



PUBLIC SECTOR SYMPOSIUM

BRUSSELS | MARCH 28, 2023

BAT304

Refactoring enterprise applications for cloud-native architecture

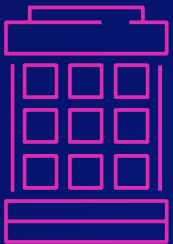
Dennis Kieselhorst
Principal Solutions Architect
Amazon Web Services



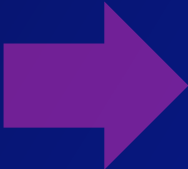
Organizations are moving legacy applications and data to the cloud

Goals

Windows
SAP
VMware
Databases
Mainframe



Legacy



Accelerate business transformation



Increase agility and innovate quickly



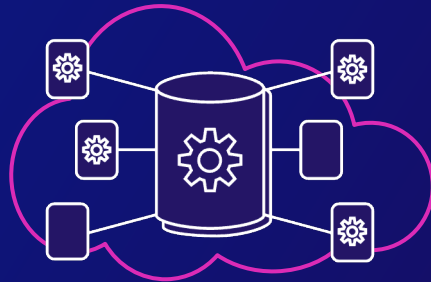
Reduce operating costs



Access cloud scale, performance, and operations



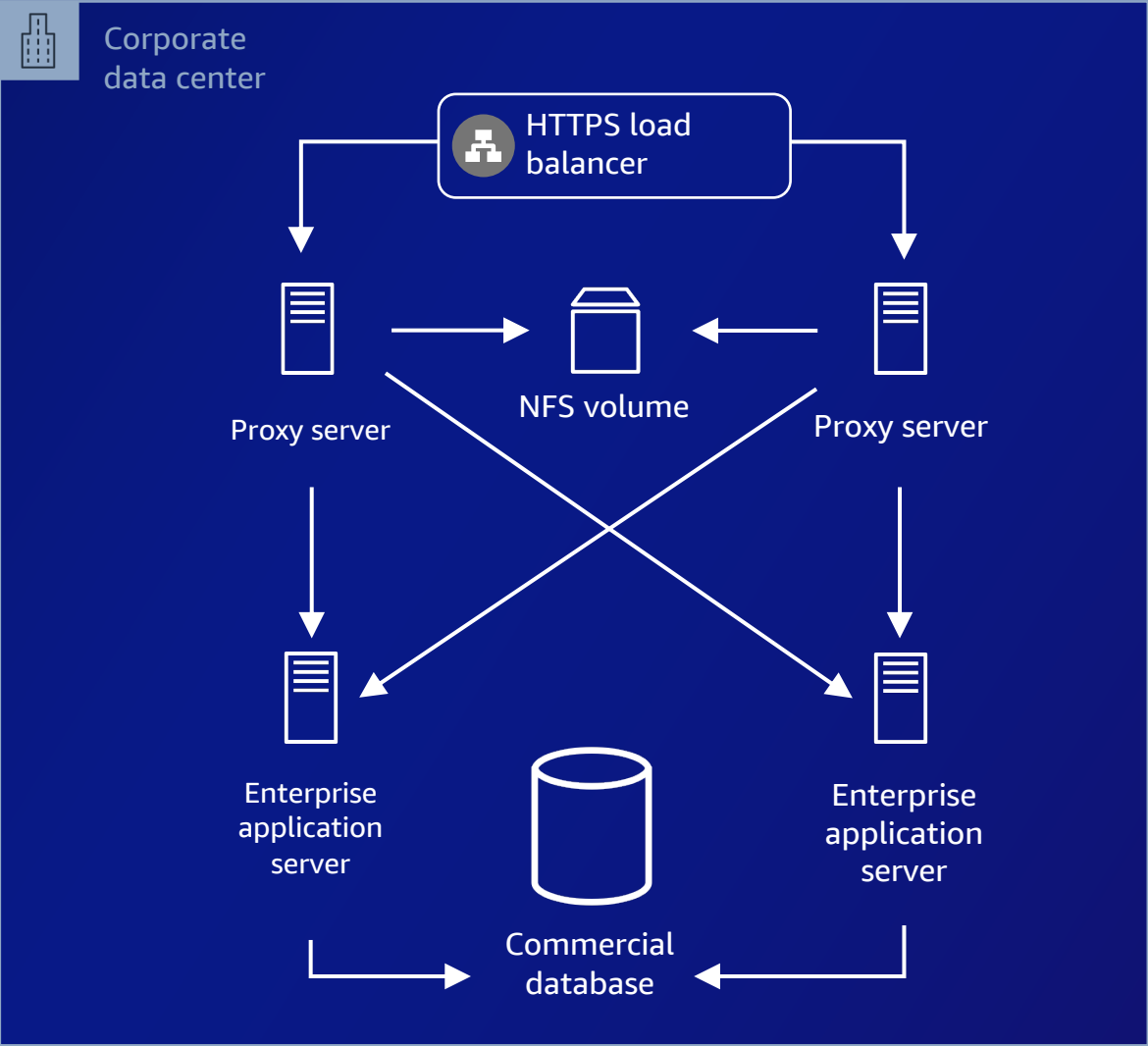
Improve security and compliance



Modern

>50% of enterprise workloads and data are expected to be in the cloud within 12 months

A sample application: How would you start?



Challenge: Refactoring monolithic applications



Business

- Slow product innovation
- Decreased time to market
- Drive unclear team ownership boundaries
- Changes have unforeseen side effects



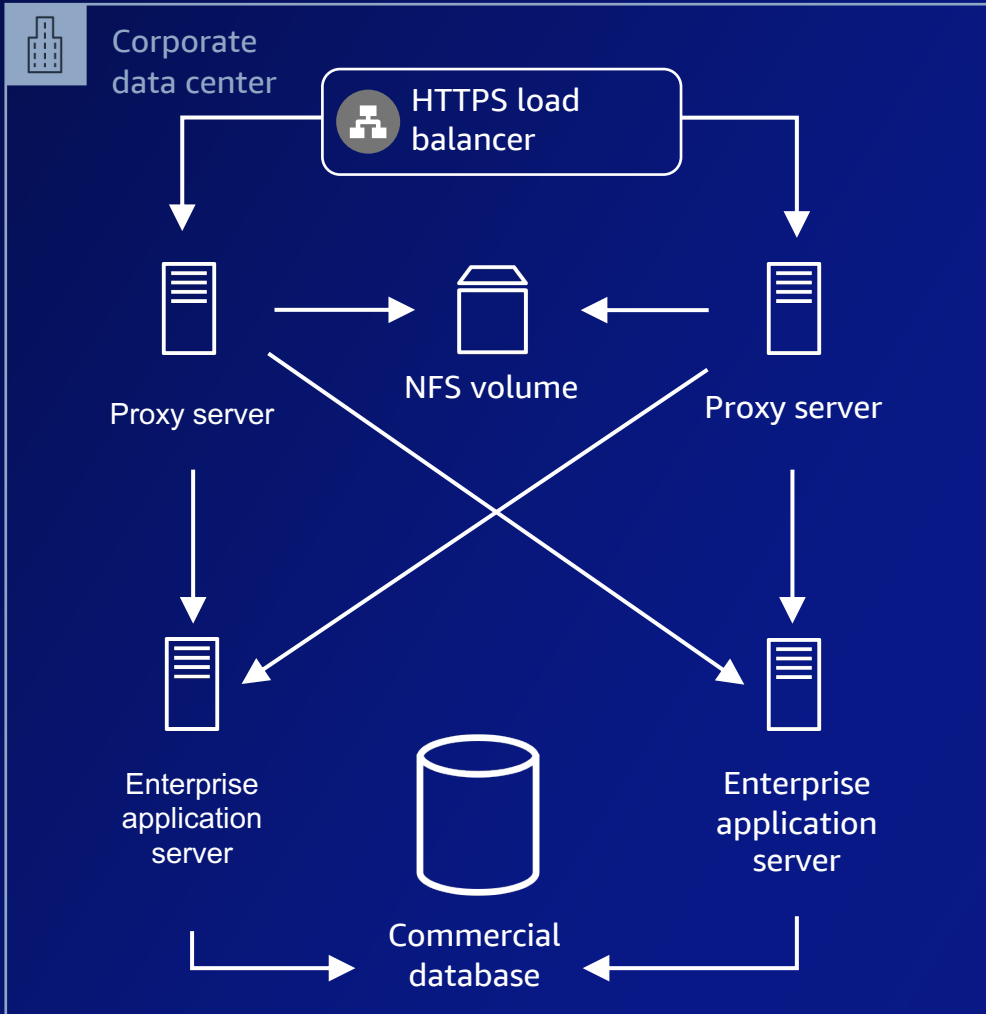
Technical

- High coupling and low cohesion
- Difficult and infrequent deployments
- Apps are brittle and difficult to understand
- Rapid experimentation has the potential of high-scale collateral impact

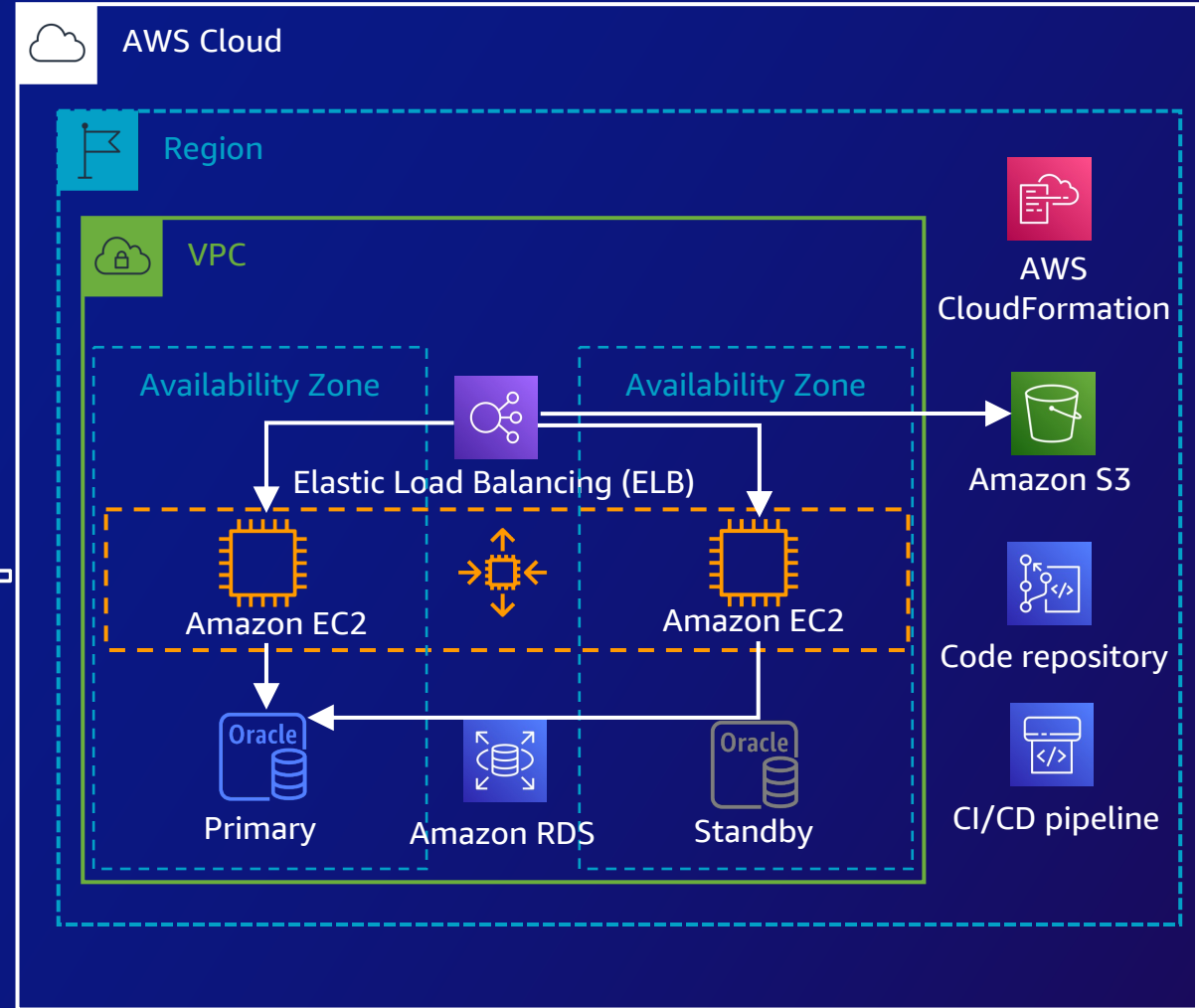
7 Rs

- ✓ Refactor
 - ✓ Replatform
 - ✓ Repurchase
 - ✓ Rehost
 - ✓ Relocate
 - ✓ Retain
 - ✓ Retire
- Business value ↑

First step: Lift and shift (rehost)



Monolith on premises



Monolith on AWS

Minimal viable refactoring

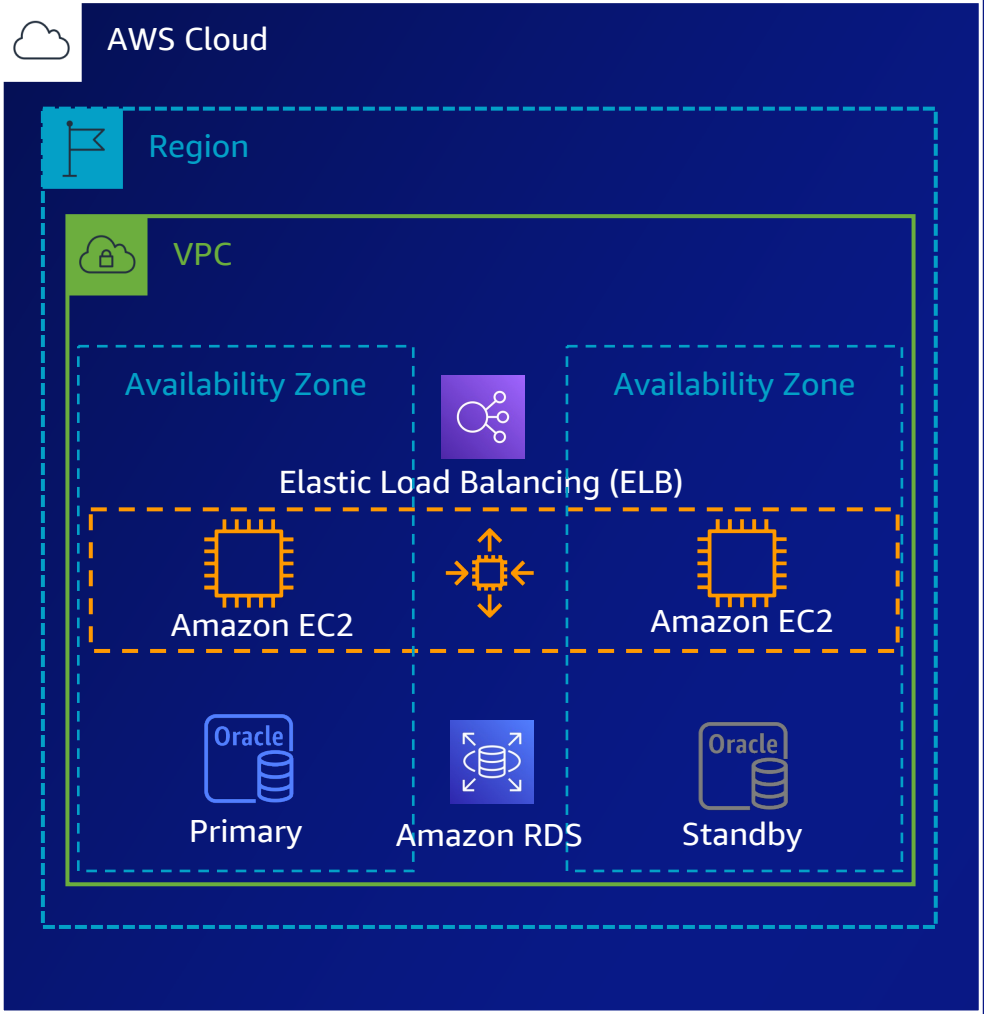
- Update the operating system
- Upgrade frameworks
- Improve security
- Clean up access rights
- Explore database options

Related blog post:

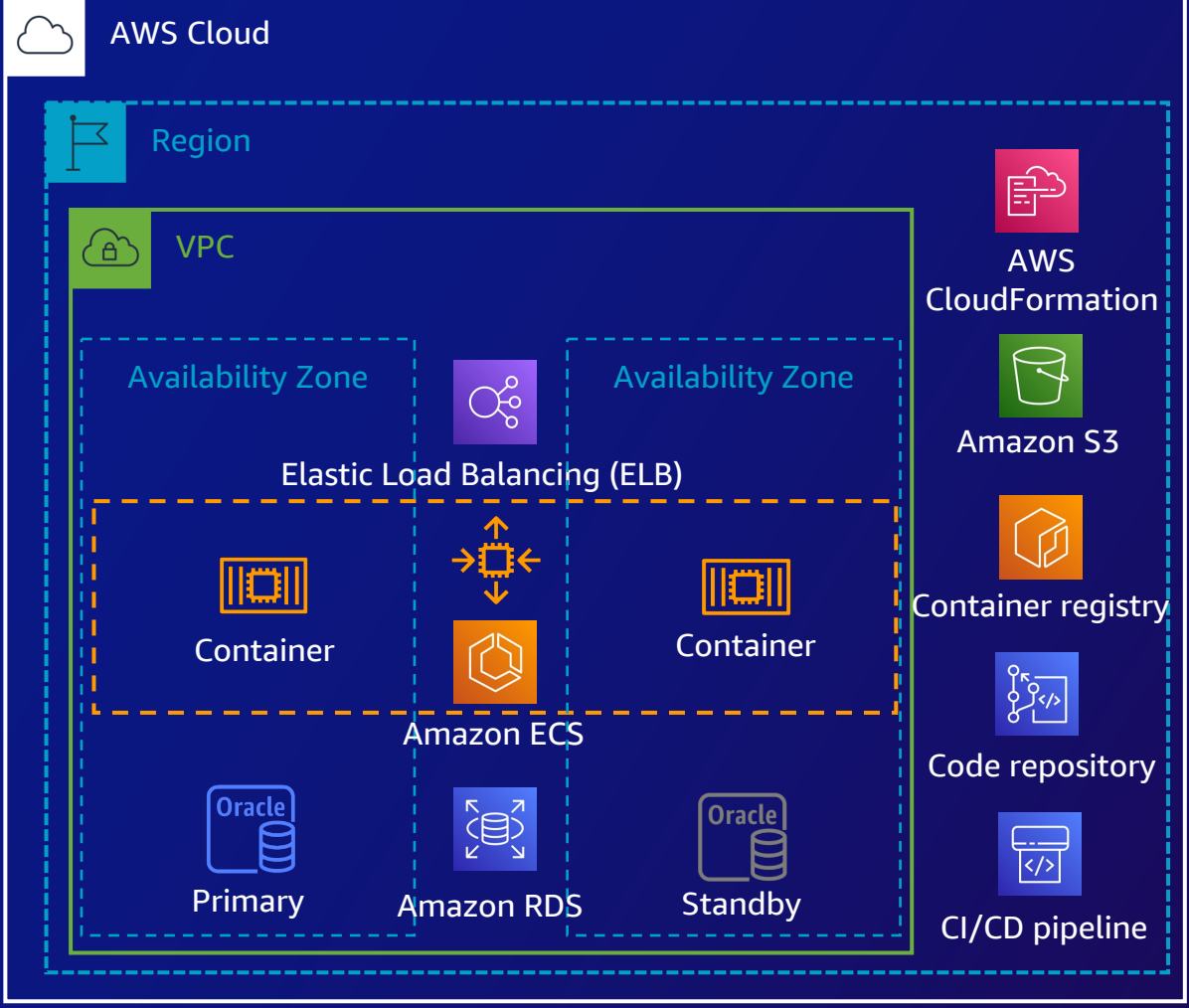
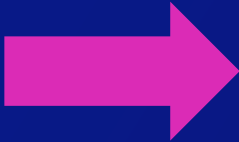
<https://a.co/4DYzYdd>



Containerize (replatform)



Monolith on VMs



Containerized monolith



AWS App2Container



Discover and analyze

Create application inventory and analyze runtime dependencies

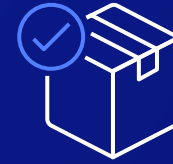
1



Extract and containerize

Extract application with dependencies and create container image

2



Create deployment artifacts

Generate the ECS tasks or Kubernetes pod definitions and create CI/CD pipelines

3



Deploy to AWS and launch

Store image in Amazon ECR and deploy to Amazon ECS or Amazon EKS

4

Best practices containerization

- Make container images complete and static
- Maintain fast container launch times by keeping container images as small as possible
- Only run a single application (server) process with a container image
- Handle SIGTERM within the application (server)
- Configure containerized applications to write logs to stdout and stderr
- Version container images using tags



Traditional three-tier application architecture



Web servers

Presentation layers



Application servers

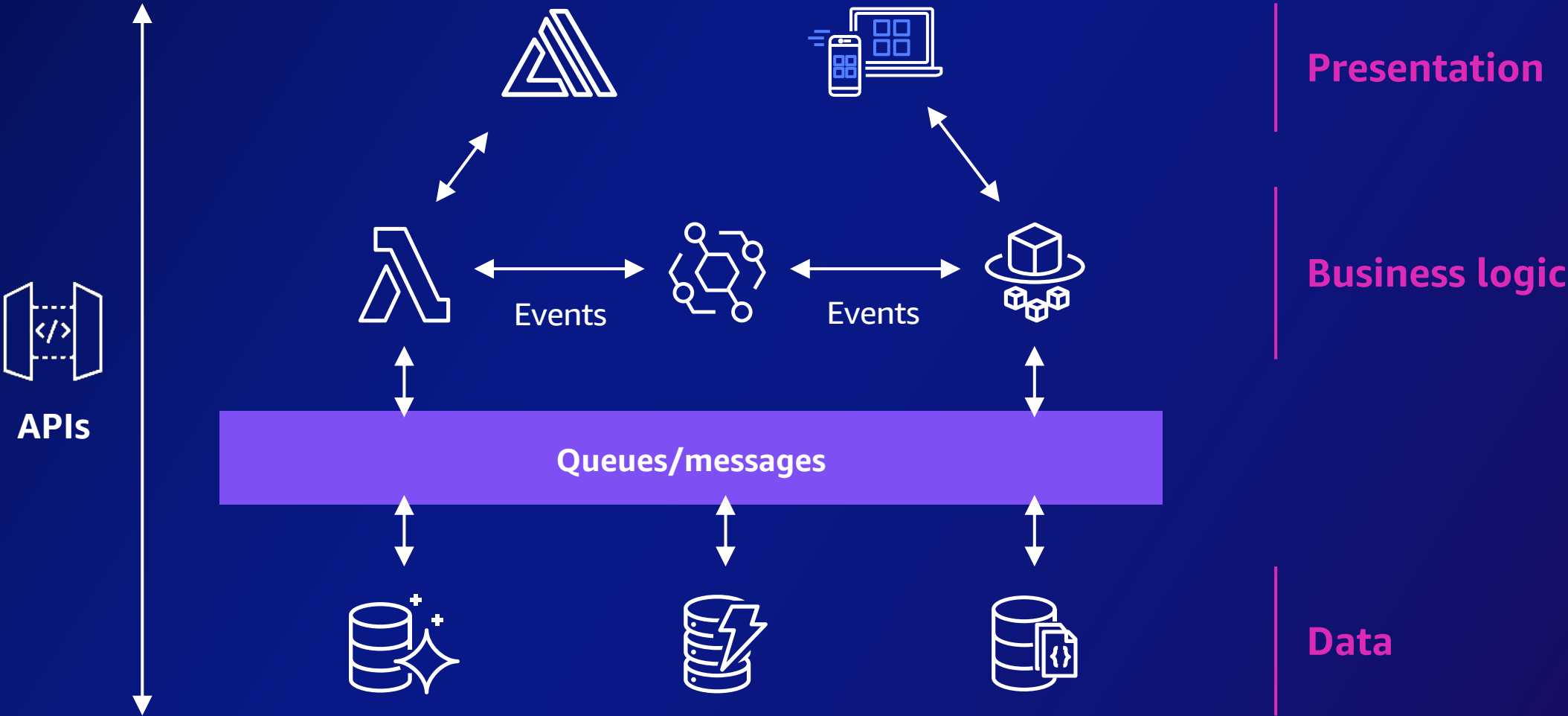
Business logic



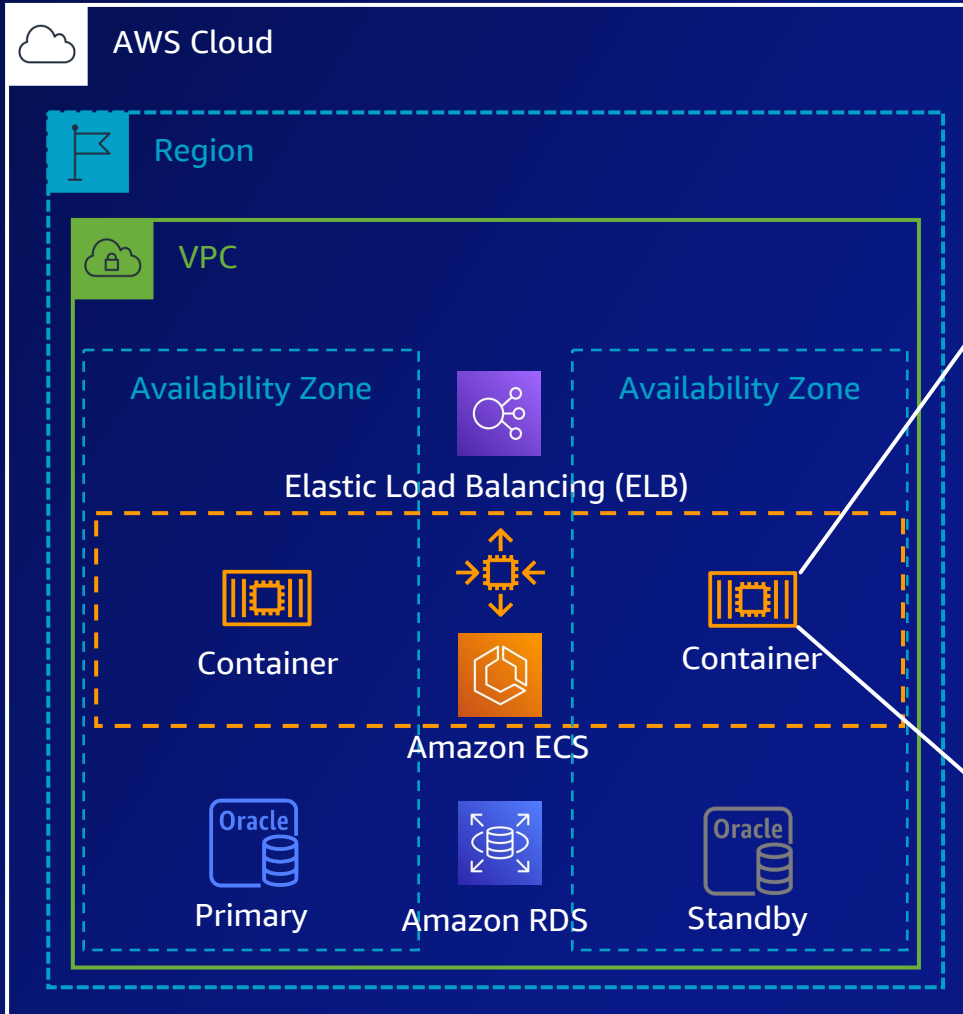
Database servers

Data layer

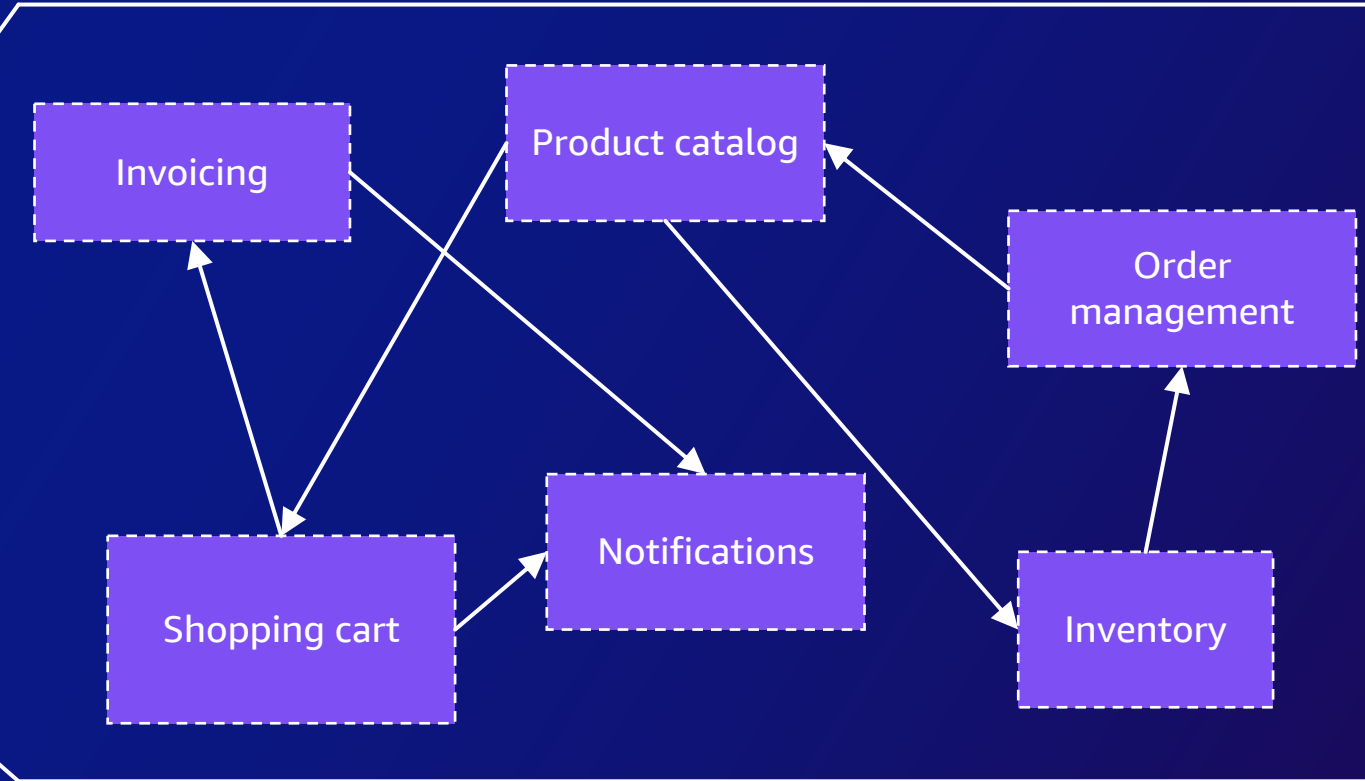
A modern three-tier application architecture



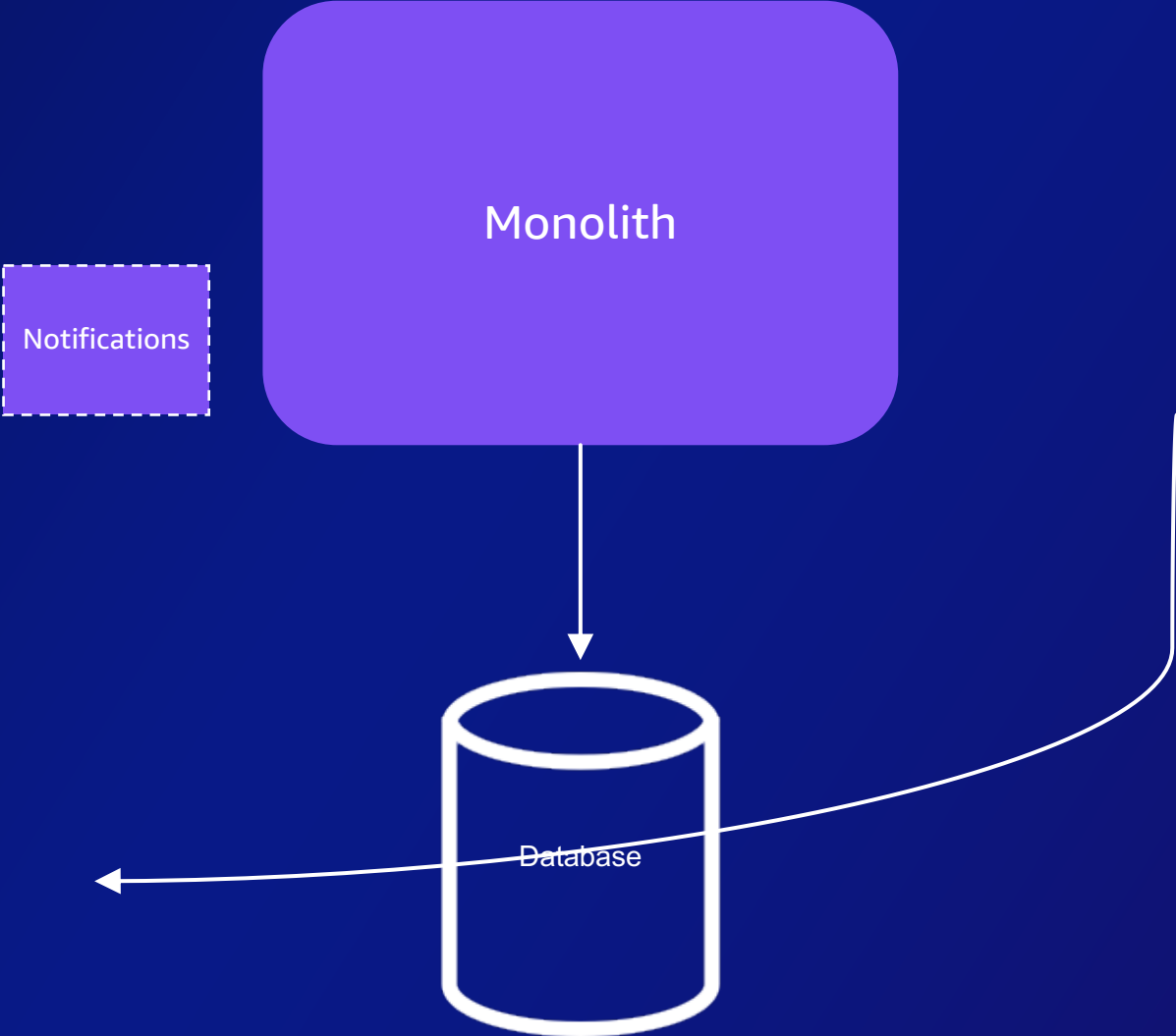
How to break the monolith



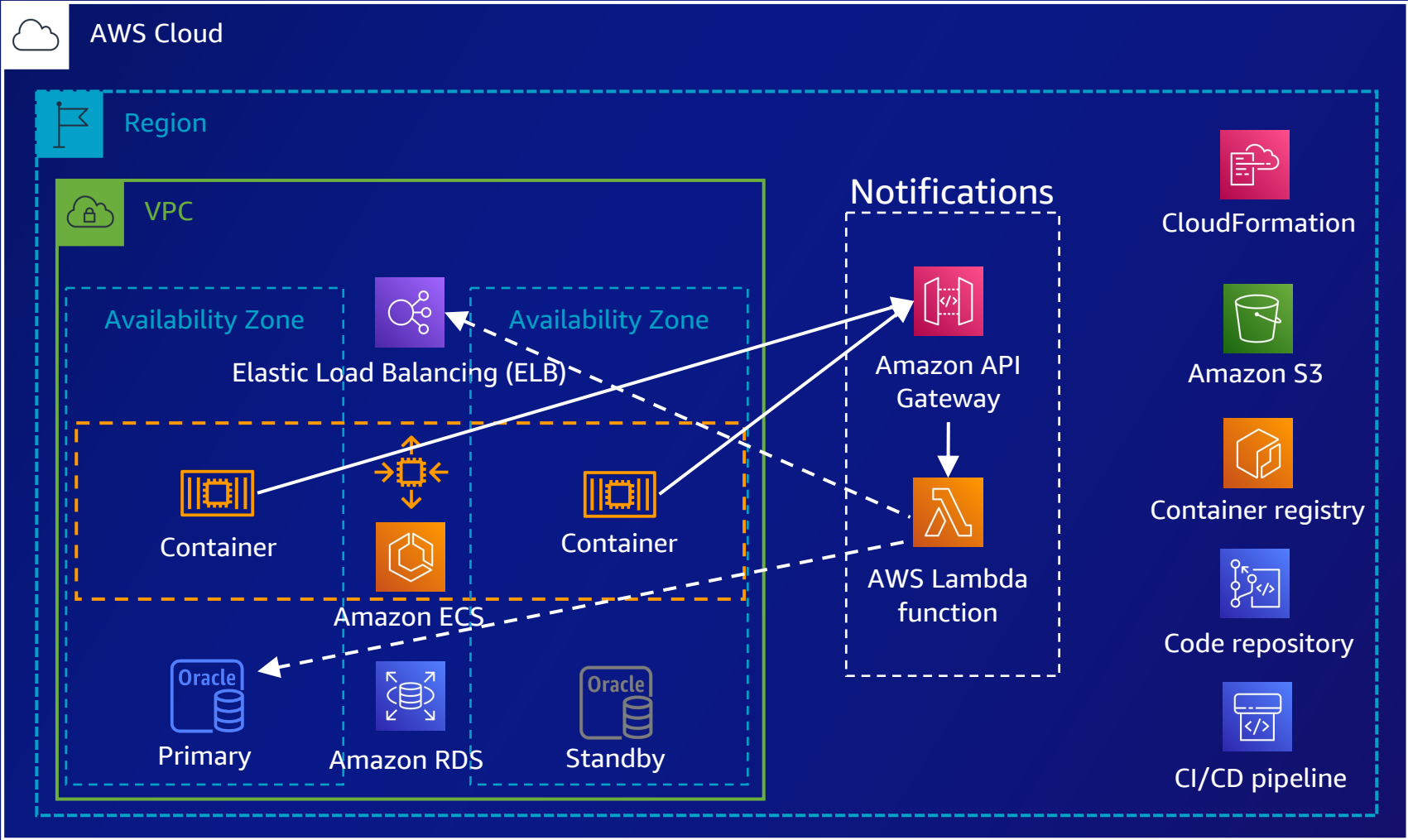
Containerized monolith



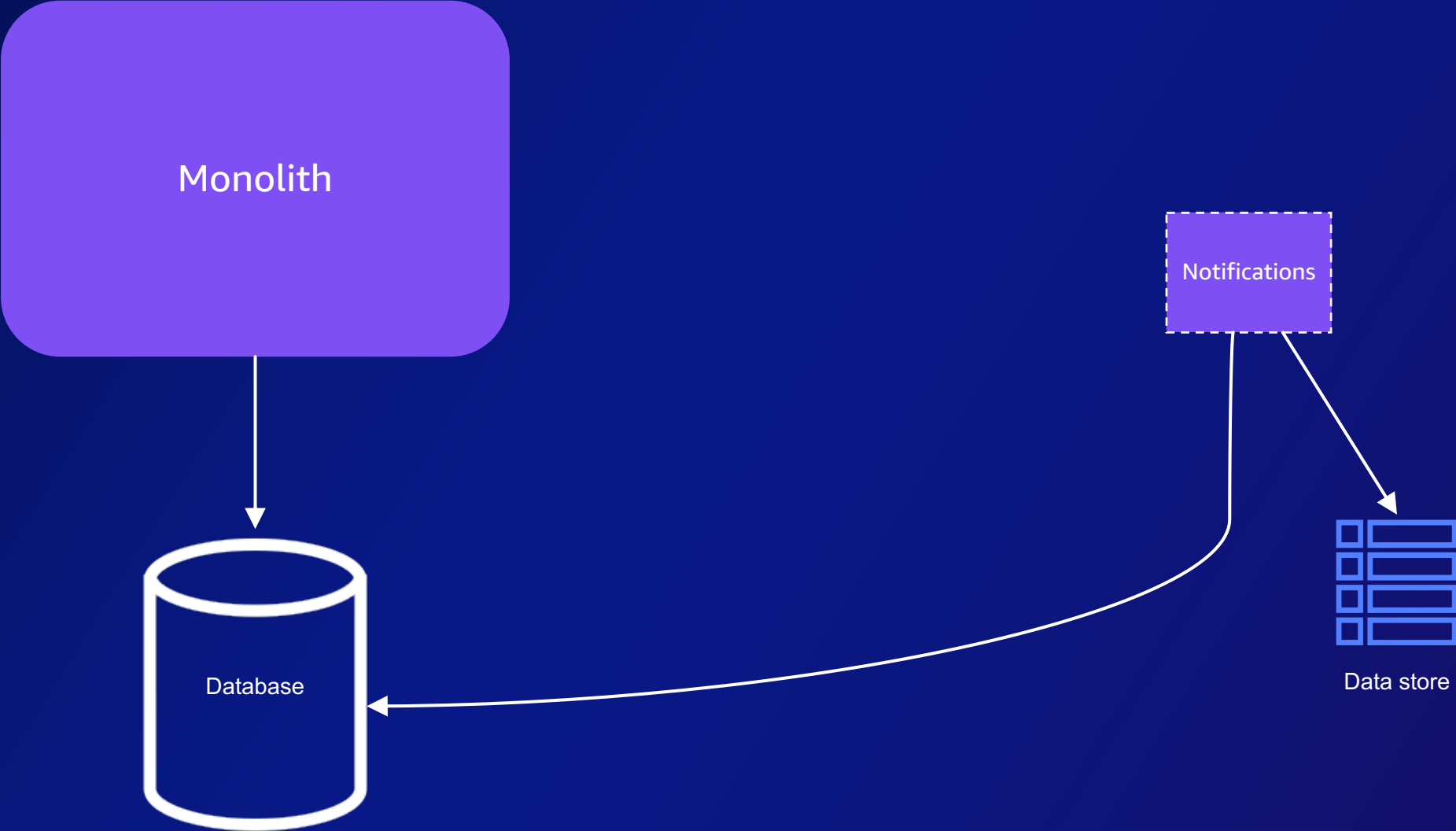
Breaking up the monolith














Create a new service for one domain (refactor)



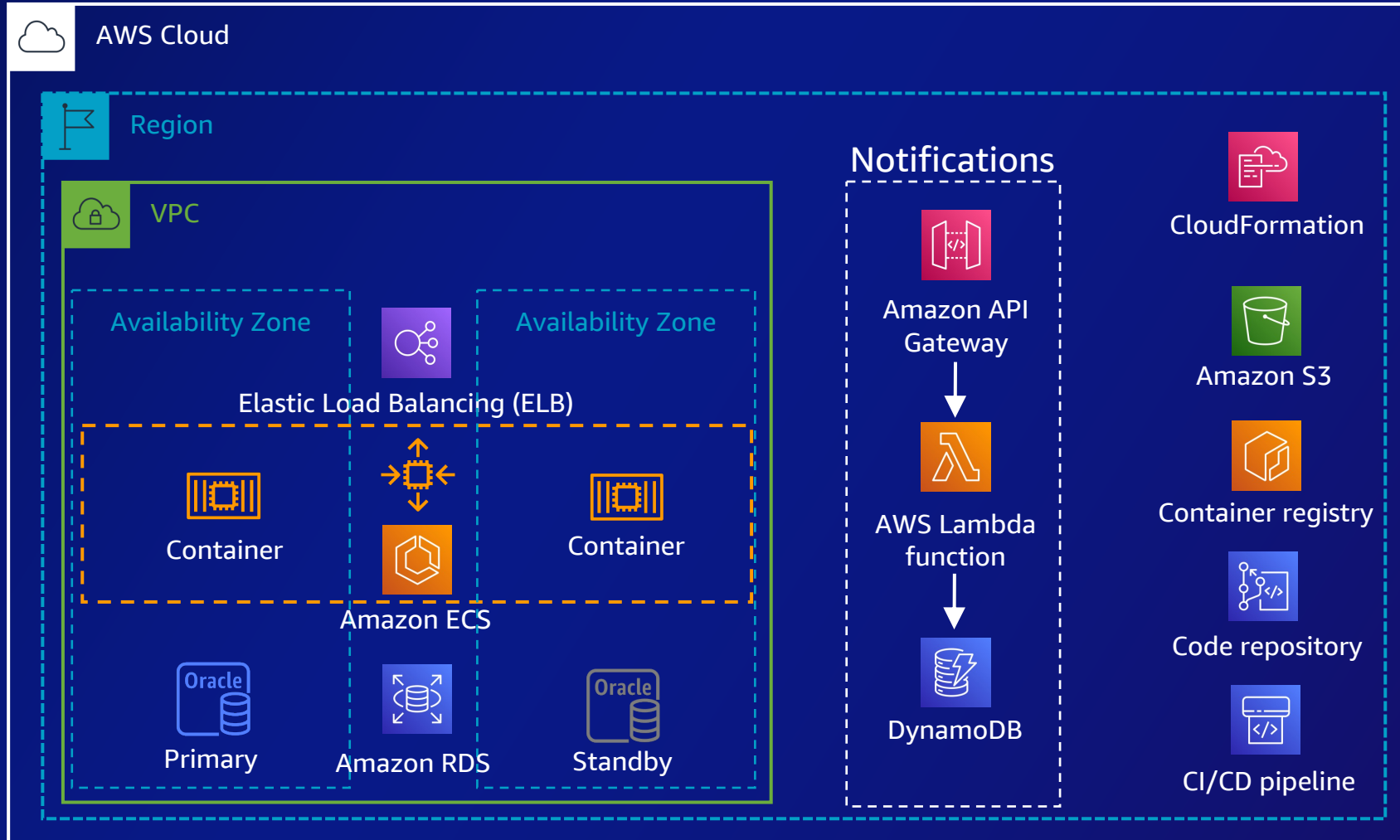
Breaking up the monolith



Best practice: Use purpose-built databases

	Relational	Key value	Document	In memory	Graph	Time series	Ledger	Wide column
	Referential integrity, ACID transactions, schema-on-write	High-throughput, low-latency reads and writes; endless scale	Store documents and quickly access querying on any attribute	Query by key with microsecond latency	Quickly and easily create and navigate relationships between data	Collect, store, and process data sequenced by time	Complete, immutable, and verifiable history of all changes to application data	Scalable, highly available, and managed Apache Cassandra-compatible service
AWS service(s)	 Amazon RDS   Amazon Aurora Amazon Redshift	 DynamoDB	 Amazon DocumentDB	 Amazon ElastiCache  Amazon MemoryDB for Redis	 Amazon Neptune	 Amazon Timestream	 Amazon QLDB	 Amazon Keyspaces for Apache Cassandra
Common use cases	Lift-and-shift, ERP, CRM, finance	Real-time bidding, shopping cart, social, product catalog, customer preferences	Content management, personalization, mobile	Leaderboards, real-time analytics, caching	Fraud detection, social networking, recommendation engine	IoT applications, event tracking	Systems of record, supply chain, healthcare, registrations, financial	Build low-latency applications, leverage open source, migrate Cassandra to the cloud

Use a separate database



How to integrate services

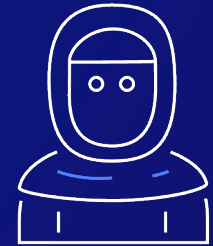
EVENT-DRIVEN ARCHITECTURE

- Event = a signal that a system's state has changed
- Services interact through events
- Events allow loose coupling

```
{  
  "detail-type": "OrderCreated",  
  "source": "com.orders",  
  "detail": {  
    "order_id": "1073459984",  
    "created_at": "2020-08-28T14:17:09-05:00",  
    "price": 24.62  
    "memberId": "USD",  
    ...  
  }  
}
```



OrderCreated



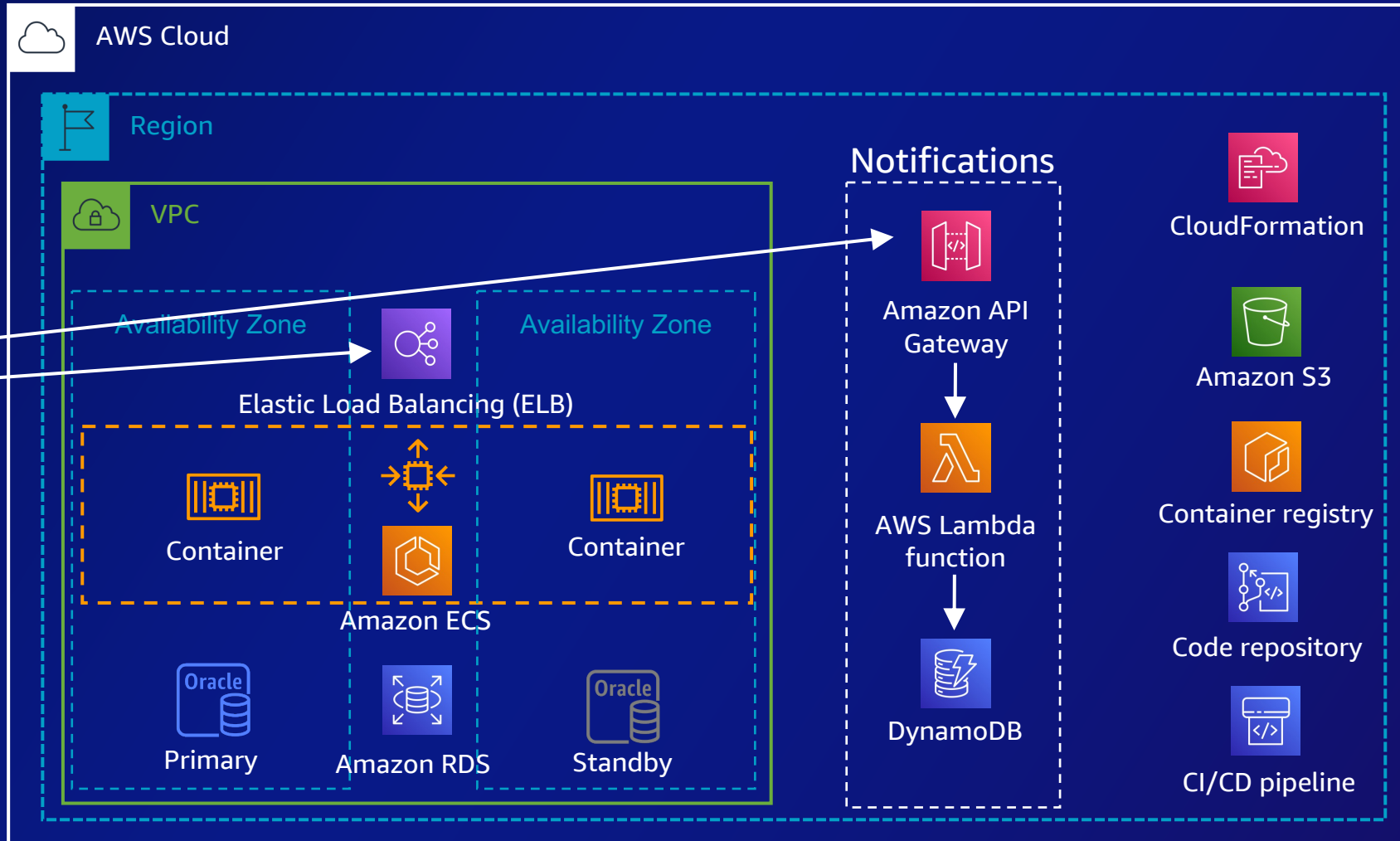
InvoiceGenerated



ShipmentReceived

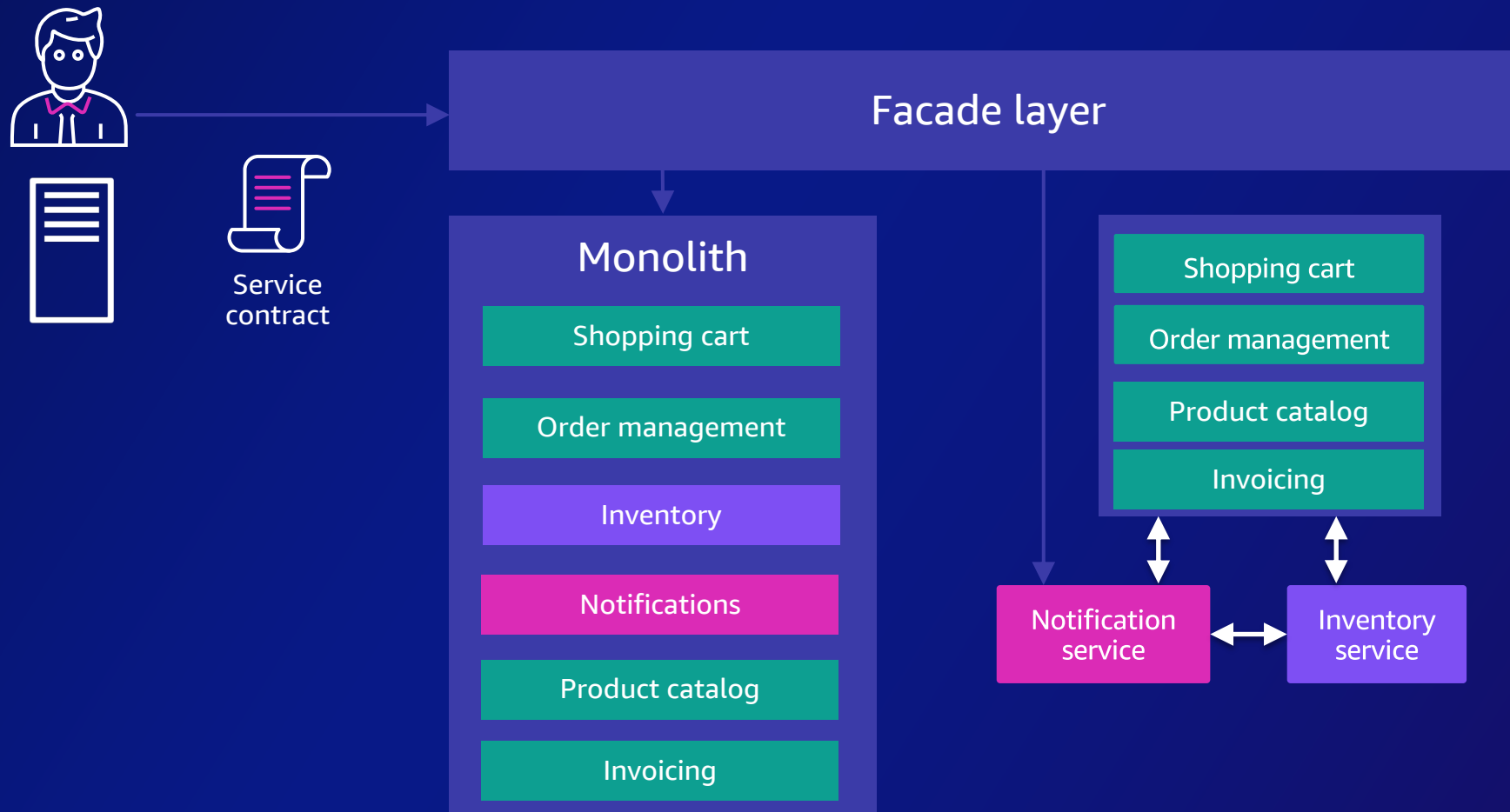


How to properly direct external consumers

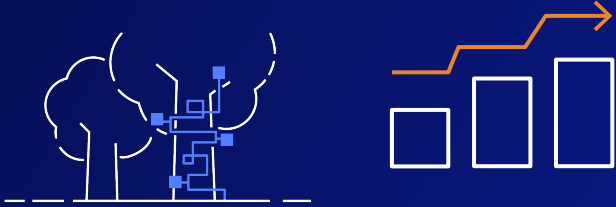


Facade pattern

Analogous to the proxy, a **facade** is an object that serves as a front-facing interface masking more complex underlying or structural code



Strangler Fig pattern



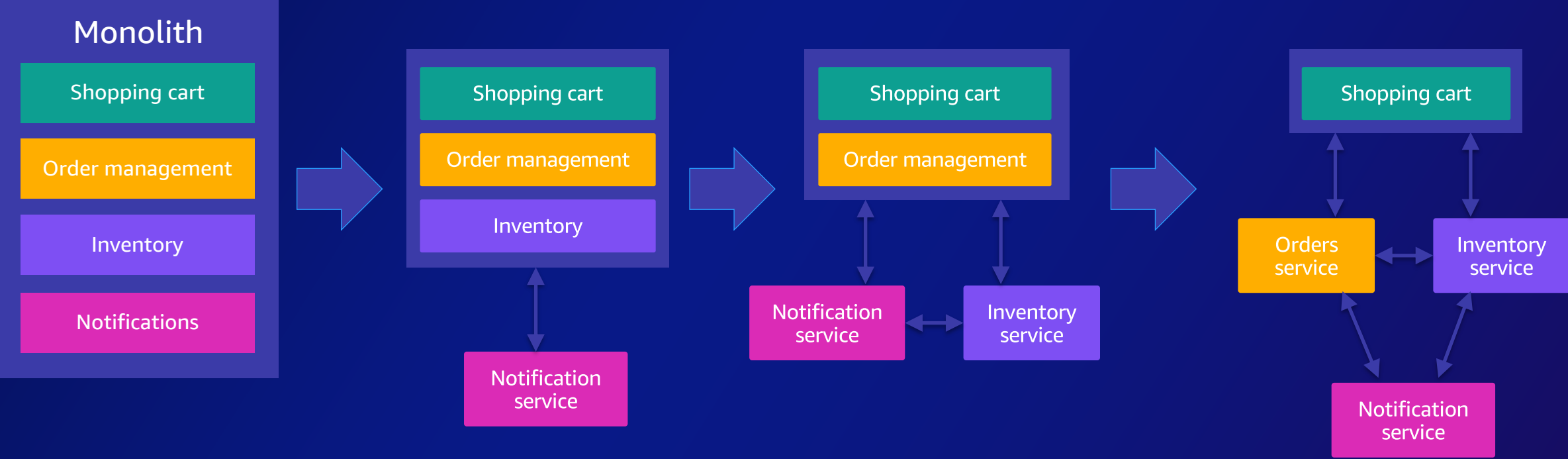
Refactor incrementally

Strangler Fig helps you achieve team independence and a modern app architecture

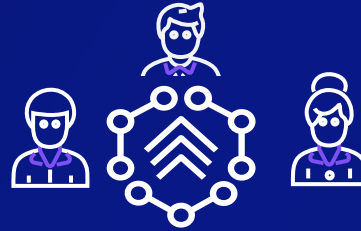
- Define the interface to the first piece to be refactored
- Design, implement, and test the new capability
- Redirect old to new
- Turn off the old code or service



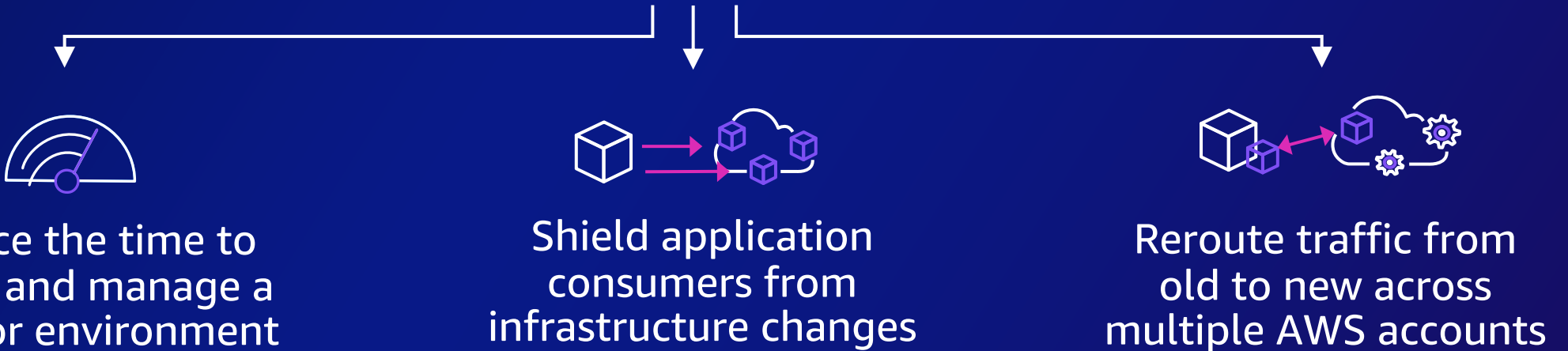
Strangler Fig pattern example



AWS Migration Hub Refactor Spaces

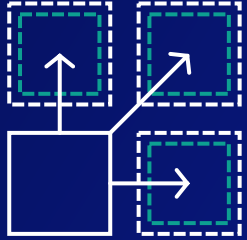


Migration Hub Refactor Spaces



Start **refactoring** applications in **days** instead of months

Best practices: Consider all key pillars of modernization



Technology and architecture

Independent business functions



People, process, and culture

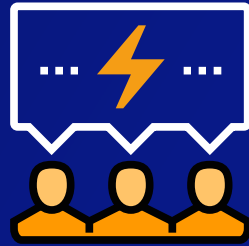
Organized for value



Ops and governance at scale

Automate, enable, and self service

Two-pizza team



Small,
decentralized
teams are
nimble

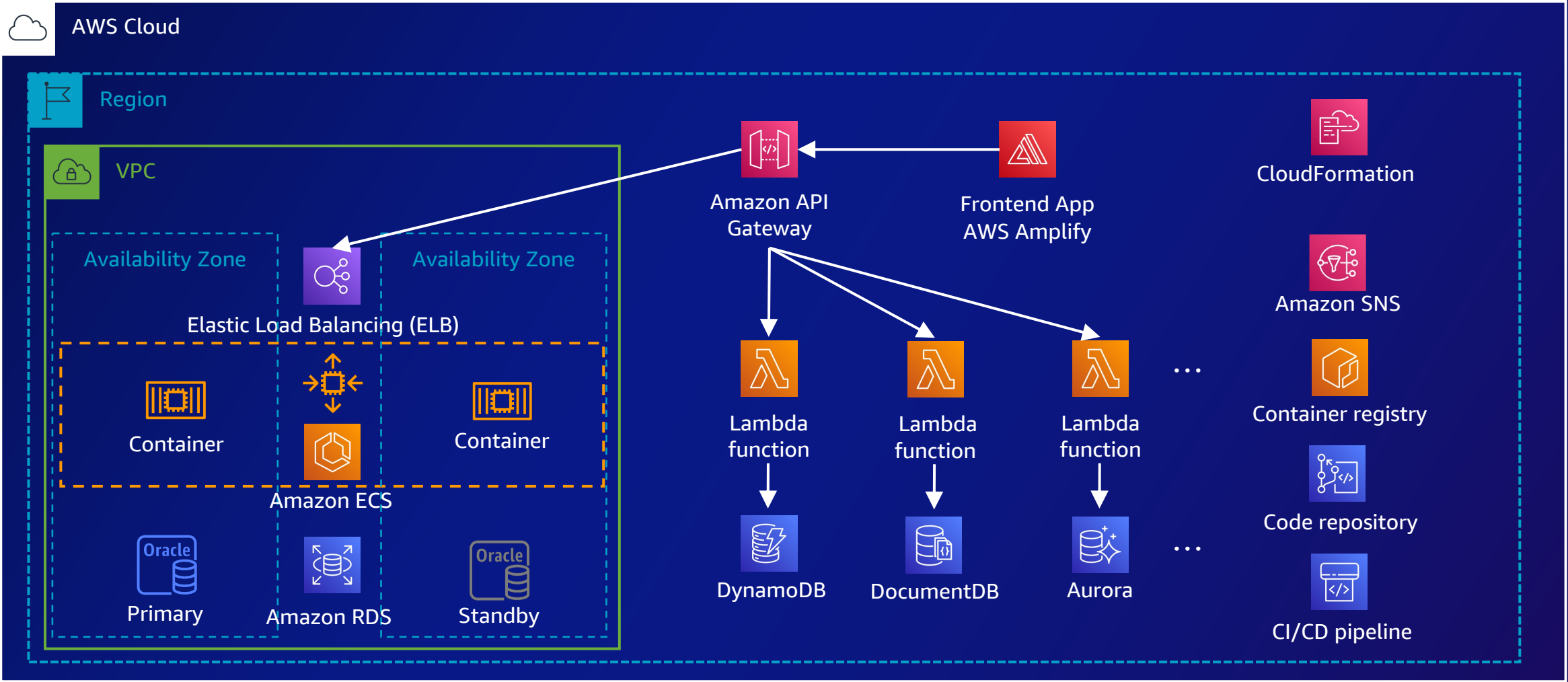


Own and
run what
you build

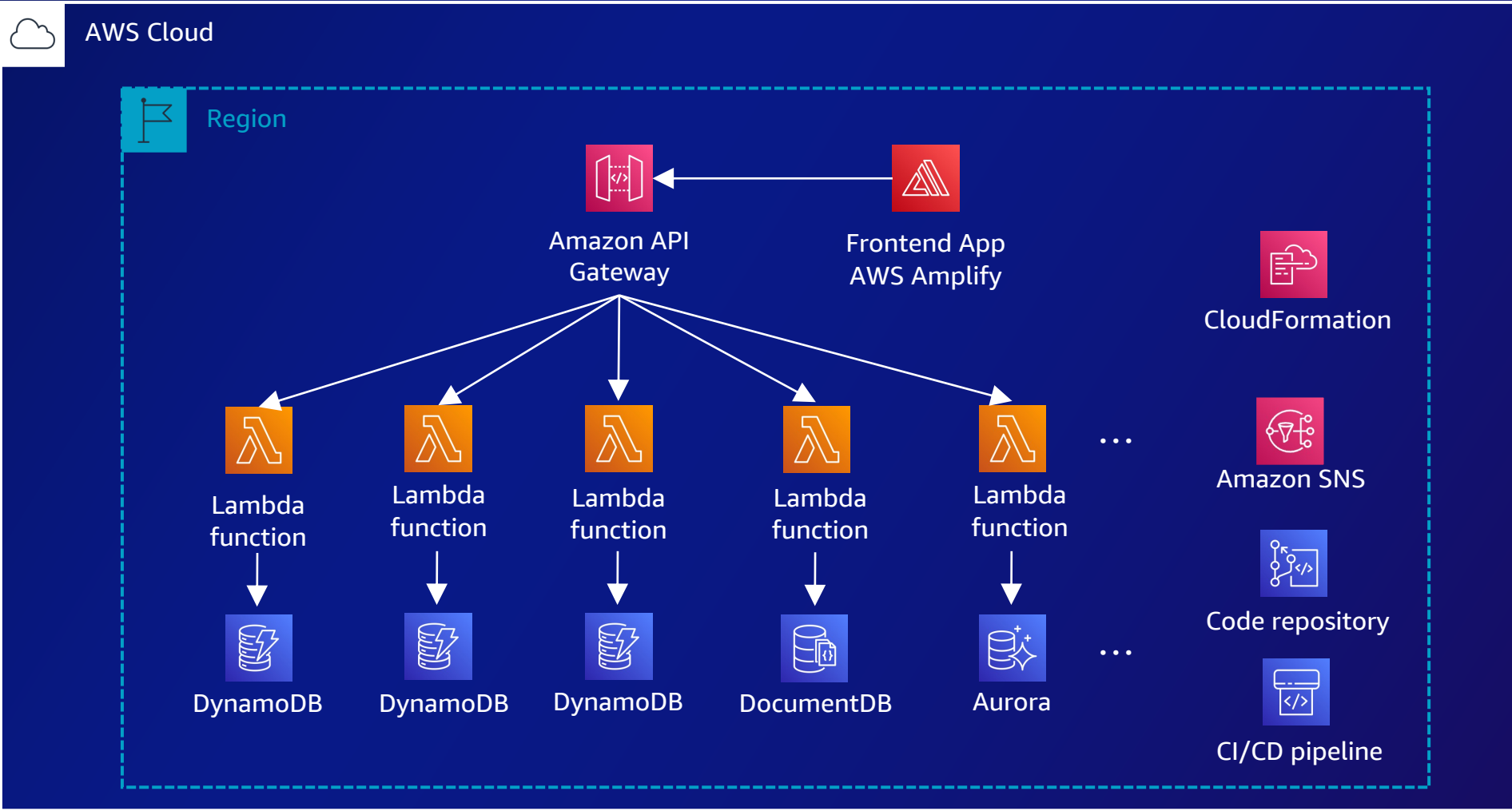


- Minimizes social constraints (Conway's law)
- Move from manual handoffs to as a service
- Automate all the things
- Simplify and decompose monoliths
- Two-pizza service teams

Add more services for different domains



Fully modernized application



Best practices: Microservices

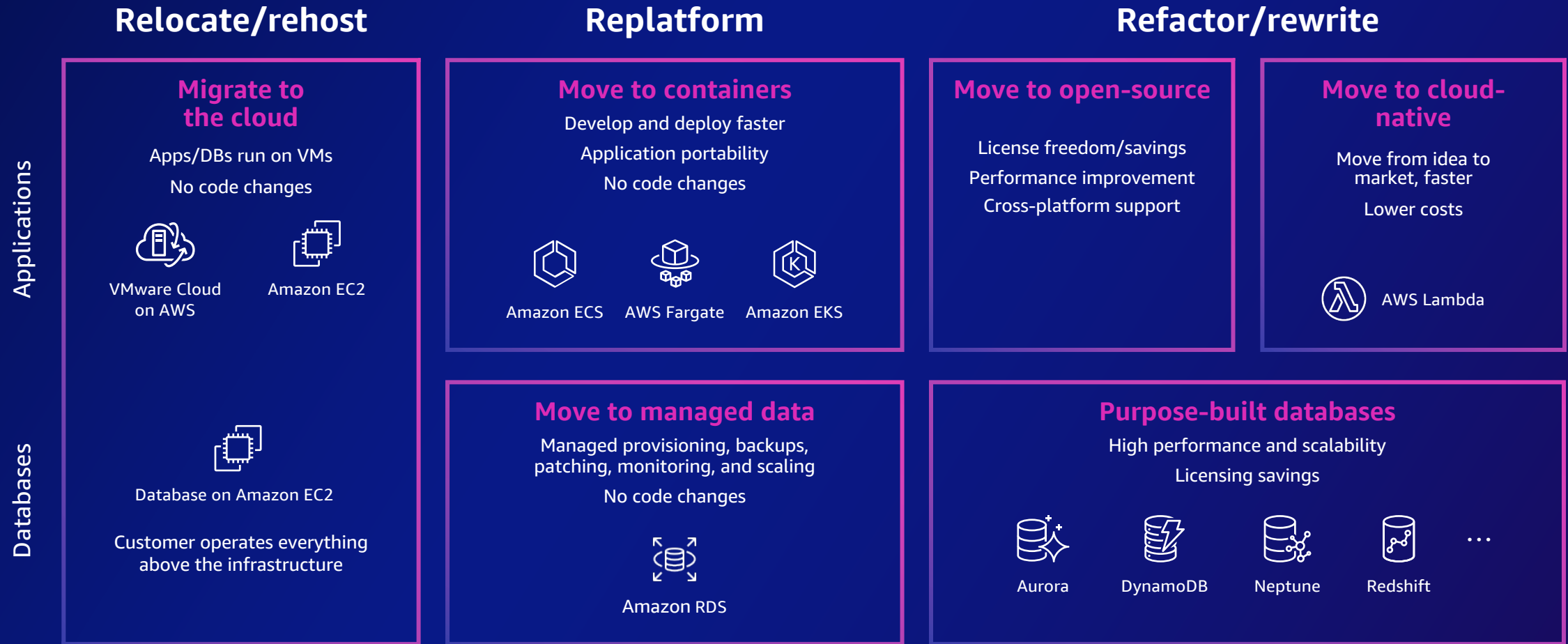
- Keep functions simple – use single-focused functions
- Each service should have its own data store
- Use asynchronous communication to achieve loose coupling
- Ensure backwards compatibility
- Establish two-pizza teams with ownership
- Automate all manual tasks



Summary and actions we recommend



Modernization pathways: Overview



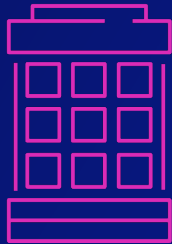
Actions we recommend

- Refactor when it provides business value
- Ensure executive sponsorship and leadership commitment
- Iterate incrementally; don't use a big bang approach
- Get in touch with AWS modernization experts and apply insights



AWS has the tools, programs, and services

Windows
SAP
VMware
Databases
Mainframe



Legacy



Modern



Purpose-built
AWS and partner
offerings



Incentive
programs



Third-party
offerings



Free
resources



AWS
Professional
Services



Certified partner
services

skillbuilder.aws 

Your time is now

Build in-demand cloud skills *your way*



Thank you!

Dennis Kieselhorst

✉ dkieselh@amazon.com

 [kieselhorst](#)



Please complete the session survey in the mobile app