

# Beyond Lambdas

Modern Serverless Design Patterns on AWS

# Agenda

- What's "Modern Serverless"?
- Event-Driven Decoupling
- Orchestration vs. Choreography
- API Composition & Real-Time Apps
- Modern Auth with Cognito
- Observability, Retries, & Failure Handling
- Bringing It All Together

What's "Modern Serverless"?

# No Longer Just Lambda

Then (2016):

- Write a Lambda function.
- Wire it to API Gateway.
- Save to DynamoDB.
- Done. 🎉



# Modern Serverless ≠ Just Lambda

## Modern serverless apps are:

- Event-driven with EventBridge, SNS, SQS
- Workflow-driven with Step Functions
- Composable with AppSync and GraphQL
- User-centric with Cognito or third-party auth
- Observable and resilient out of the box

Modern serverless is about architecture, not just code.

# The “Everything Lambda” Anti-Pattern

- Every feature lives in one Lambda
- Hard-coded business logic + database access
- Direct integration between services
- Debugging is painful
- Scaling is tied to code execution

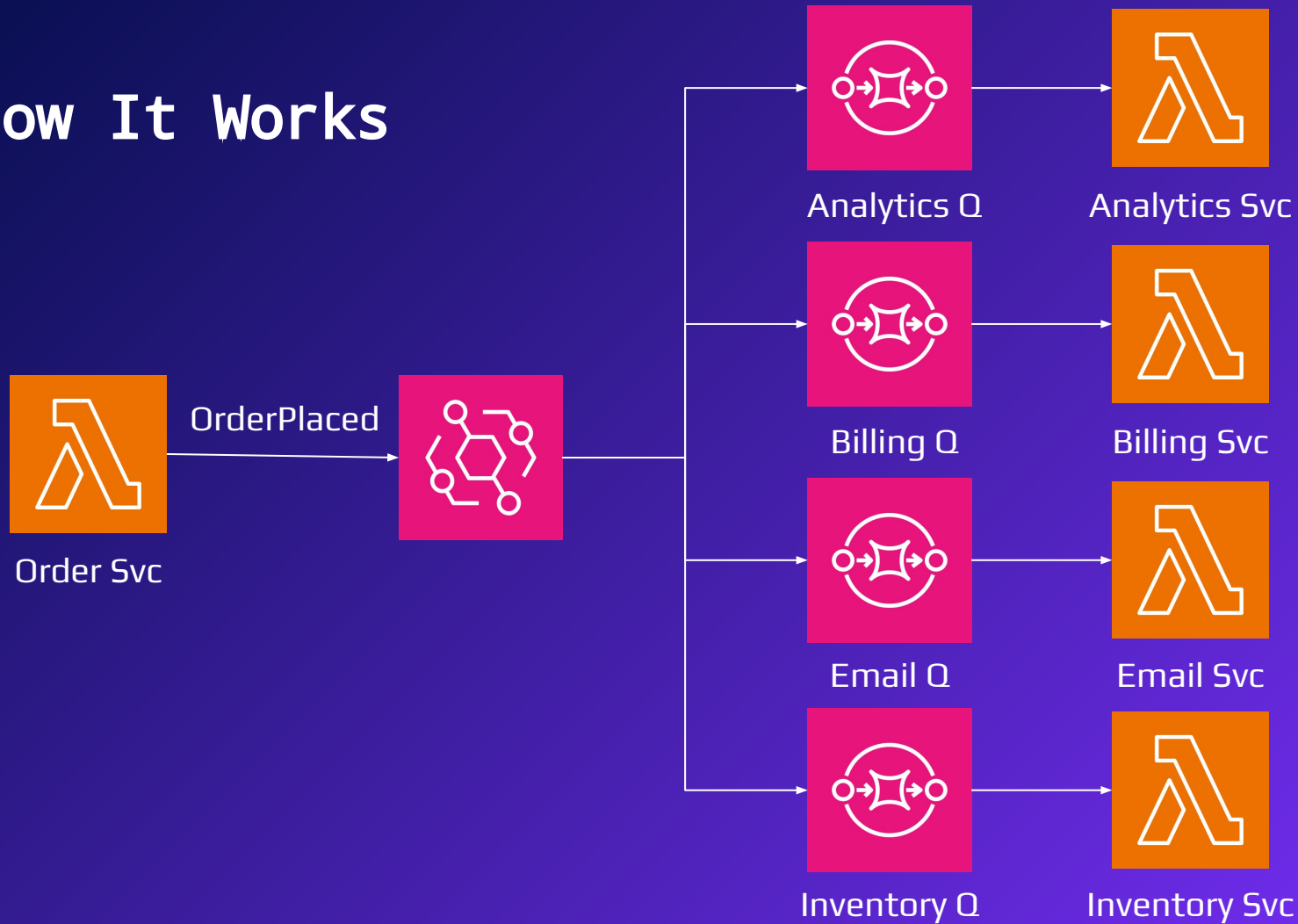
# Event-Driven Decoupling

# Decouple Everything: EventBridge + SQS

- EventBridge acts as the central event bus to decouple producers from consumers
- Use SQS queues to ensure reliable delivery and allow scalable fan-out
- Avoid tight coupling between services—no more point-to-point dependencies
- Enables modular, reusable services that don't have to know about each other
- Improves fault isolation and simplifies testing



# How It Works



# Code Example: Publishing an Event

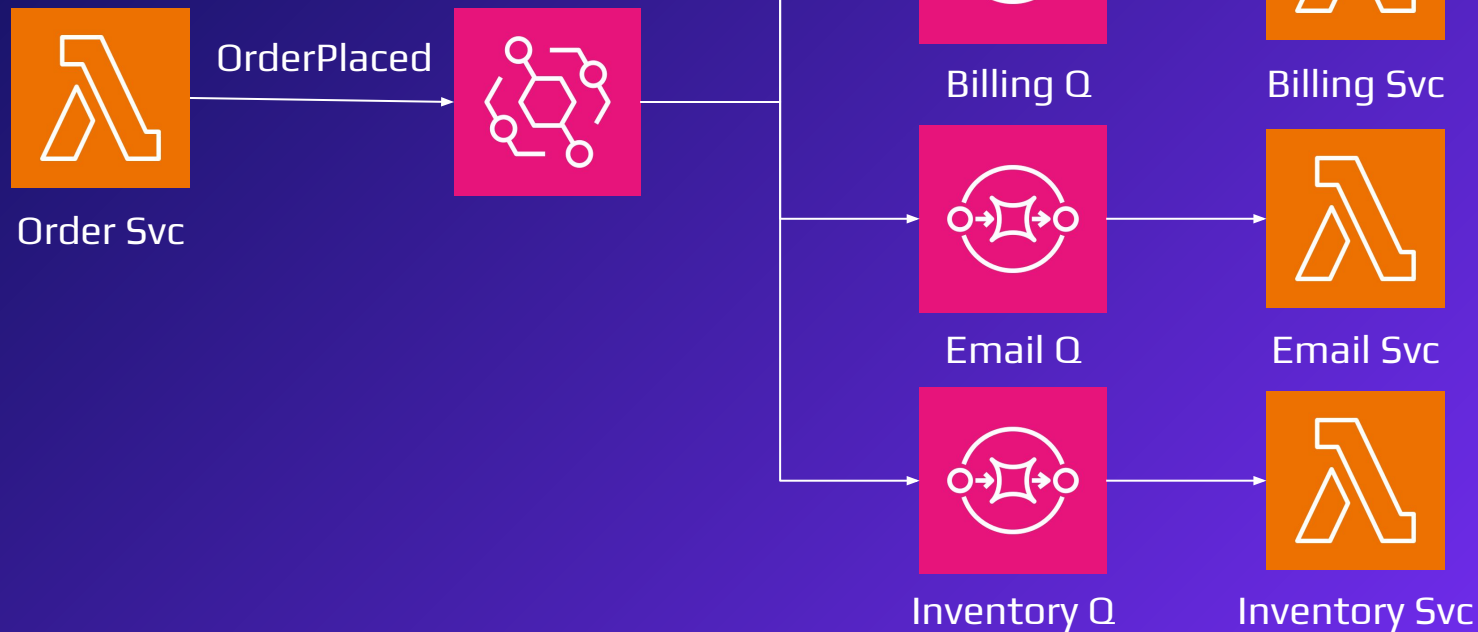
```
exports.handler = async (event: any) => {  
  const detail = event;  
  const params = {  
    Entries: [  
      {  
        Source: 'custom.orders',  
        DetailType: 'OrderPlaced',  
        Detail: JSON.stringify(detail),  
        EventBusName: process.env.EVENT_BUS_NAME || 'default',  
      },  
    ],  
  };  
  
  await client.send(new PutEventsCommand(params));  
  console.log('OrderPlaced event sent:', detail);  
  return { statusCode: 200, body: 'Event sent' };  
};
```

# Orchestration vs. Choreography (Step Functions)

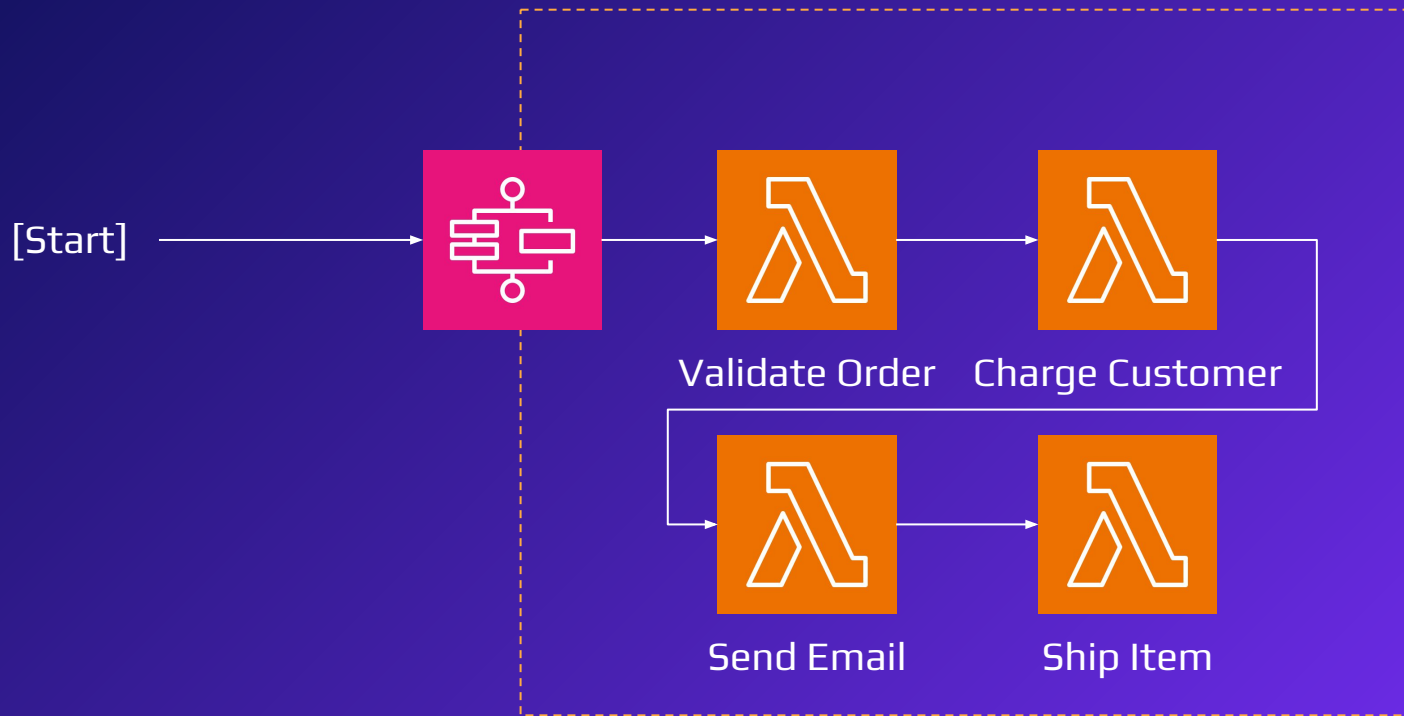
# Step Functions or EventBridge? *It Depends*

Feature	Orchestration (Step Functions)	Choreography (EventBridge)
Coordination	Centralized	Distributed
Visibility	High (visual workflows, state tracking)	Low (logs, tracing required)
Flexibility	Rigid but clear	Flexible but harder to trace
Cost	Higher per step	Cheaper (pay per event)
Ideal For	Complex, tightly timed workflows	Loose coupling, optional consumers

# EventBridge Choreography

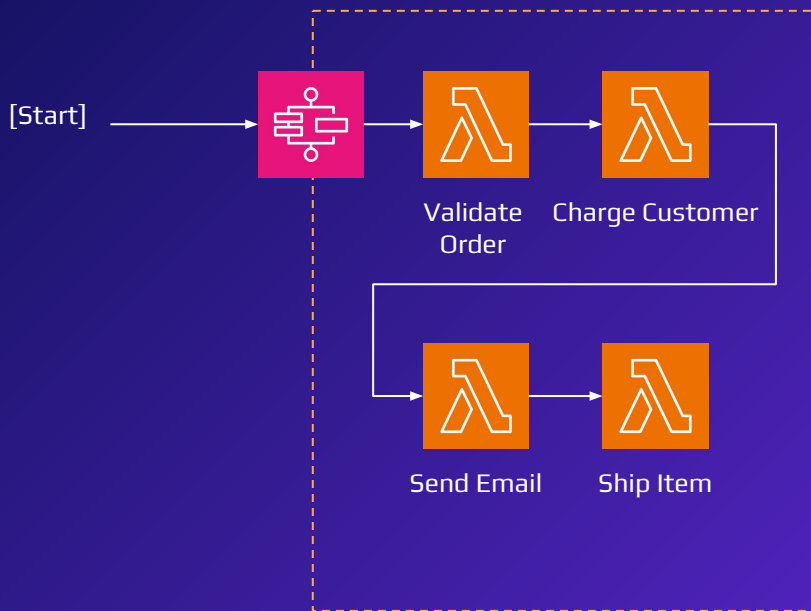


# Step Functions Orchestration

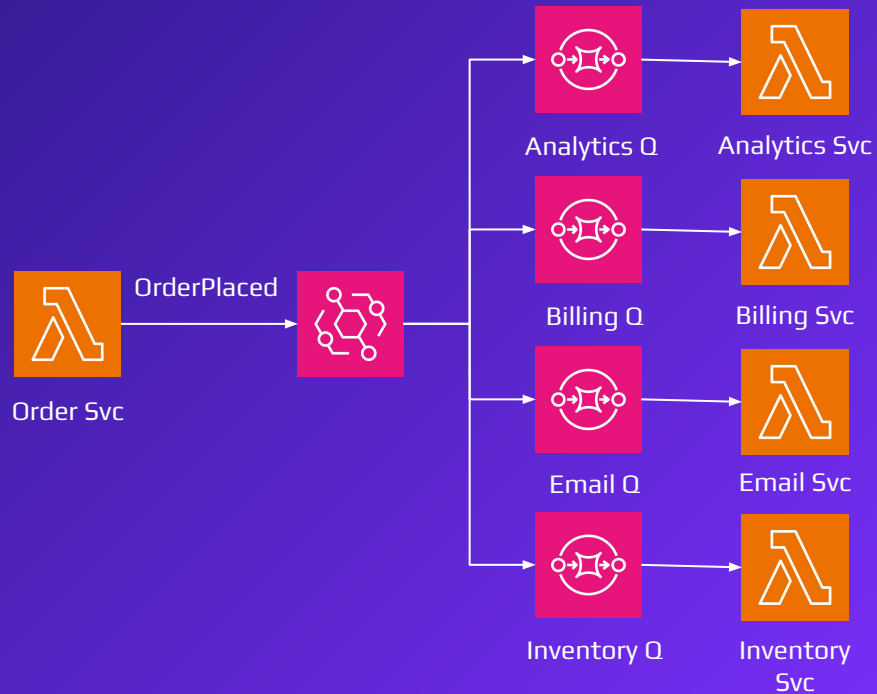


# Architecture Comparison: Order Workflow

## Step Functions Orchestration



## EventBridge Choreography



# When to Use What

- Use Step Functions when:
  - You need strict sequencing
  - You want built-in retries, wait times, or error handling
  - You need long-running workflows (e.g. approvals)
- Use EventBridge when:
  - You want independent services to subscribe to events
  - Your services may evolve separately
  - You want to add new consumers without code changes

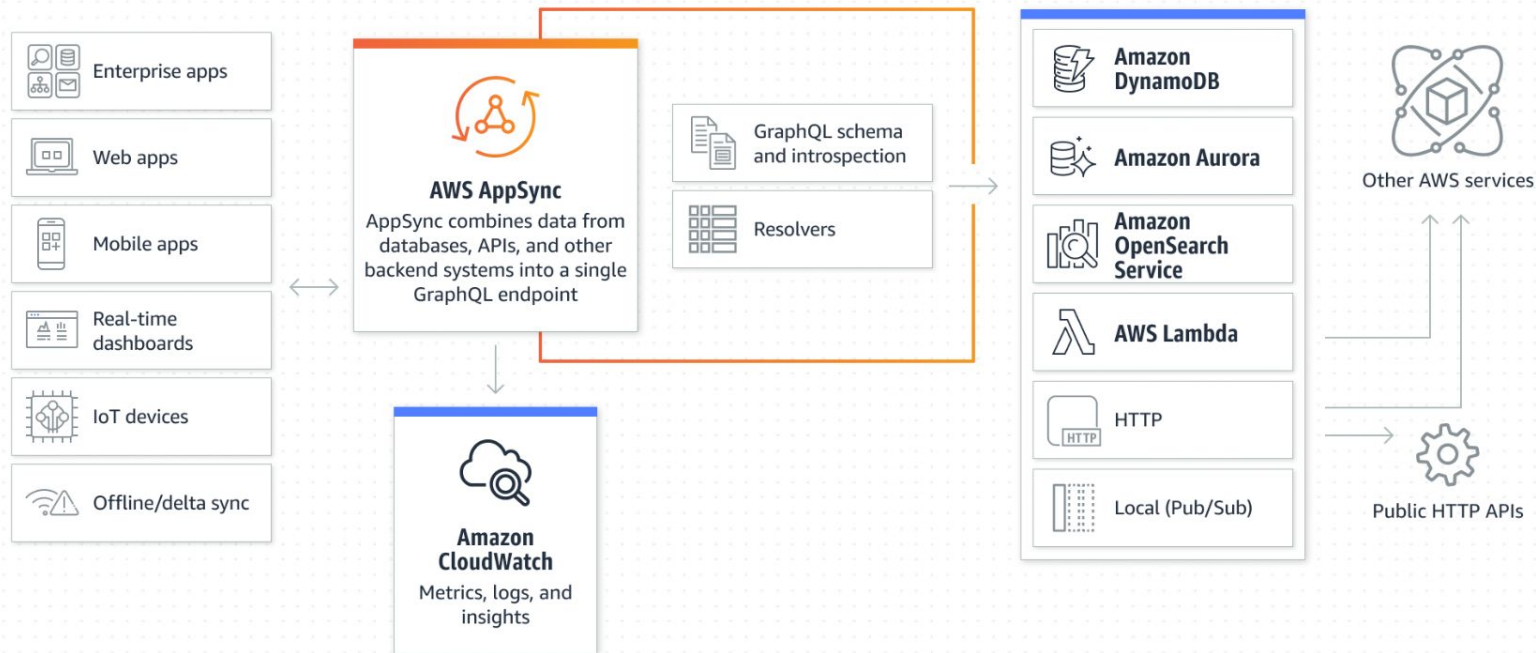


# API Composition & Real-Time Apps (AppSync + GraphQL)

# Why GraphQL + AppSync Makes Sense for Serverless

- GraphQL = Ask for exactly what you need
- Composes data from multiple sources (DynamoDB, Lambda, Aurora, HTTP APIs)
- Supports real-time updates via GraphQL subscriptions
- Eliminates over-fetching and under-fetching
- Perfect for modern frontend frameworks (React, Vue, mobile apps)

# AWS AppSync Components



# Sample Query & Real-Time Power

```
query GetUserDashboard {  
  user(id: "123") {  
    name  
    recentOrders {  
      id  
      total  
      items {  
        name  
        quantity  
      }  
    }  
    notifications {  
      message  
      createdAt  
    }  
  }  
}
```

# Real-Time with Subscriptions

```
mutation PlaceOrder {  
  placeOrder(input: { userId: "123",  
    items: [...] }) {  
    id  
    total  
    status  
  }  
}
```



```
subscription OnOrderUpdate {  
  onOrderUpdate(userId: "123") {  
    id  
    status  
    updatedAt  
  }  
}
```

# Modern Auth with Cognito

# Built-in Identity for Serverless Apps

- Cognito provides user sign-up, sign-in, and access control with federated identity support (Google, Apple, SAML, etc.)
- User Pools handle authentication (username/password, social login)
- Identity Pools issue temporary AWS credentials for accessing services like S3 or AppSync
- Works seamlessly with AppSync, API Gateway, and IAM policies
- Supports token-based auth (ID token, access token, refresh token)

# Gotchas and Best Practices

 <b>Pitfall</b>	 <b>Best Practice</b>
Customizing the Hosted UI is limited	Use <b>Amplify Auth UI</b> or <b>custom UI + SDK</b>
Hard to debug token issues	Use <b>JWT.io</b> + browser dev tools
Role management can get messy	Use <b>Cognito Groups</b> + IAM role mappings
Limited built-in analytics	Pair with <b>Pinpoint</b> or <b>CloudWatch Logs</b>



# Observability, Retries, & Failure Handling

# Observability: See Everything or Fly Blind

- CloudWatch Logs: default for all Lambda invocations
- AWS X-Ray: trace across services (Lambda, Step Functions, API Gateway)
- Powertools for AWS Lambda:
  - Structured logging
  - Tracing
  - Metrics (CloudWatch Embedded Metric Format)
- Compatible with 3rd-party tools: Datadog, Lumigo, Sentry

# Retry & Failure Patterns

AWS Service	Built-In Retry	DLQ Support	Timeout Control
Lambda (sync)	Yes	Yes	Yes
EventBridge	Yes (up to 24h)	Yes	No
SQS	Yes (client)	Yes	Yes
Step Functions	Yes (config)	N/A	Yes (per state)

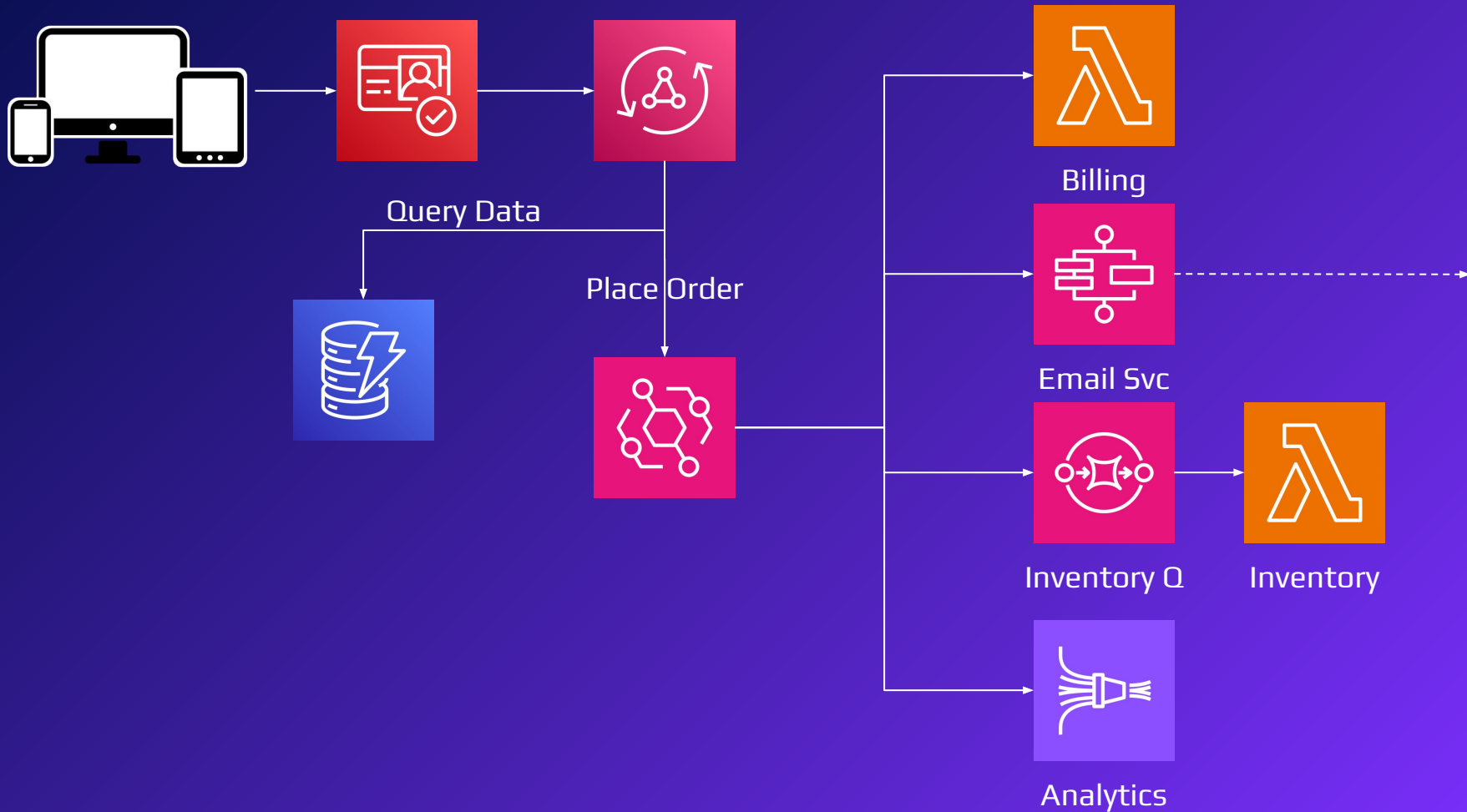
# Tactics for Resilience

- Use timeouts to avoid zombie executions
- Add retry policies (but cap attempts!)
- Set up DLOs + CloudWatch alarms
- Add fallback paths in Step Functions (Catch states)
- Monitor with dashboards + anomaly detection

```
"Retry": [  
  {  
    "ErrorEquals": ["States.ALL"],  
    "IntervalSeconds": 2,  
    "MaxAttempts": 3,  
    "BackoffRate": 2.0  
  }  
]
```

*Step Function retry block*

**Bringing It All Together**



# Key Takeaways

- Where to Start

- Find a tight coupling in your current app—can it become an event?
- Replace fragile dependencies with EventBridge or Step Functions
- Simplify your API layer with AppSync and GraphQL
- Add observability with AWS Powertools and DLOs
- Try a new pattern this week—even in a side project or a prototype

- Resources

- AWS Serverless Patterns Collection - [serverlessland.com/patterns](https://serverlessland.com/patterns)
- AWS Well-Architected Serverless Lens - [docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens](https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens)
- <https://github.com/chaotictoejam/beyond-lambdas-serverless-patterns>