AWS ONLINE TECH TALKS

Enable cross-team, federated GraphQL APIs with AWS AppSync Merged APIs

Brice Pelle

Principal Product Manager, Front-End Web & Mobile
AWS

Agenda

GraphQL

- Why GraphQL

AWS AppSync

- Why AppSync
- Benefits

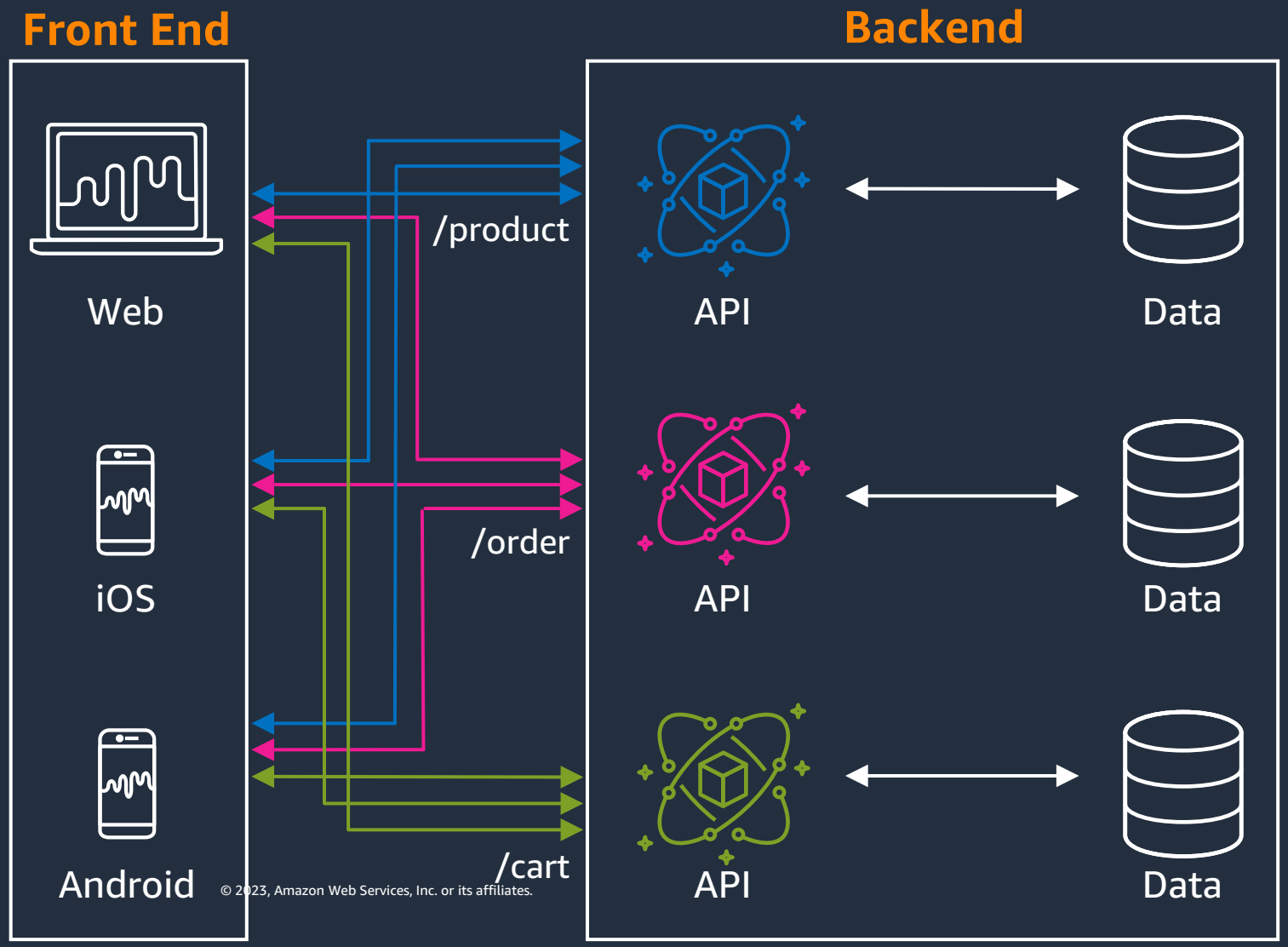
AWS AppSync Merged APIs

- Why Merged APIs
- Benefits
- Demo

GraphQL

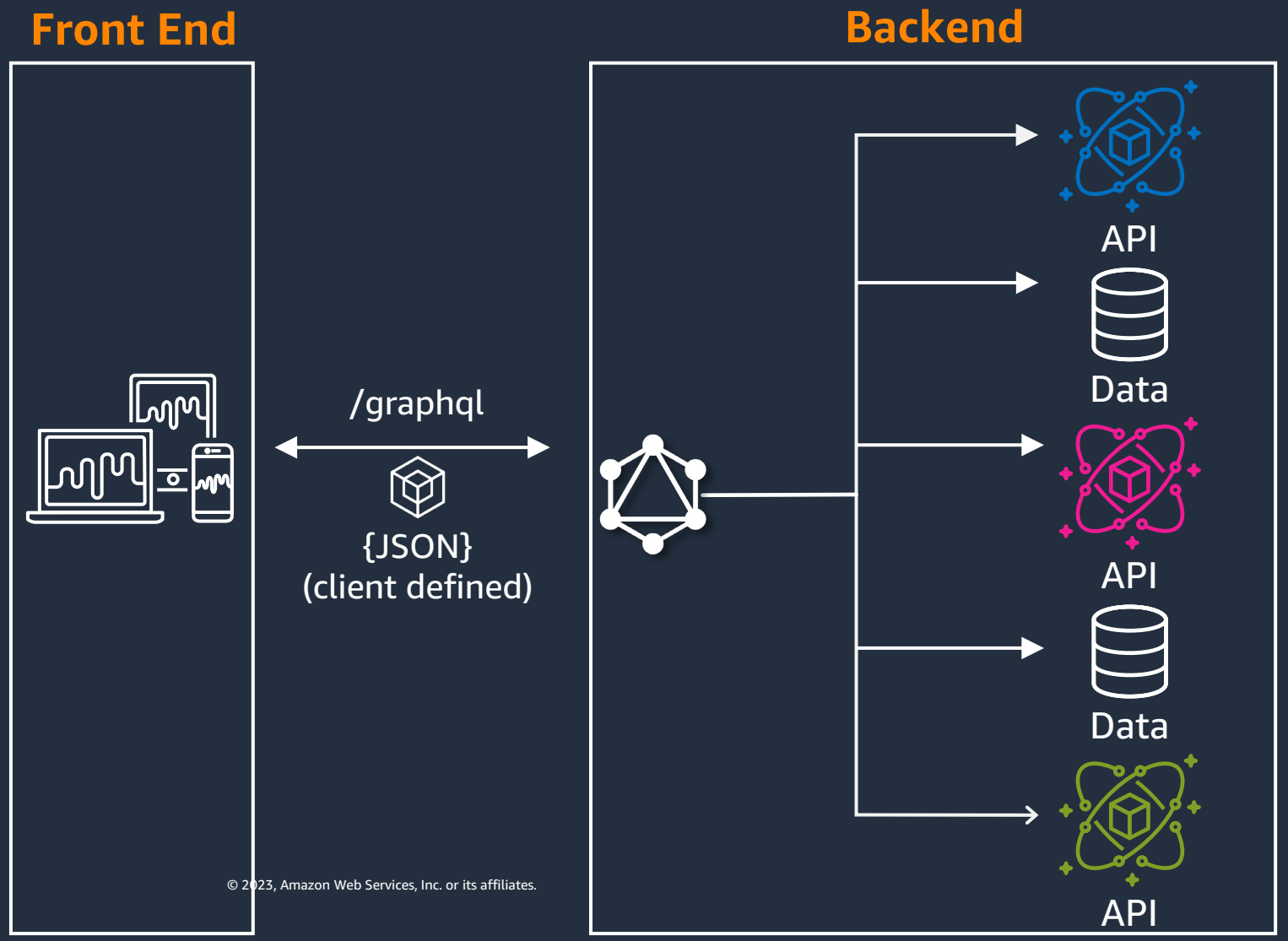
Challenges with traditional API architectures

- Data spread across hundreds of databases and microservices
- Inconsistent approaches for querying and control
- Sequential loading, over/under fetching, static payloads
- Reduced feature velocity



How GraphQL simplifies API architectures

Unified access to fetch only the data you need through a single API call



AWS AppSync

Self-management of GraphQL is complex, costly, tedious

Tedious infrastructure management

Complex server setup, config and administration; Upgrades, patches adds to operational overhead

More code means more complexity

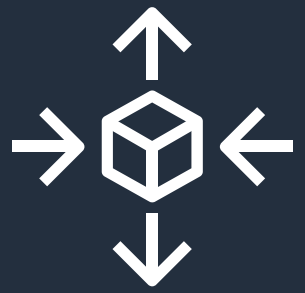
Developers still have to write undifferentiated code for auth, cache, HA, instrument for observability, encryption etc.

Ensuring security with high performance is tricky

Since APIs provide exposed interface to data, security is essential; requires skillset for self-management

AWS AppSync

SIMPLEST WAY TO BUILD SCALABLE APIS THAT CONNECT APPLICATIONS TO DATA



100% serverless

Fully managed GraphQL API and Pub/Sub API setup, administration and maintenance, infra provisioned for high availability

Less code, less complexity

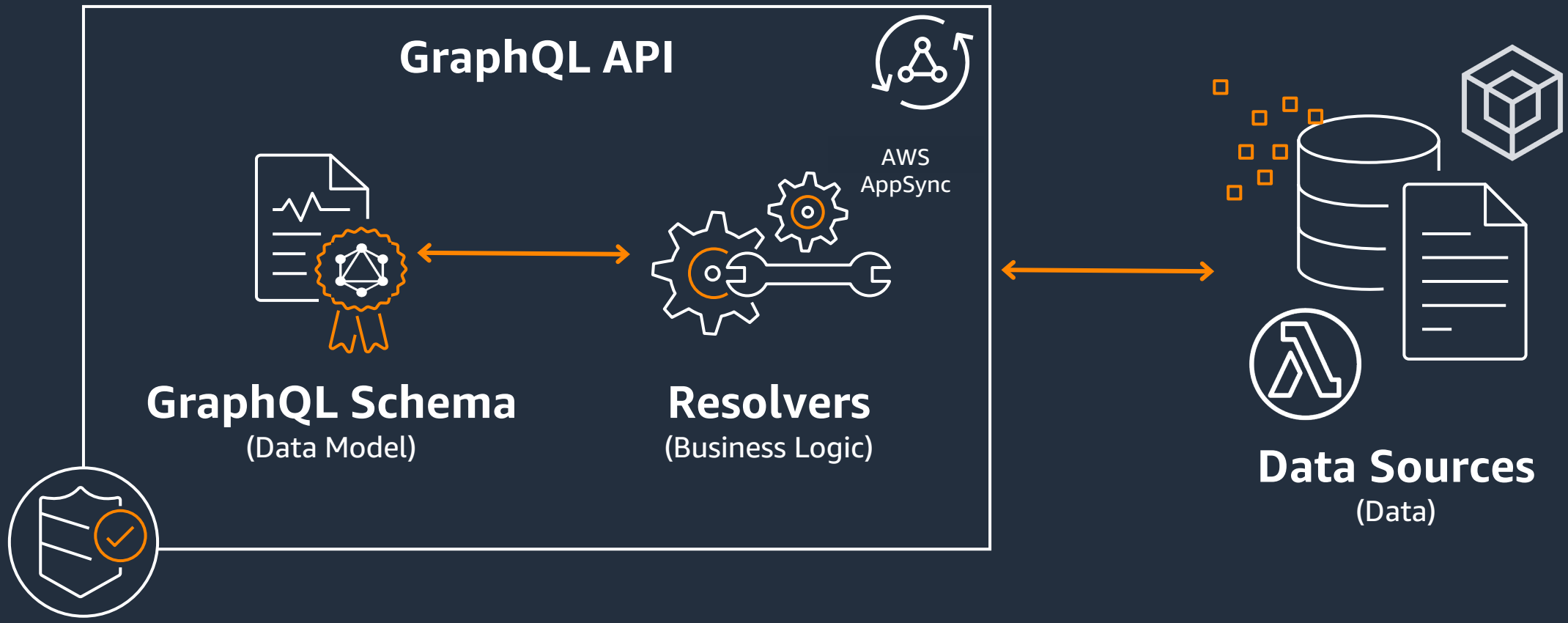
Handles low-value, utility code with support for AWS services e.g. CloudWatch, Cognito, DynamoDB etc to reduce code and associated manual errors

Secure and performant

Built-in resiliency with TLS encryption, secured APIs, custom domains, caches, auth, WAF, audit with CloudTrail; compliant with several security standards

What is AWS AppSync

MANAGED SERVERLESS GRAPHQL AND PUB/SUB API SERVICE



AWS AppSync benefits for front-end developer teams

EASIEST WAY TO BUILD A FEATURE-RICH APP QUICKLY

Build apps faster
without waiting on backend teams to create new APIs, interact with data from multiple data sources using single API

Build performant apps
with fetching only required data, and reducing number of network requests to fetch data for a particular optimized view

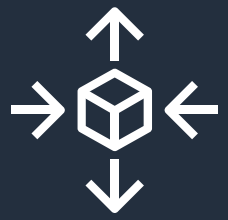
Serve advanced use cases
such as real-time using pub/sub, chat functionality, location-aware notifications, offline support and more

AWS AppSync benefits for back-end teams

MODERNIZE API STRATEGY AND IMPROVE EASE OF DATA ACCESS FOR FRONT-END TEAMS

Less complexity

Committed service levels for API availability, low API latency to improve app performance



Scalable

Scale up or down serverlessly to meet the demand of any API

AWS Ecosystem

Use AppSync as a gateway to any AWS service with built-in integrations

Security

Utilize auth, encryption, private APIs, activity logging and other AWS services to meet compliance objectives

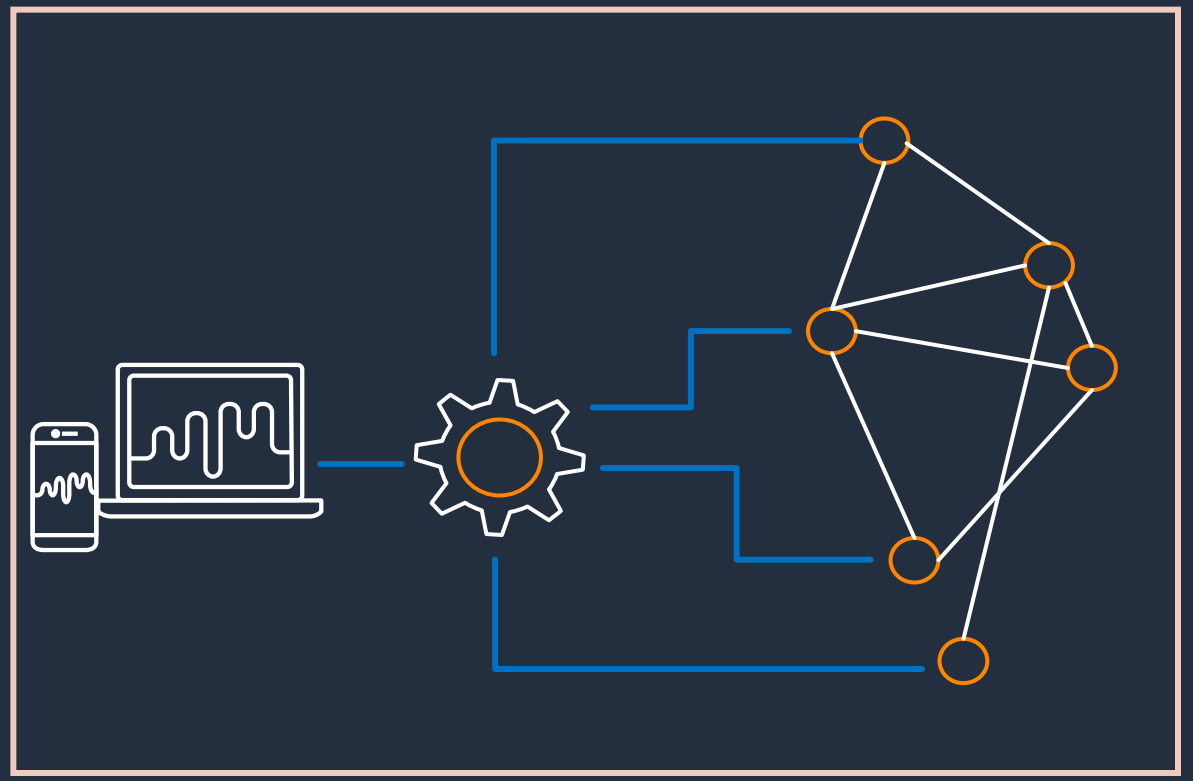
Collaboration

With independent operation by merging different team GraphQL APIs into one federated/ merged API

AWS AppSync Merged APIs

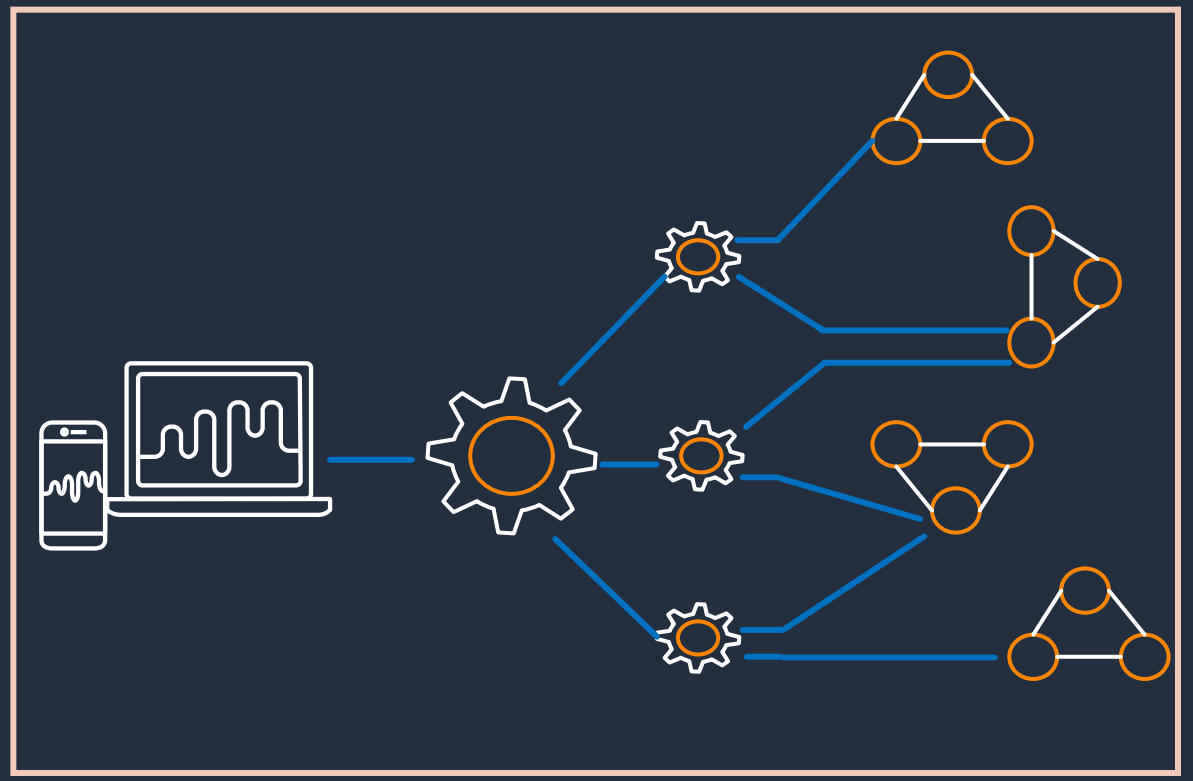
Requirements for multi-service environments

- Teams owning distinct but related GraphQL data types may need to work simultaneously in same API
- Working on same API without proper guardrails may lead to accidental breaking changes
- Important to ensure **isolation** while preserving **ownership** and **velocity**



Merged APIs for cross-team collaboration

- A **merged API** combines a group of AppSync GraphQL APIs owned and managed by different teams by composing a single GraphQL schema and securely exposing data from backend APIs to clients in a single endpoint
- Enables **cross-team collaboration** while **operating independently**



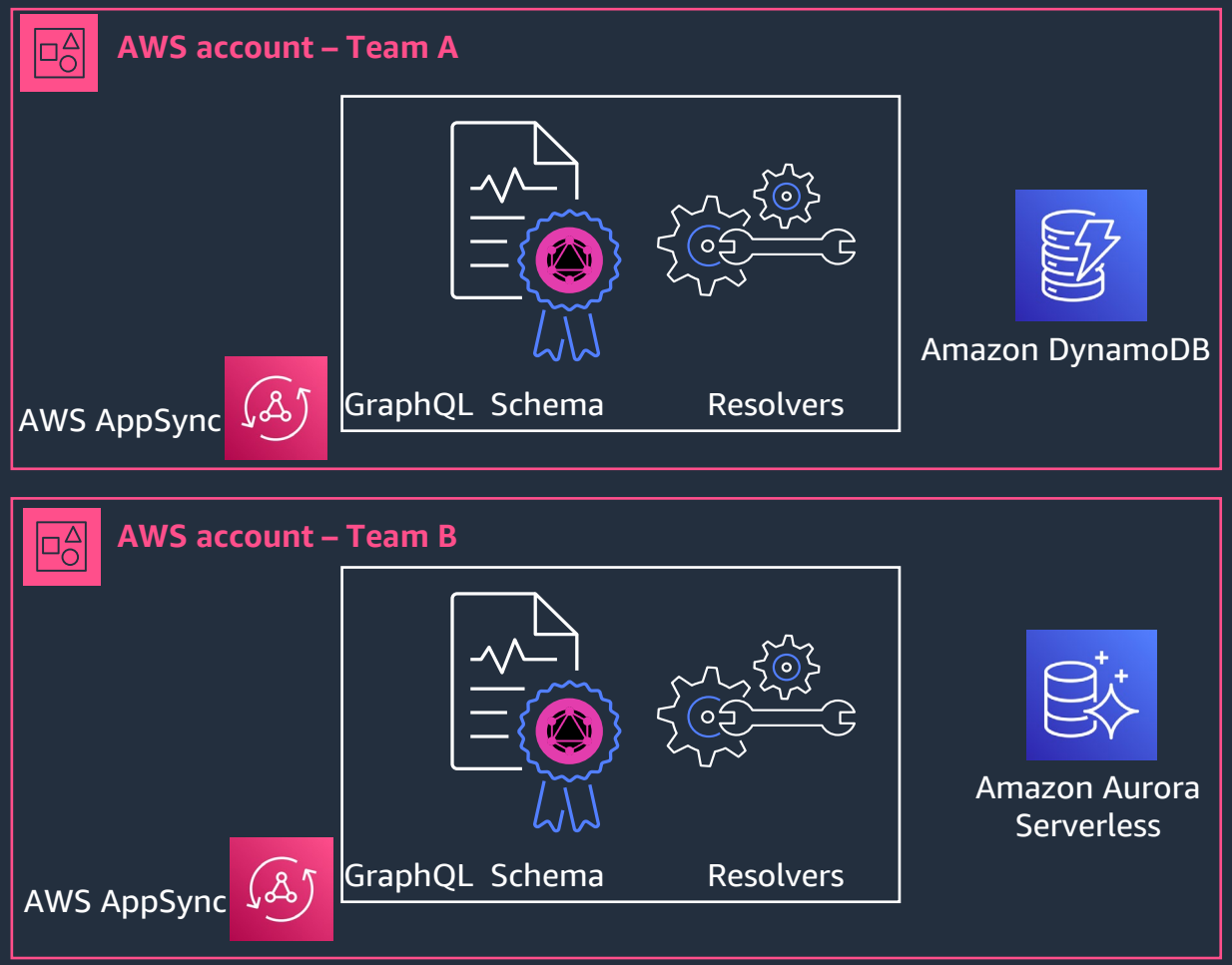
Merged APIs example

```
type Order {
  id: ID!
  total: Float
  inStock: Boolean
  date: AWSDate
  items: [Product]
}

Type Query {
   getOrder(id: ID!): Order
}
```

Schema A

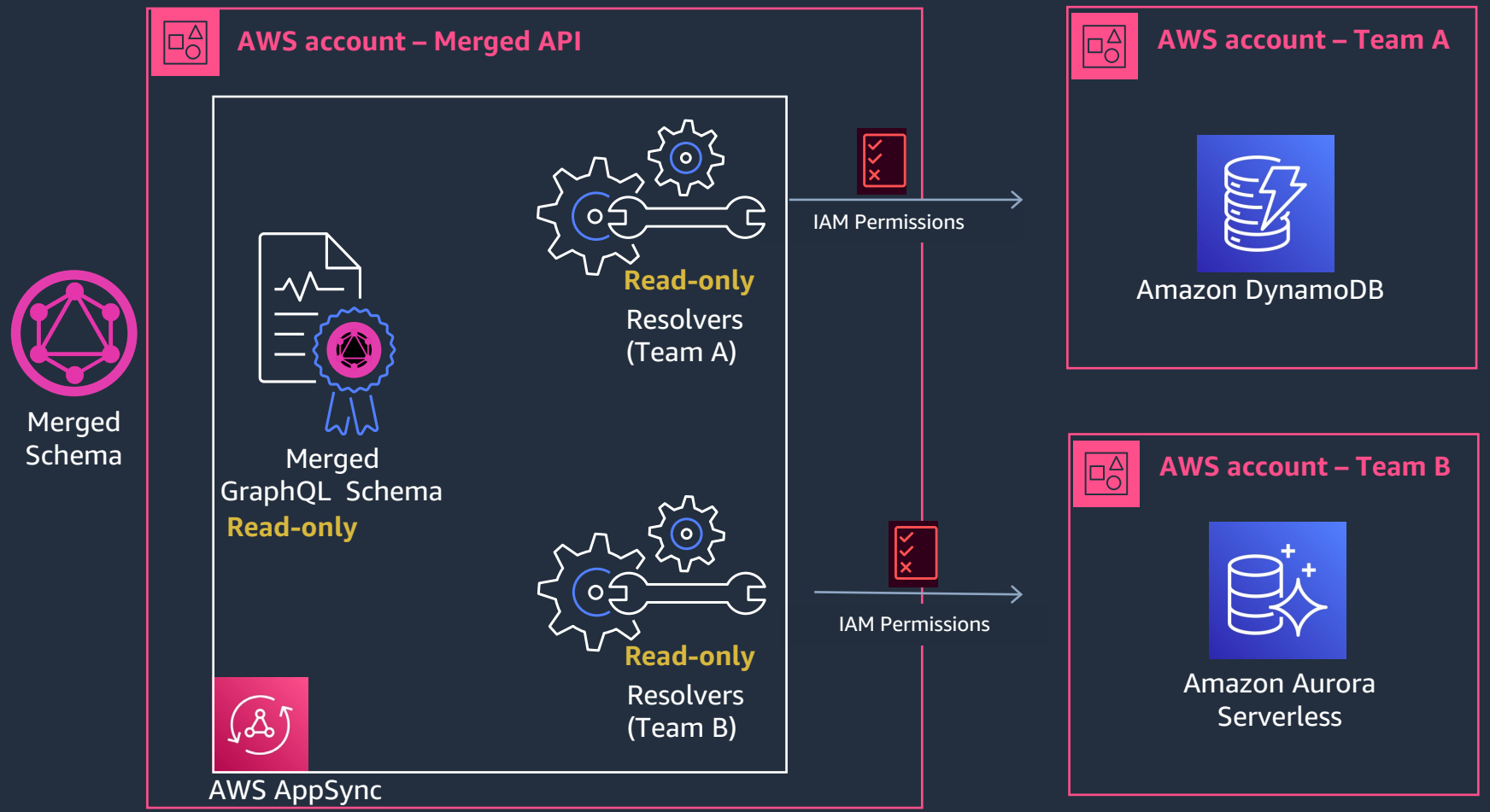
```
type Product {
  id: ID!
  name: String
  description: String
  price: Float
}

Type Query {
   getProduct(id: ID!): Product
}
```

Schema B



Merged APIs example

```
type Order {
  id: ID!
  total: Float
  inStock: Boolean
  date: AWSDate
  items: [Product]
}

type Product {
  id: ID!
  name: String
  description: String
  price: Float
}


Type Query {
   getOrder(id: ID!): Order
   getProduct(id: ID!): Product
}
```



AppSync Merged API approach vs. existing approaches

Runtime Composition

Existing approaches to federation

- Central gateway with secondary GraphQL APIs as part of query execution path
- Additional network hops adds complexity
- No support for real-time subscriptions

Build-time Composition

Merged APIs approach

- Single API composed at build time, no other APIs involved in the execution path
- Single server simplifies operations and performance management
- Real-time subscriptions fully supported

Merged APIs Demo

Thank you!

Brice Pelle
@BricePelle