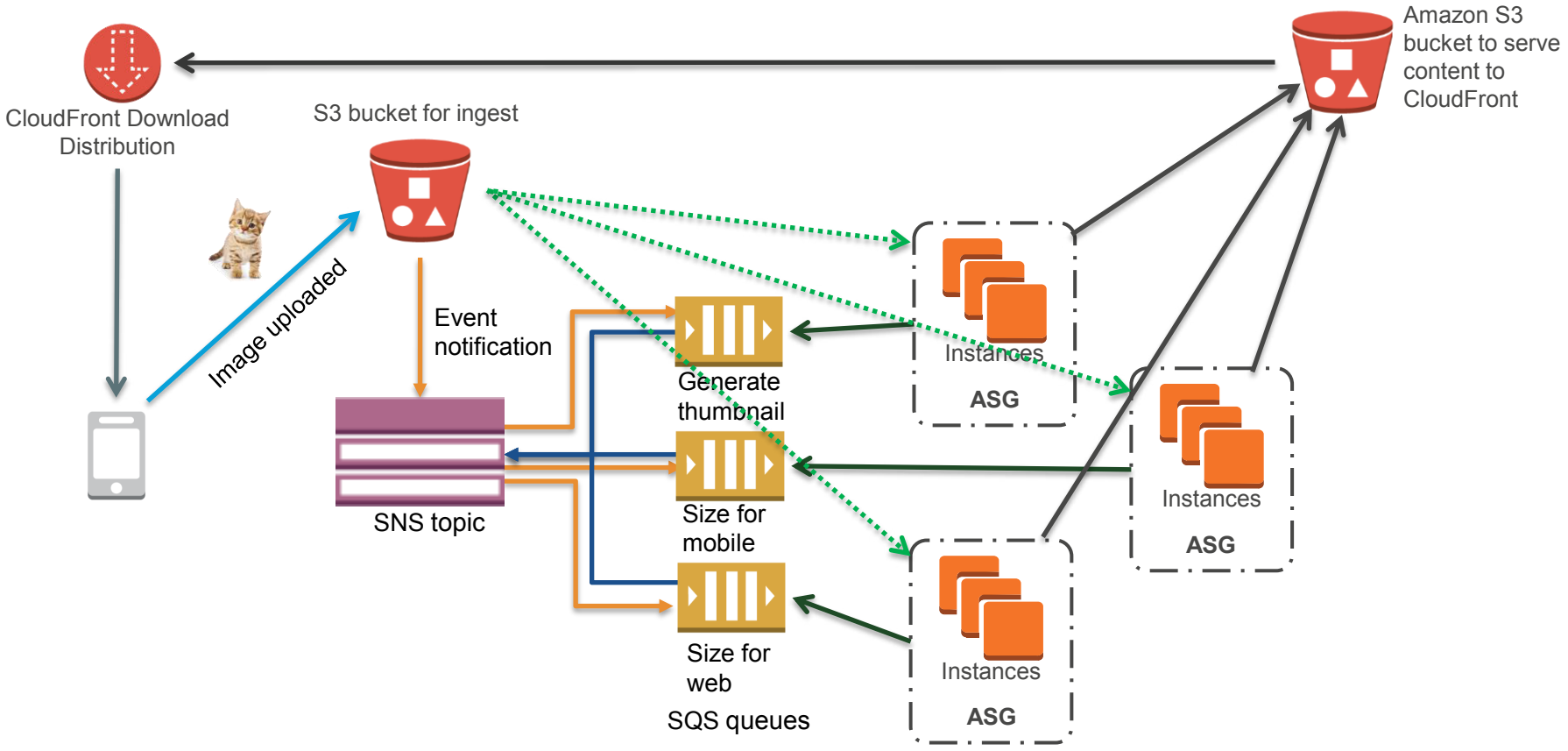
Use Case: Fan-Out



End-to-End Scenario: Image Processing



End-to-End Scenario: Image Processing – S3 Event Notifications



Loose Coupling and Amazon DynamoDB

Use DynamoDB With Loosely Coupled Infrastructure

DynamoDB is a great solution for storing and retrieving your processing output with high throughput.

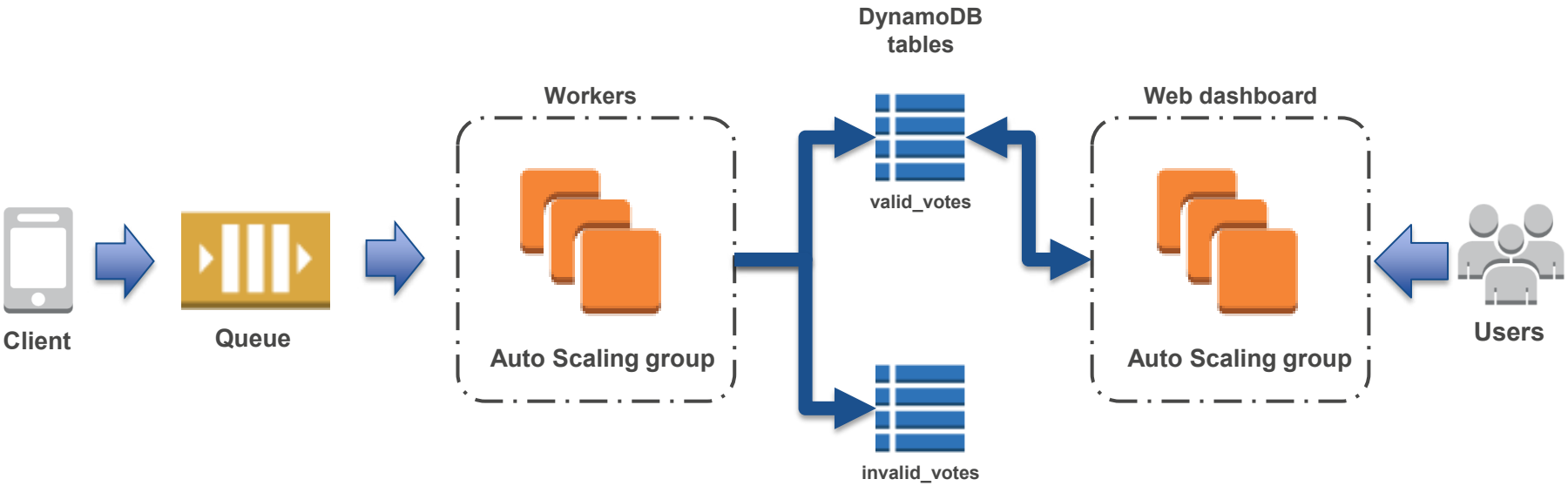
DynamoDB is:

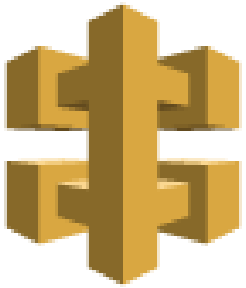
- Highly available
- Fault-tolerant
- Fully managed

Loosely coupled systems can work very well with a managed NoSQL database solution like DynamoDB.

Example Pattern With Amazon SQS And DynamoDB

An example mobile voting platform:





Amazon API Gateway

Amazon API Gateway

Allows you to create APIs that act as "front doors" for your applications to access data, business logic, or functionality from your back-end services.

Fully managed and handles all tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls.

Can handle workloads running on:

- Amazon EC2
- AWS Lambda
- Any web application

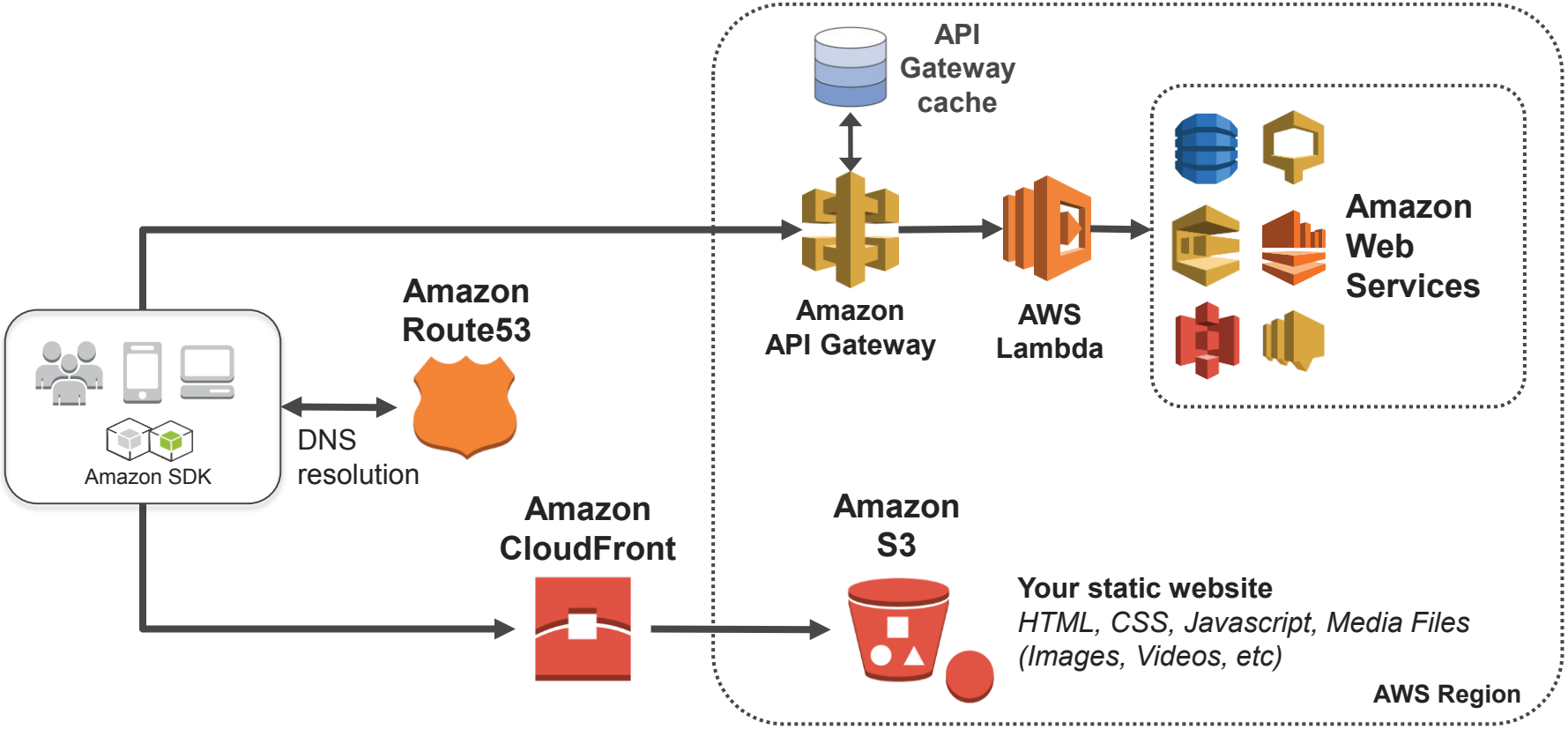
Features Of Amazon API Gateway

- Host and use multiple versions and stages of your APIs.
- Create and distribute API keys to developers.
- Leverage signature version 4 to authorize access to APIs.
- Throttle and monitor requests to protect your back end.
- Deeply integrated with AWS Lambda.

Benefits Of Amazon API Gateway

- Managed cache to store API responses
- Reduced latency and Distributed Denial of Service (DDoS) protection through Amazon CloudFront
- SDK generation for iOS, Android, and JavaScript
- OpenAPI Specification (Swagger) support
- Request/response data transformation

Serverless Architecture Using API Gateway



Serverless architectures: Do you really need all of your instances?

Use AWS Lambda To Decouple Your Infrastructure

AWS Lambda is a great solution for processing data with high availability and a limited cost footprint.

AWS Lambda allows you to further decouple your infrastructure by replacing traditional servers with simple microprocesses.

Serverless Computing With AWS Lambda

AWS Lambda starts code within milliseconds of an event such as:

- An image upload
- In-app activity
- A website click
- Output from a connected device

Consider AWS Lambda if:

- You're using entire instances to run simple functions or processing applications.
- You don’t want to worry about HA, scaling, deployment, or management.

Triggers For AWS Lambda Functions

Amazon DynamoDB

Amazon S3

Amazon SNS

Amazon Kinesis

Amazon SES

Scheduled Events

Amazon Cognito

AWS SDKs via Amazon API Gateway

Amazon CloudWatch

Amazon Echo: Alexa Skills

HTTPS via API Gateway

How To Use AWS Lambda

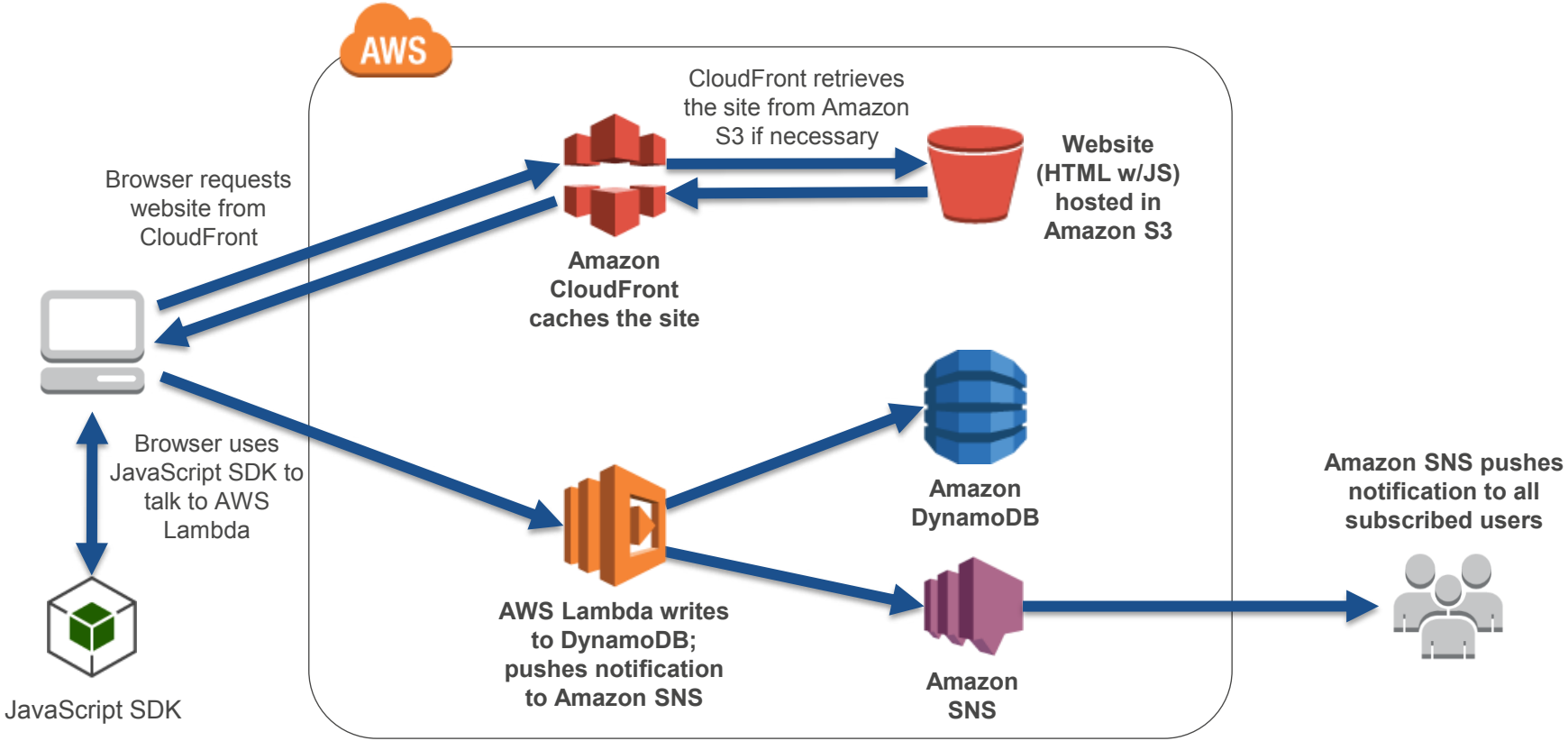
1. Upload your code to AWS Lambda (in .zip form)
   - You can also write your code directly into an editor in the console or import it from an Amazon S3 bucket.
2. Scheduled function? Specify how often it will run.
   Event-driven function? Identify the event source.
3. Specify its necessary compute resources (128MB-1.5GB of memory).
4. Specify its timeout period.
5. Specify the VPC whose resources it needs to access (if applicable).
6. Launch the function.

That's it. From there, AWS deploys and manages it for you.

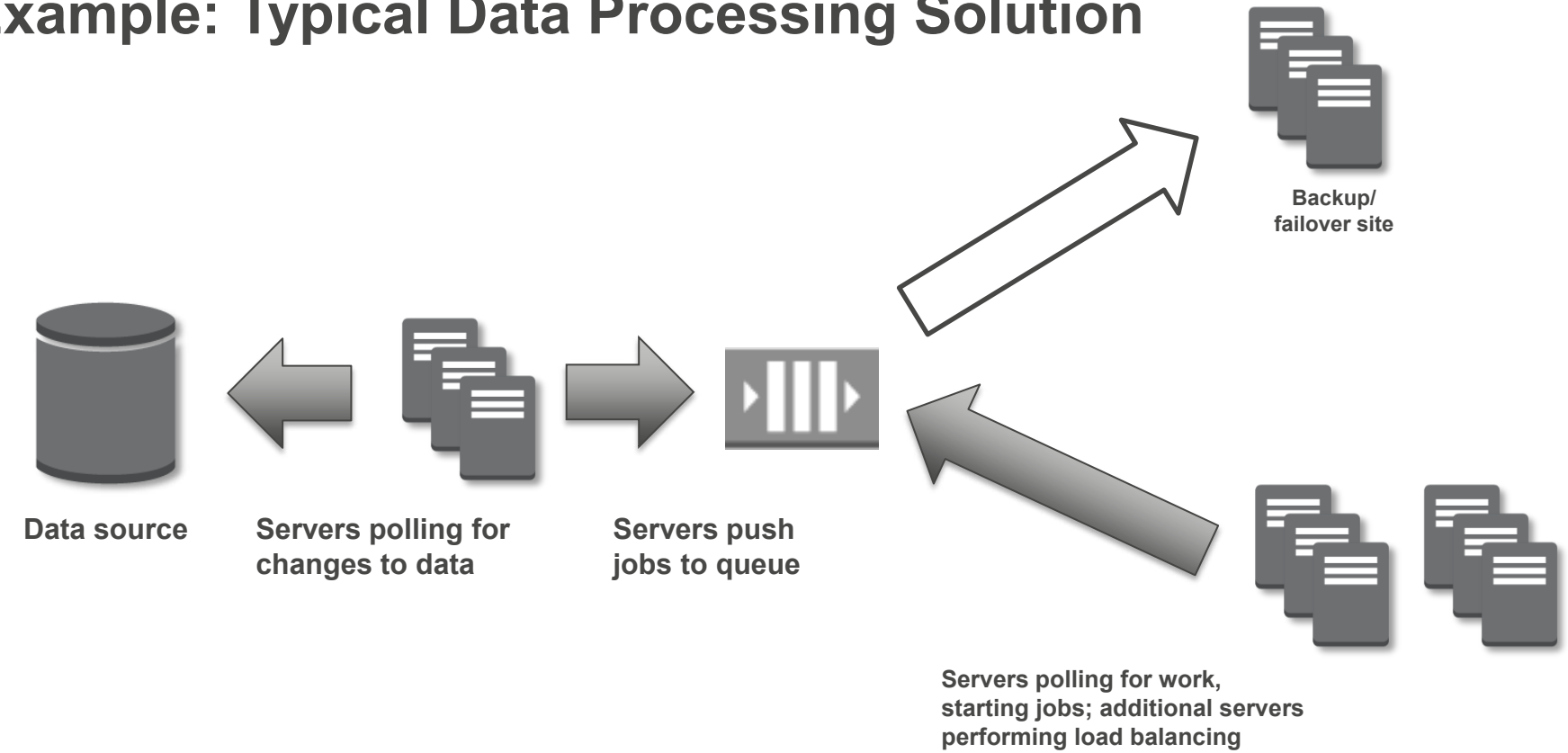
Resource Sizing With AWS Lambda

- AWS Lambda currently offers 23 different levels of resource allocation, which range from:
  - 128 MB of memory and the lowest CPU power, to
  - 1.5 GB of memory and the highest CPU power
- More resources = lower latency for CPU-intensive workloads.
- Compute price scales with resource level.
- Functions can run between 100 ms and five minutes in length.
- Free tier: 1M free requests and 400,000 GB-s/mo of compute time.

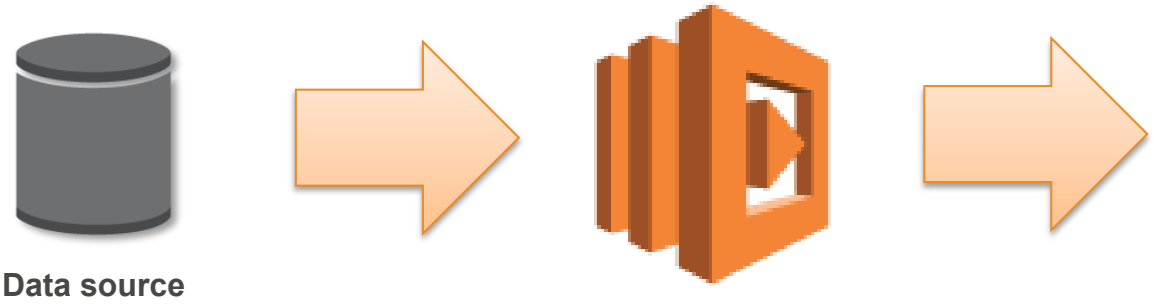
Example: Using AWS Lambda As A Web Server



Example: Typical Data Processing Solution



Example: Data Processing With AWS Lambda



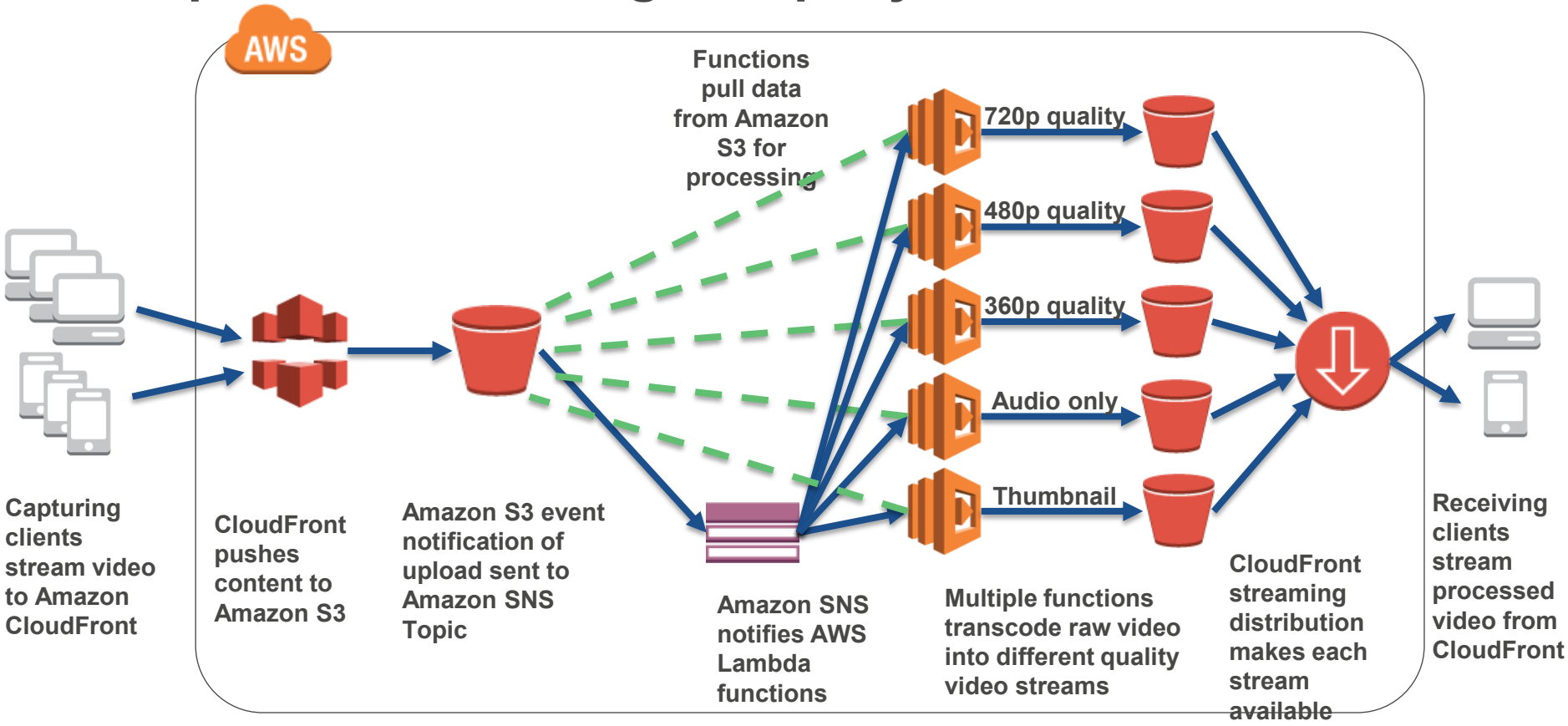
That's it.

AWS Lambda handles:

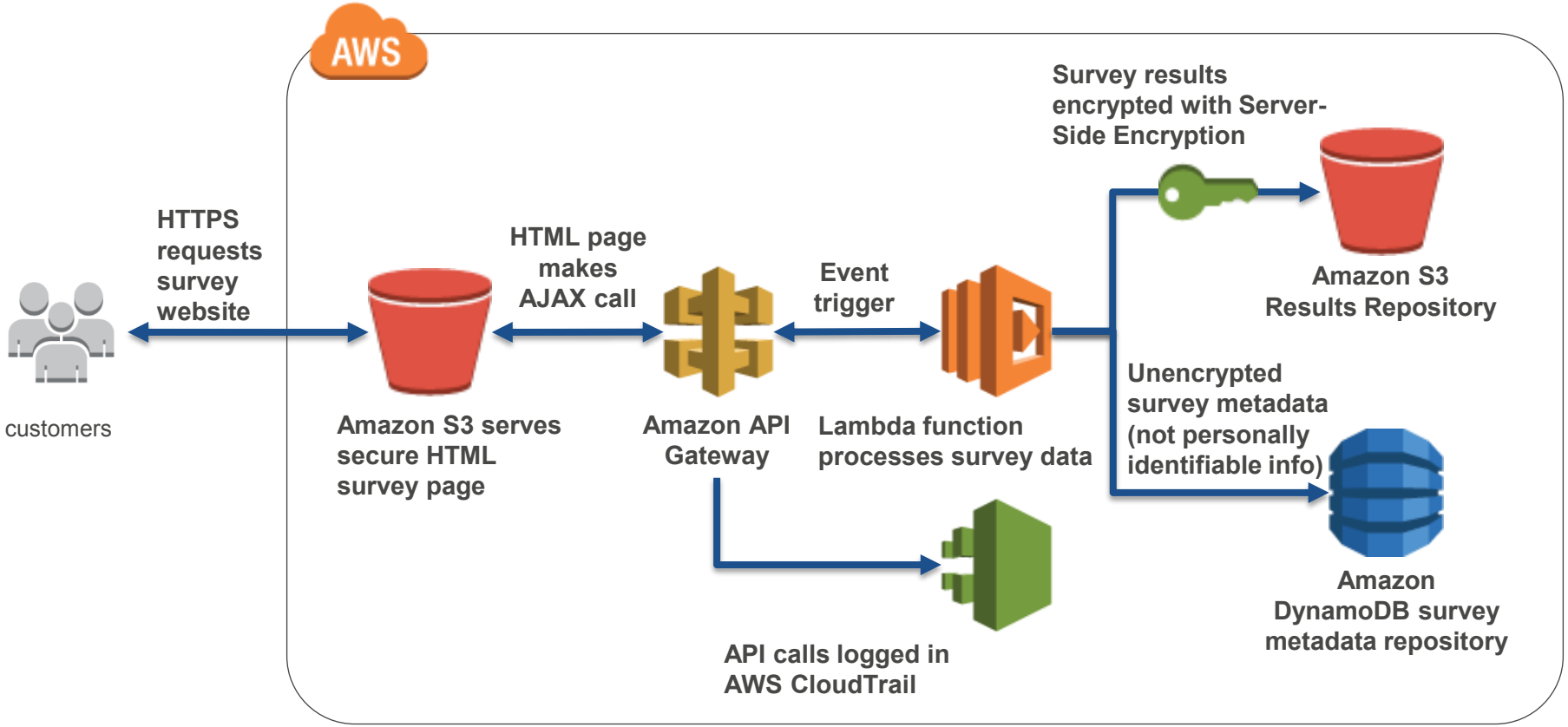
- Listening/polling
- Queueing
- Processing
- Autoscaling
- Redundancy
- Load balancing

Decoupling Examples

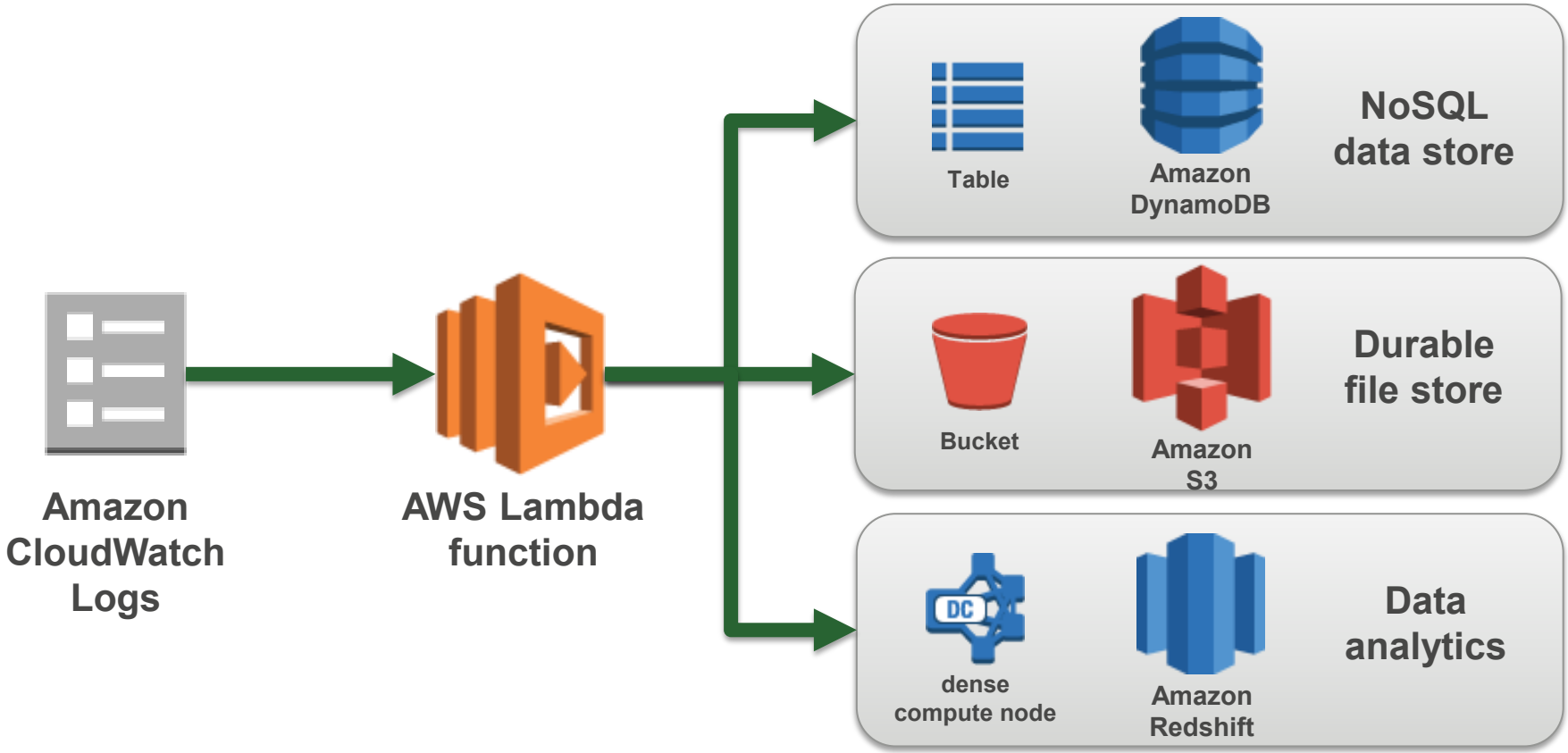
Example: Livestreaming Company



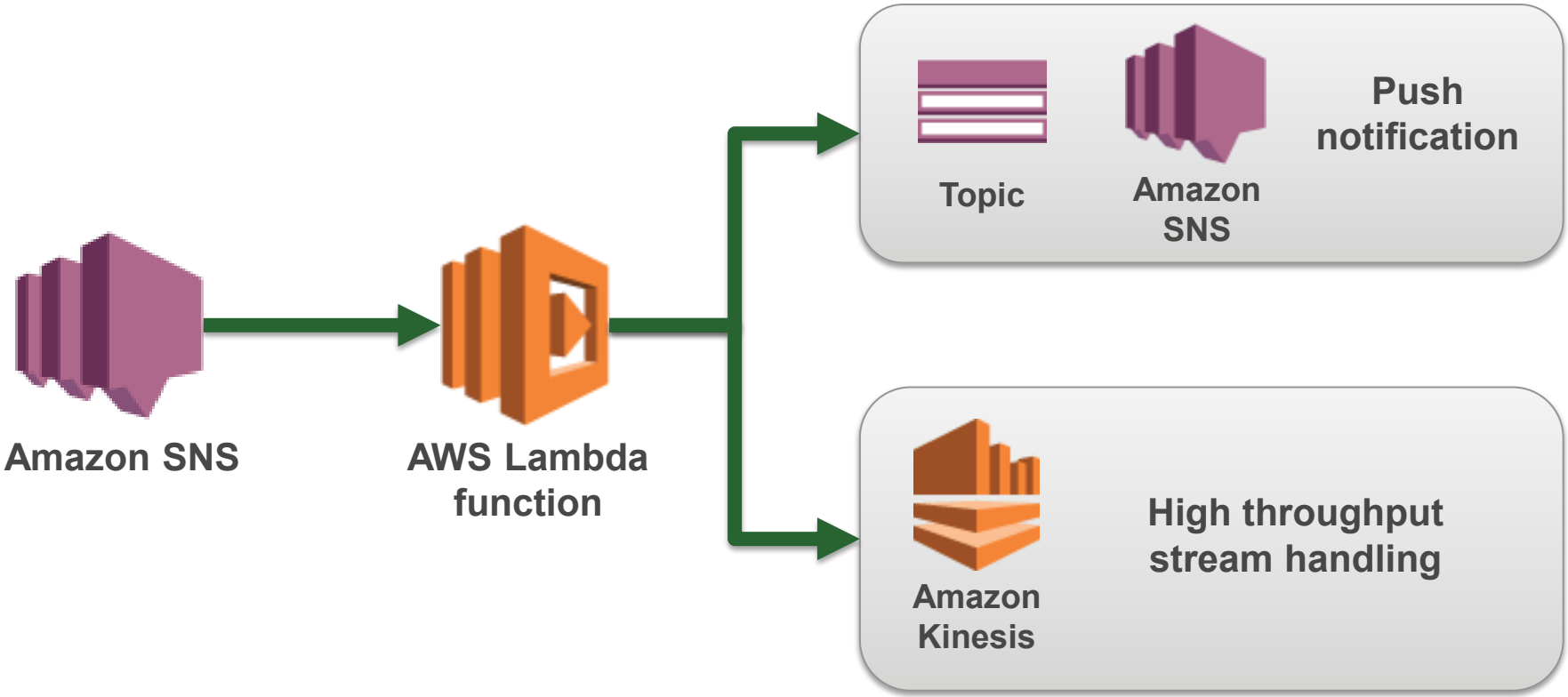
Example: Serving Secure Surveys



Example: Processing Amazon CloudWatch Logs



Example: Real-Time Message Handling Workflow



Example: CRUD Backend Workflow