PUBLIC SECTOR SUMMIT ONLINE

Building modern applications with AWS containers and serverless solutions

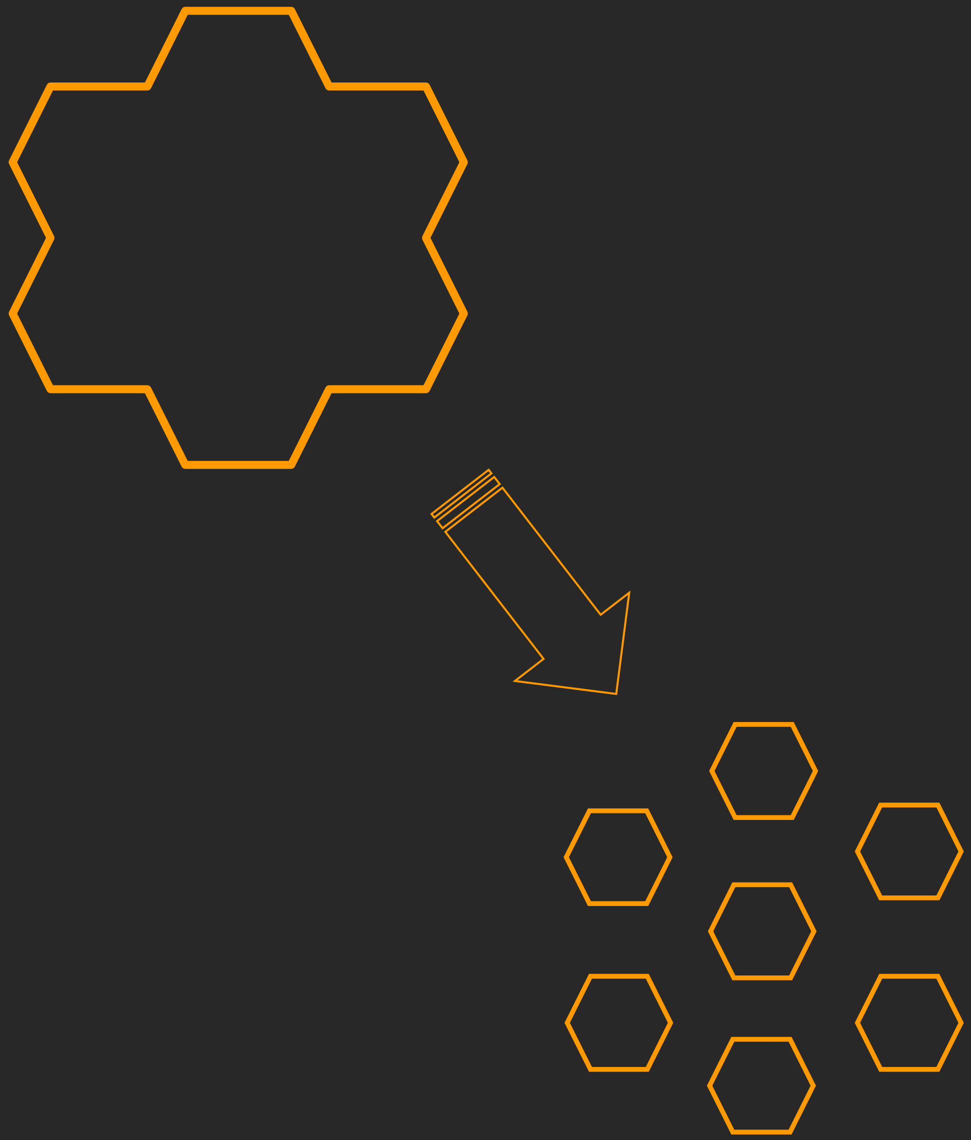
Stephanie Chiao

Developer Solution Architect, Technical Product Manager
Asia Pacific and Japan
Amazon Web Services

aws PUBLIC SECTOR SUMMIT ONLINE

Agenda

- Why are companies moving towards microservices?
- Addressing the complexities of microservices and how AWS can help you manage these
- Best practices for companies looking to break down monolithic architectures



Why are companies moving towards microservices?

Monolithic applications – traditional approach

Monolith

Does everything

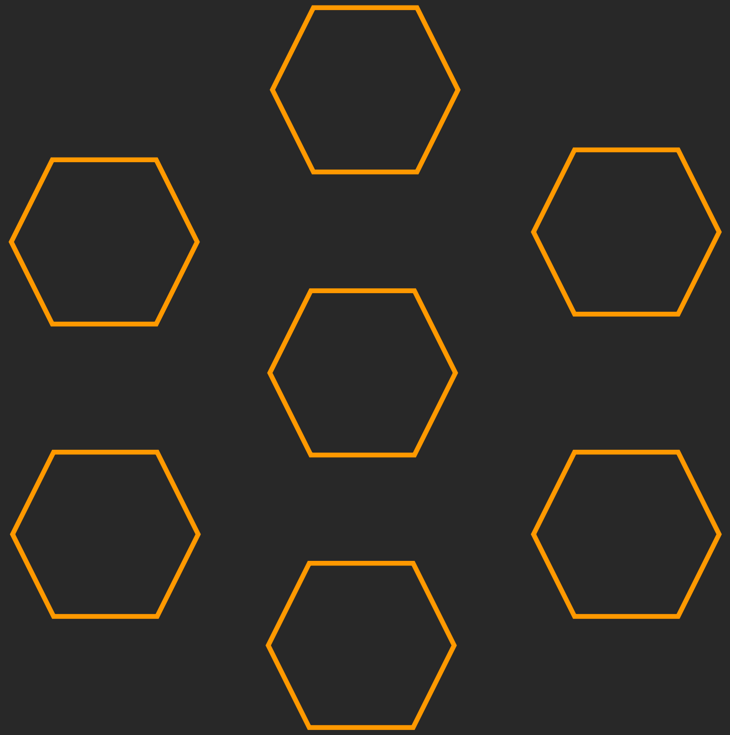
Advantages

- Single codebase
- Easier to develop
- All computational logic are in-memory

Disadvantages

- Applications can be difficult to modify
- All components have to be coded in the same language
- Long deployment cycles
- Requires a long-term commitment to a technology stack

Microservices – paradigm shift



Microservices

Does one thing

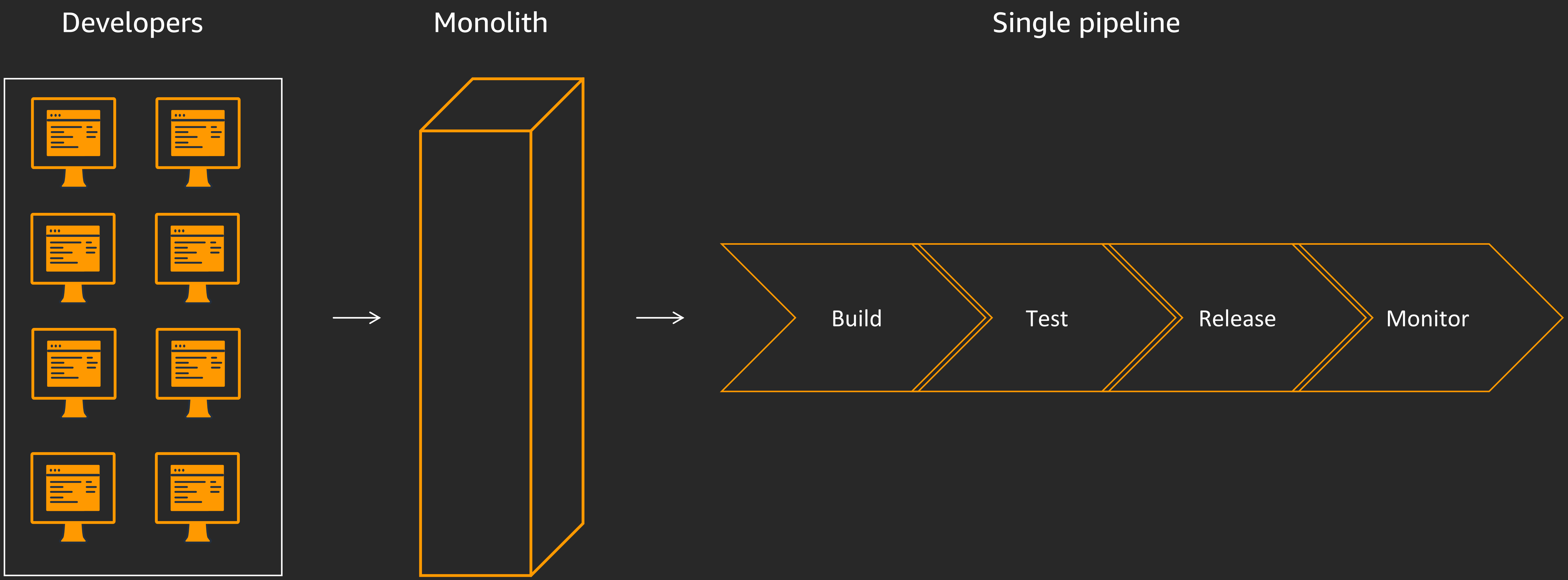
Key elements of microservices:

- Polyglot technical stack
- Independent components that run each application process as a service
- Communicate via APIs

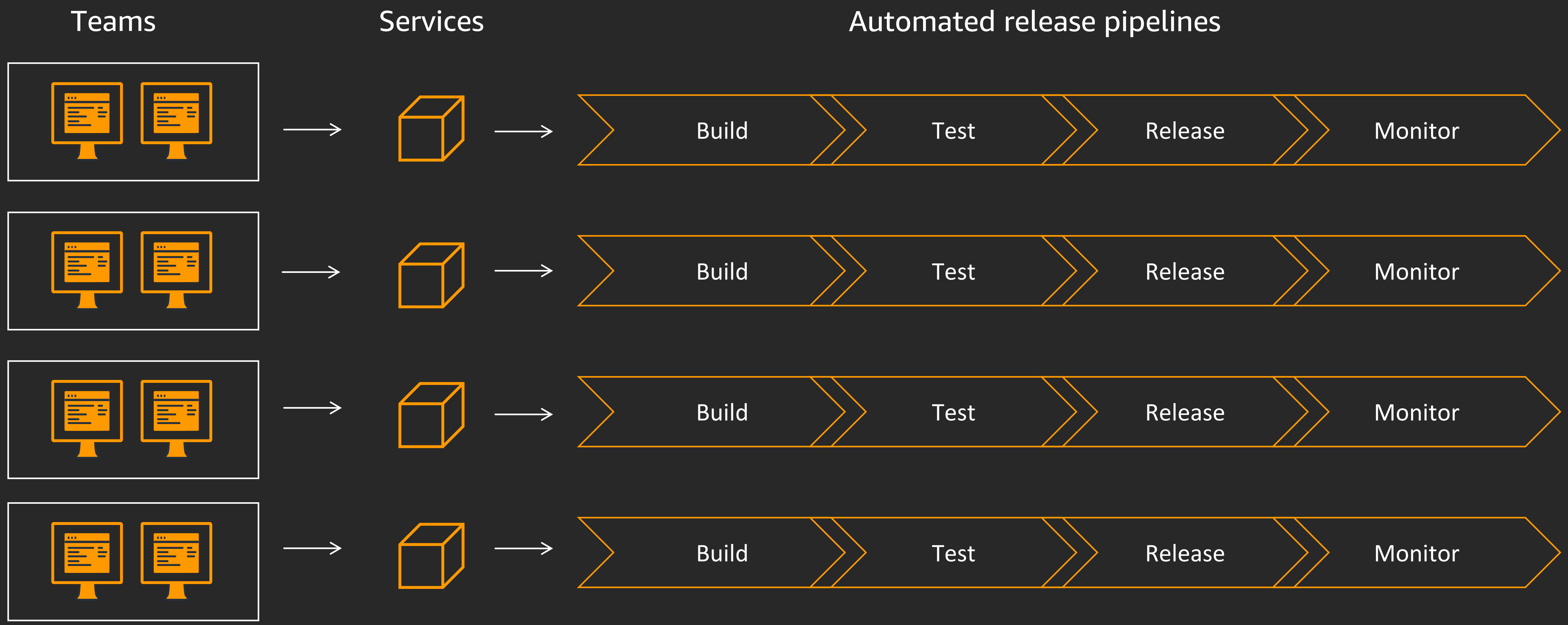
Each microservice is:

- **Elastic** – scales up or down independently of other services
- **Resilient** – services provide fault isolation boundaries
- **Composable** – uniform APIs for each service

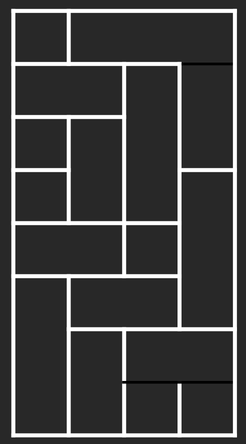
Long deployment cycles with monoliths to…

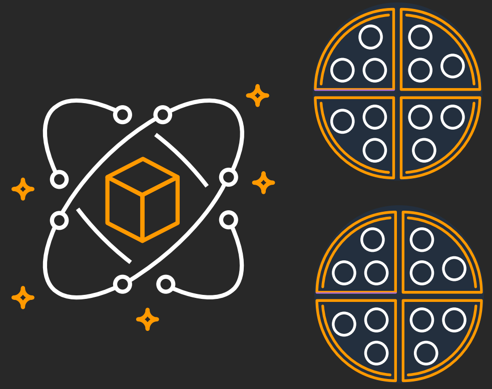


Rapid development with microservices



Development transformation at Amazon: 2001-2002





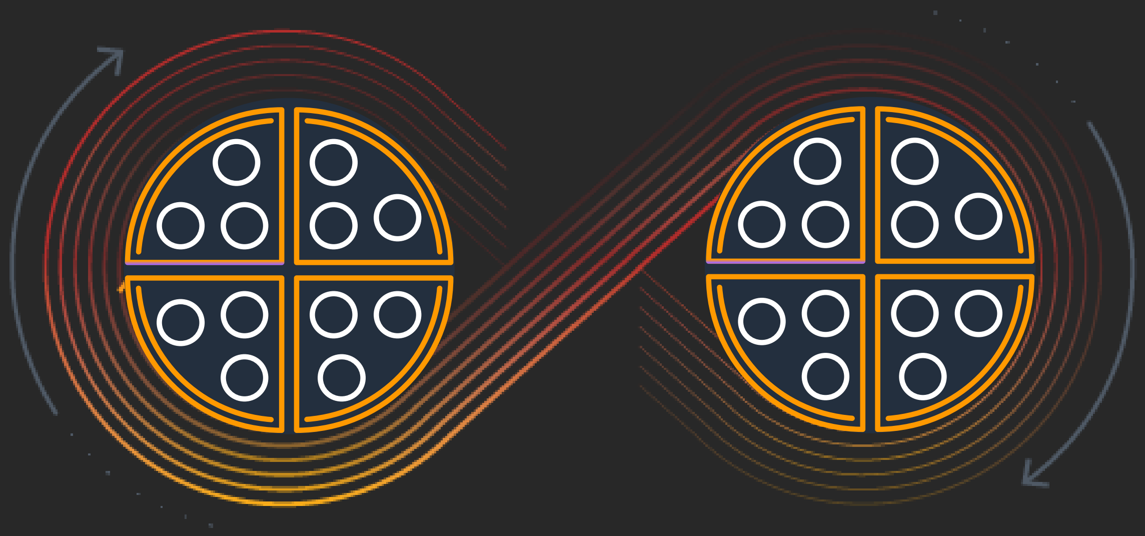
2001 → **2002**

Monolithic application + teams

Microservices + two pizza teams

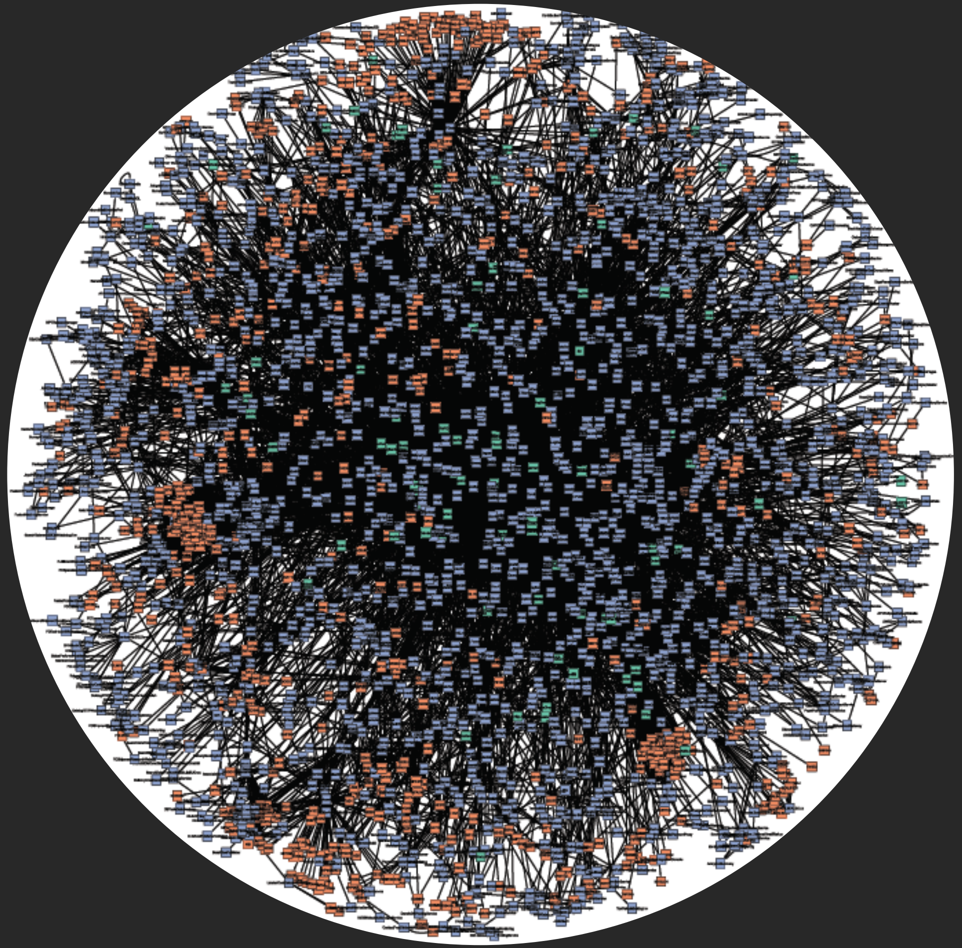
Two-pizza teams are fast and agile

- Full ownership and autonomy
- You build it, you run it
- DevOps – small, nimble teams
- Focused innovation



Shift to microservices

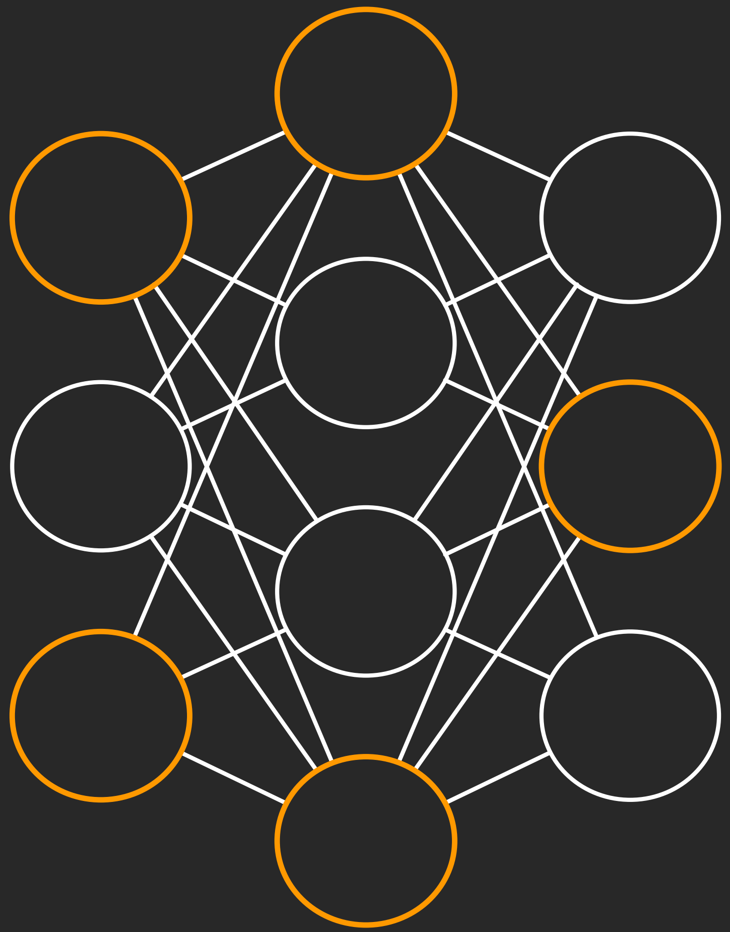
- Single-purpose
- Connect only through hardened APIs
- Largely “black boxes” to each other
- Business logic and data only accessible through APIs



Addressing the complexities of microservices and how AWS can help you manage these

Why isn't everyone moving to microservices?

Isn't all of this very hard now that we have lots of pieces to operate?



Overcoming the fear of complexity

Serverless functions or serverless containers?

How do I manage clusters at scale?

How would I know if one of my clusters has failed?

How do I decompose a monolithic database?

Serverless functions or serverless containers?

Lambda

Serverless functions

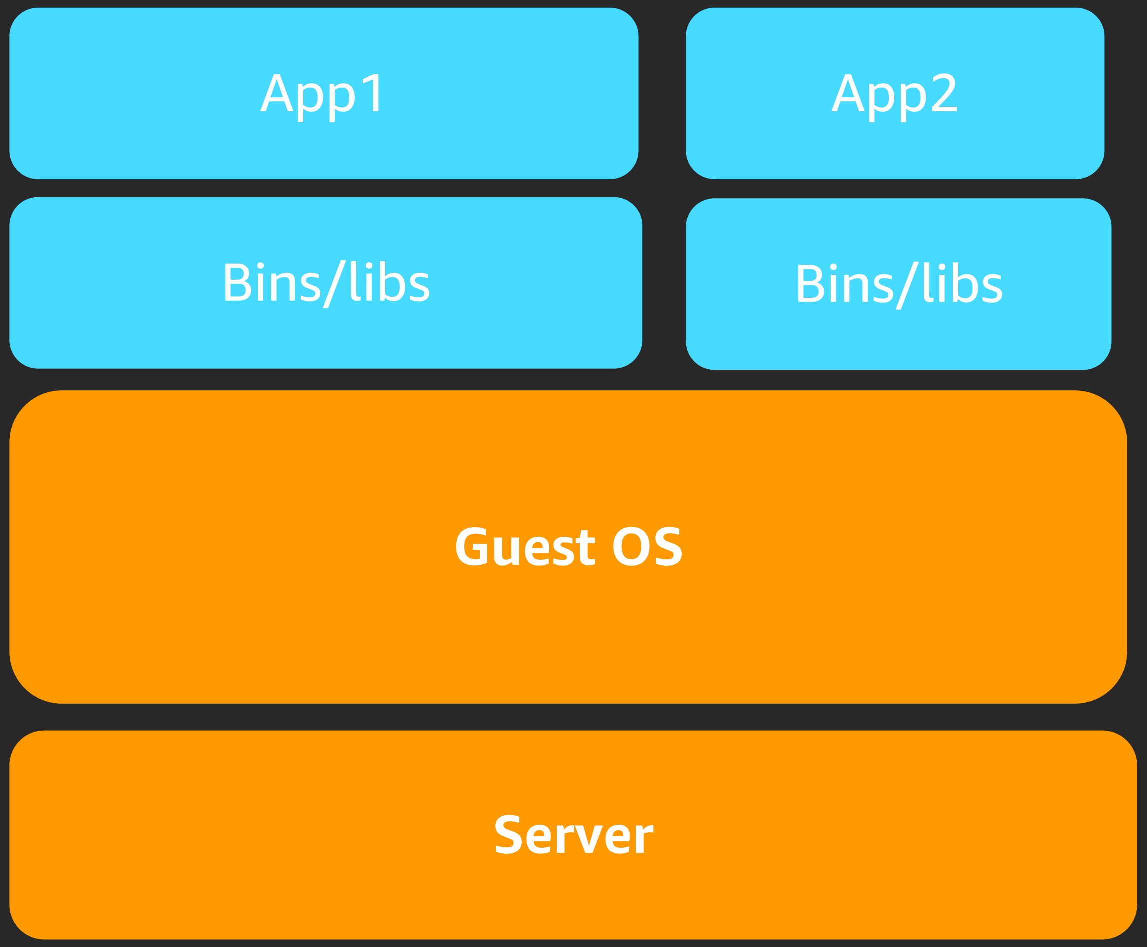
- Short lived
- Event driven
- Many language runtimes
- Data source integrations
- Fully managed infrastructure

Fargate

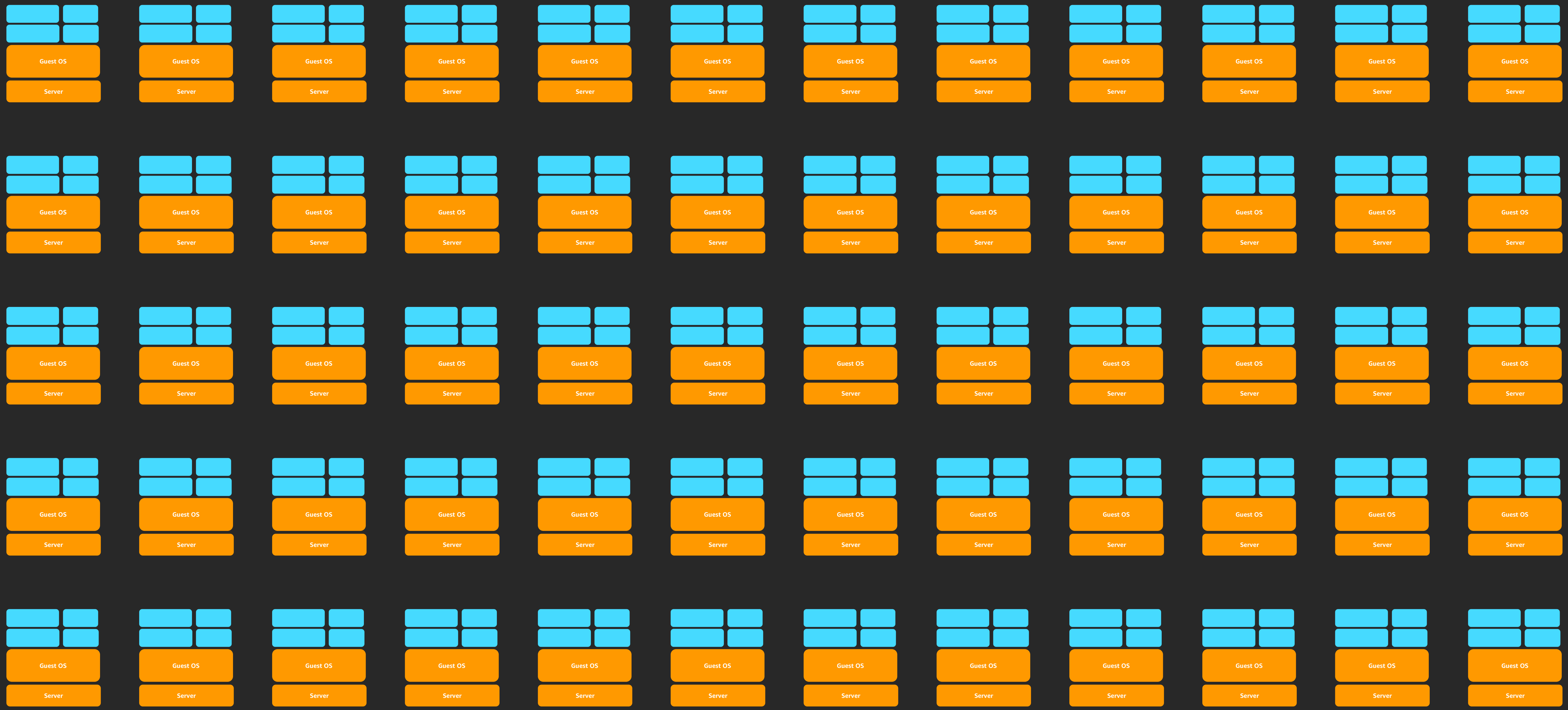
Serverless containers

- Long running
- Abstracts the platform
- Fully managed orchestration
- Fully managed cluster scaling

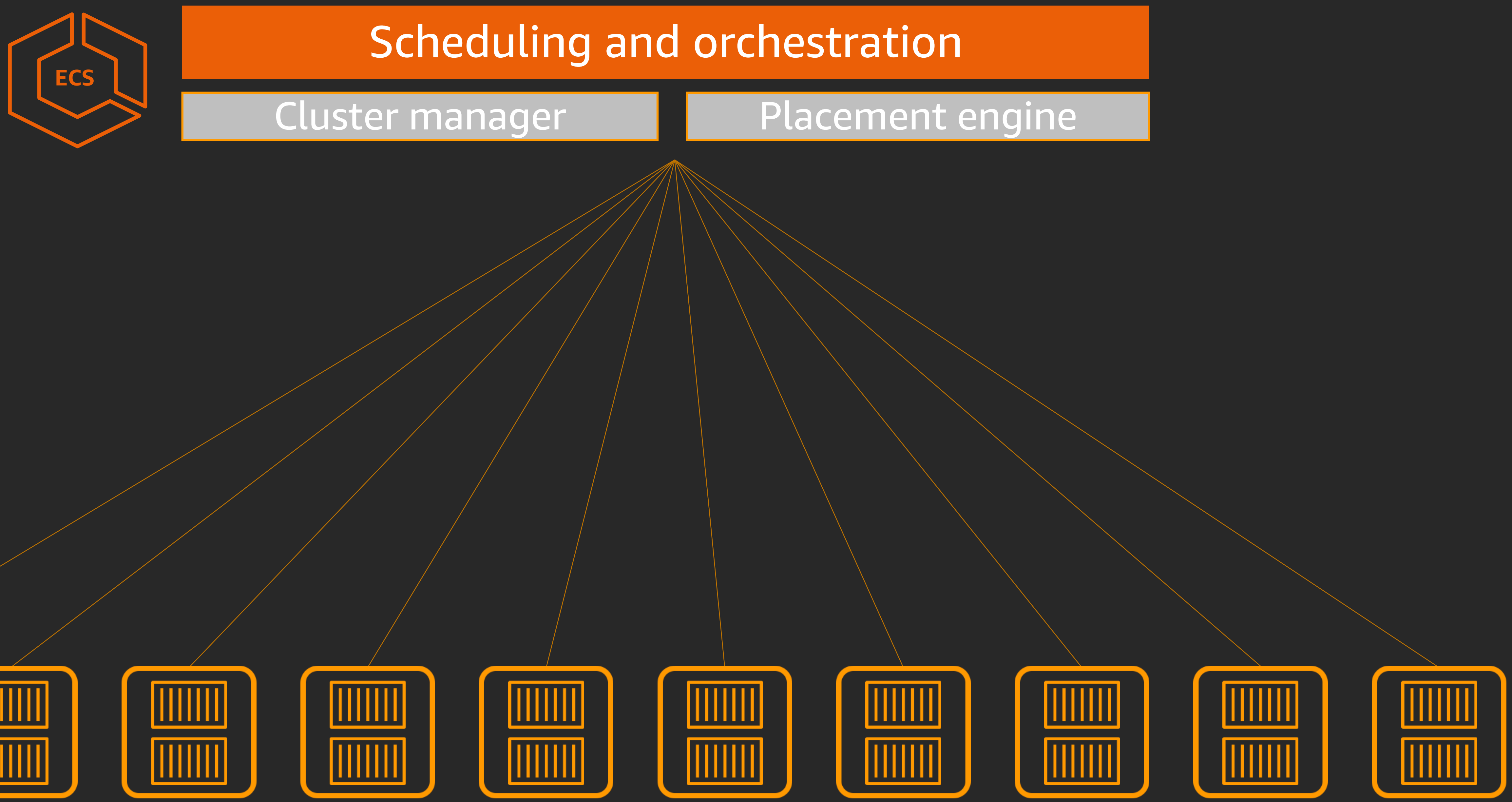
Managing one resource is straightforward



But imagine managing clusters at scale…



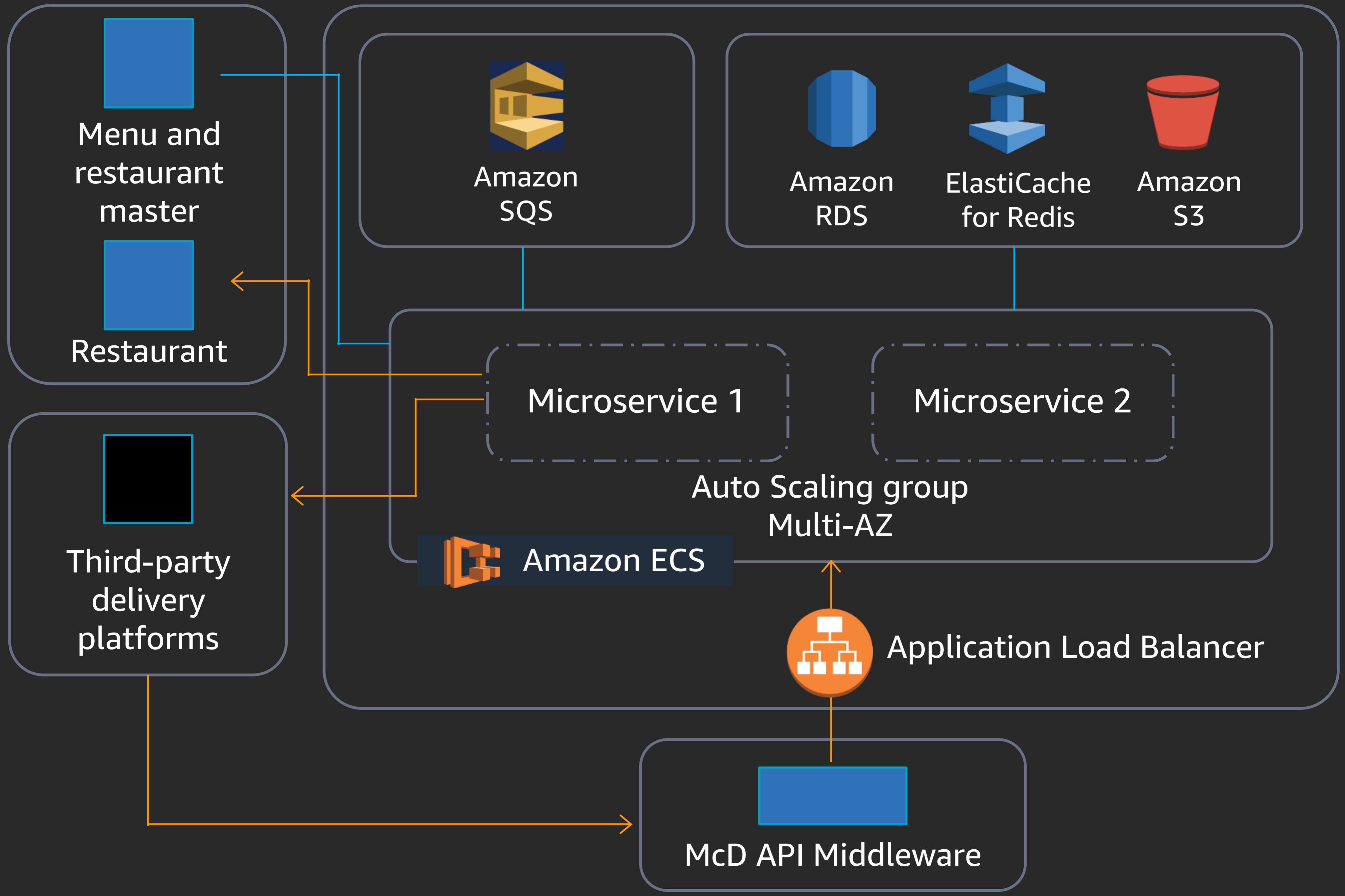
Cluster management made easy with Amazon ECS



Critical business requirements for McDonalds

- Speed to market
- Scalability and reliability
- Multi-country support and integration
- Cost sensitivity

Simplified view of McDonalds' architecture



Key business outcomes for McDonalds

- Four months from concept to production
- Achieved scale targets of 250k-250k order per hour with ~100 ms latency
- Increased speed to market
- Polyglot tech stack using .net and Java

How would I know if one of these clusters has failed?

Guest OS
Server

Logging, monitoring, and tracing

- Having the ability to aggregate data is key to discovering trends in errors in services
- System incident and event monitoring systems are needed
- Key considerations:
  - What resources will you monitor?
  - How often will you monitor these resources?
  - What monitoring tools will you use?
  - Who will perform the monitoring tasks?
  - Who should be notified when something goes wrong?

Building blocks for observability in AWS

CloudWatch

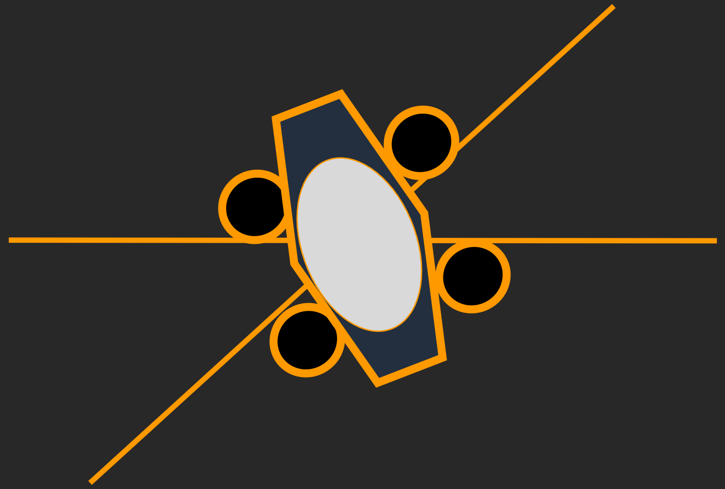
- Metrics
- Logs
- Events
- Alarms
- Dashboard

X-Ray

- Trace
- Debugging
- Service map
- Latency detection (server and client)

The data problem

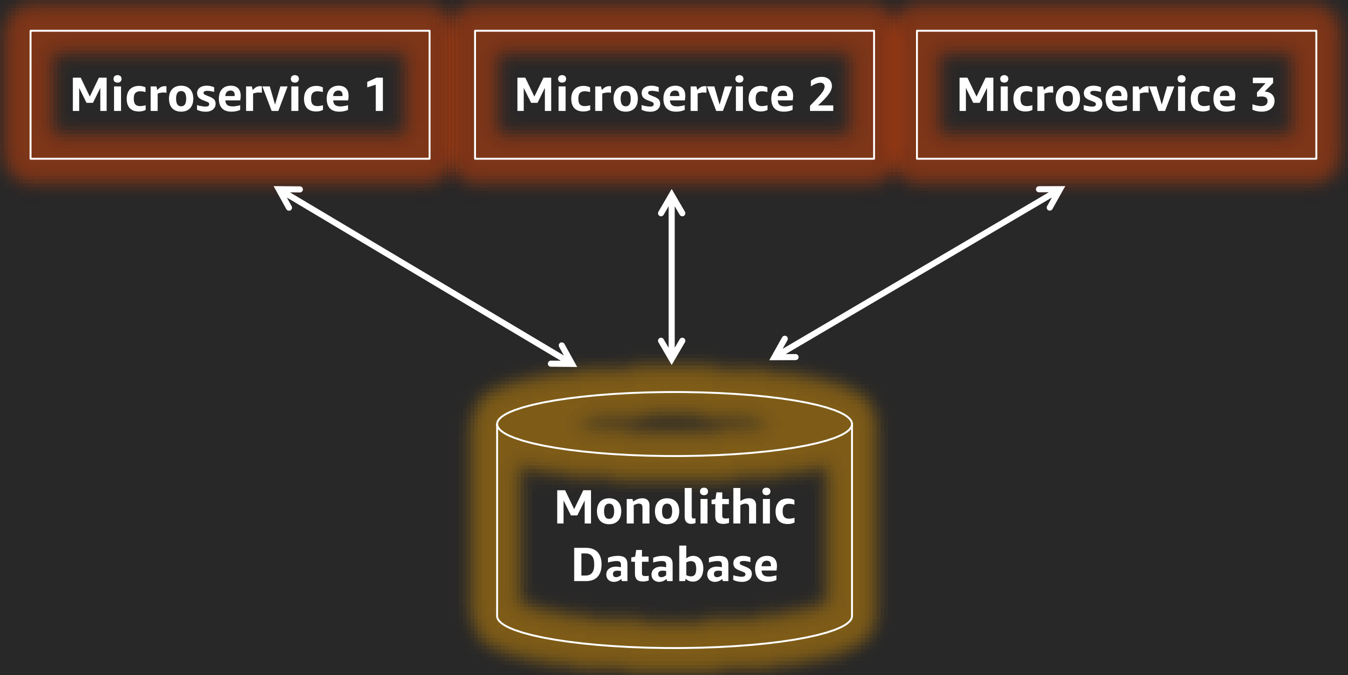
- The database in a traditional enterprise software environment is at the center of everything
- It is centralized, big, inflexible, and relational
- Multiple applications talk to the same big database
- Difficult to scale



Single monolithic database

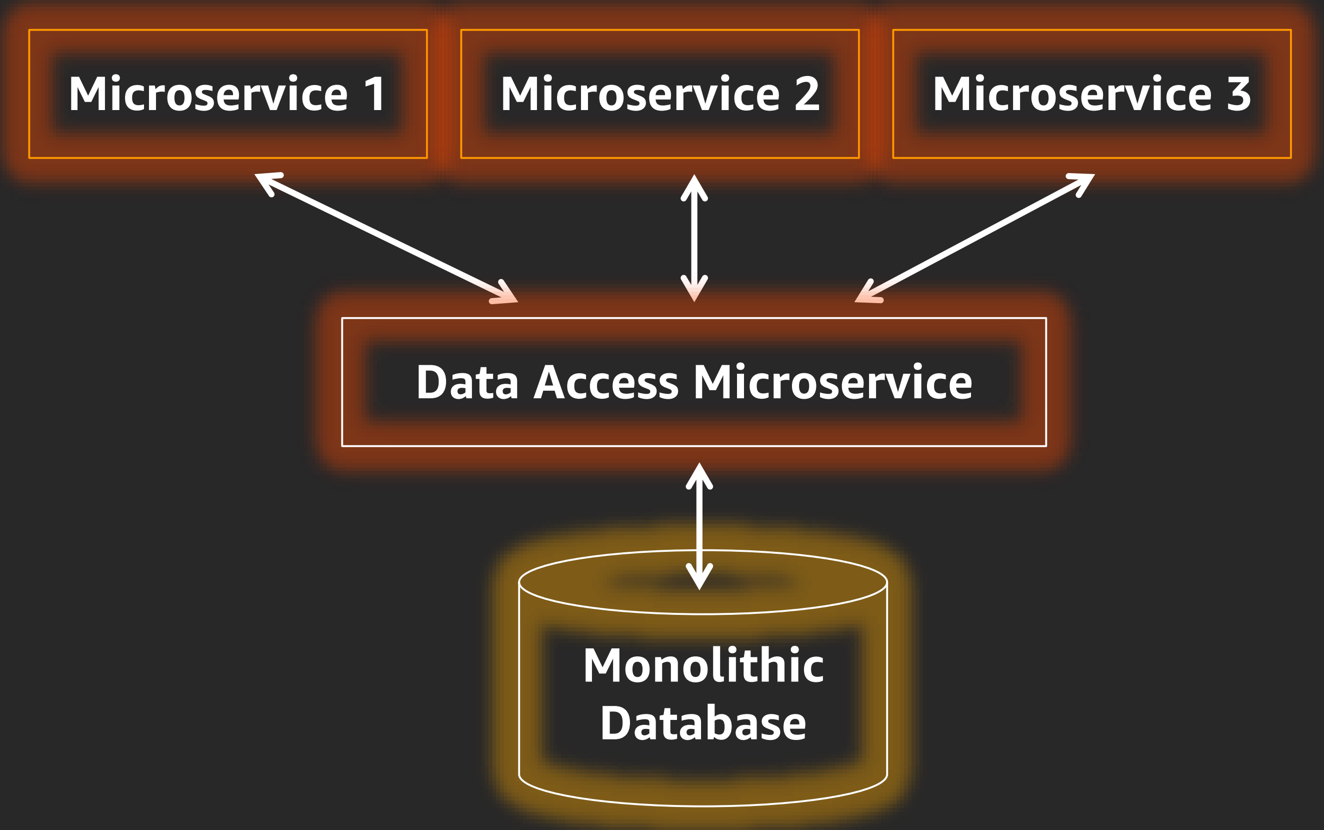
Multiple microservices are dependent on the same database

- Introduces high coupling between microservices
- Introduces schema dependencies
- Bottlenecks and noisy neighbors



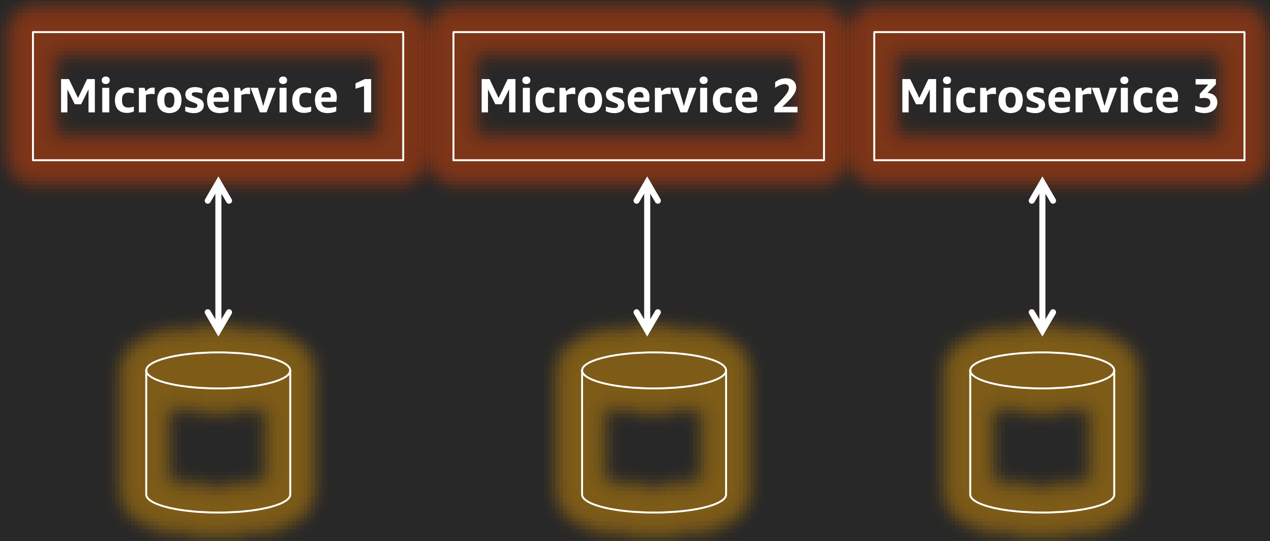
Centralized data service

- Still using a centralized database, but all access is via a central microservice API
- Not scalable if you have hundreds or thousands of microservices all requiring access to some part of the data



Database per microservice

- Each microservice has its own data layer
- Small autonomous services that work together
- Teams for each microservice choose the database that best suits the service



Best practices for companies looking to break down monolithic architectures

Monoliths to microservices – it's not just about tech

Cultural philosophy

Practices

Tools

- Tearing down barriers
    - Between teams
    - Mid-process
- Assign a custodian team that is in charge of the development, maintenance, and operation of microservices
- More ownership and cross-functional skills in the team
- Adoption of a mindset that security is everyone's job
- Increase visibility to the big picture and the results of work being done

Think about DevOps best practices

Cultural philosophy

Practices

Tools

- Clearly defined APIs
- Continuous integration
  - Application testing/QA applied throughout the development
- Continuous delivery
  - Automated deployment capabilities of code across environments
- Infrastructure as code
- Break down complicated monolithic applications in to smaller ones

Choose the right tool for the job and automate

Cultural philosophy

Practices

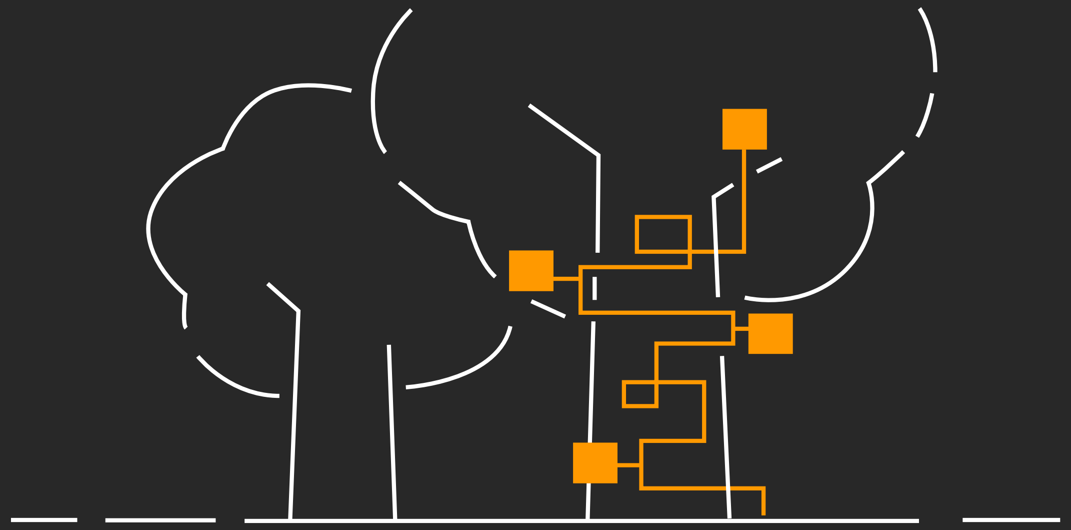
Tools

- Automated development pipeline tooling
  - Application testing frameworks
  - Code review/feedback tools
- Application management tools
- Configuration management tools
- Consistent infrastructure measurement tools
  - Metrics
  - Logging
  - Monitoring
- Security analysis and management tools

Don’t boil the ocean

Break down the monolith gradually, rather than taking a big bang approach

- Build a new system around the edges of the old, letting it grow slowly as needed until the old system is strangled
- Gradually create events and APIs for various components on of the legacy application
- What microservices will benefit most from speed of innovation?


THE STRANGLER PATTERN

Thank you!

Stephanie Chiao

https://www.linkedin.com/in/stephanie-chiao-1a622576/