Microservices Considerations and Design

Roland Guijt
MICROSOFT MVP, CONSULTANT, AUTHOR AND SPEAKER

@rolandguijt rolandguijt.com

Module Overview

The monolith

Up- and down-sides of microservices

Why ASP.NET Core?

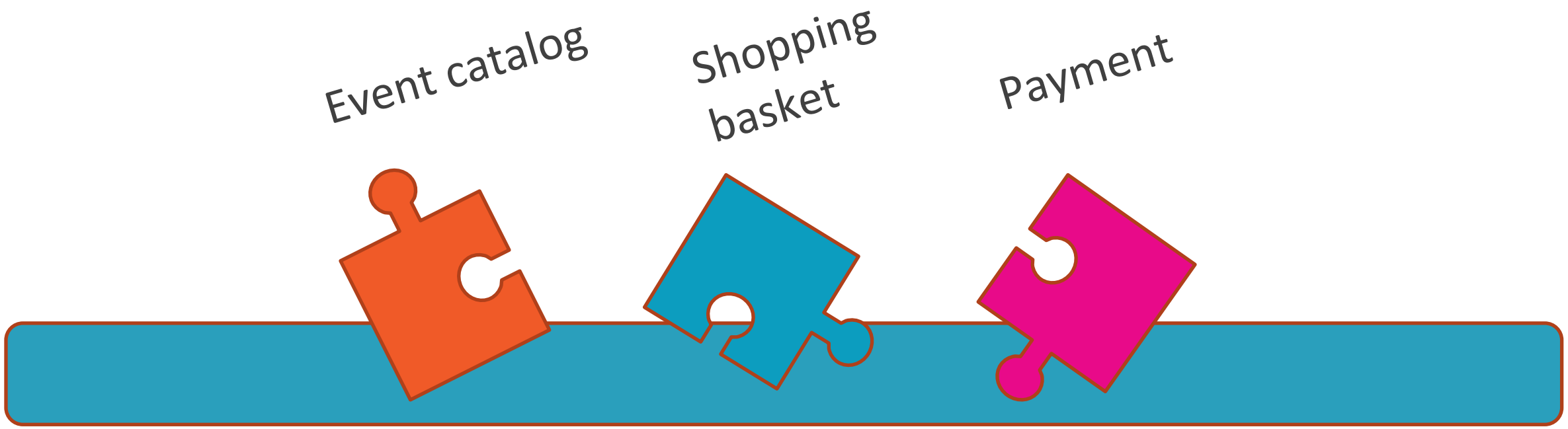
Deployment and monitoring

More options for service bus and transport

Benefits of Monoliths

- Easier to deploy
- Easier to test
- Well known
- Easier to comprehend
- No calls over the wire

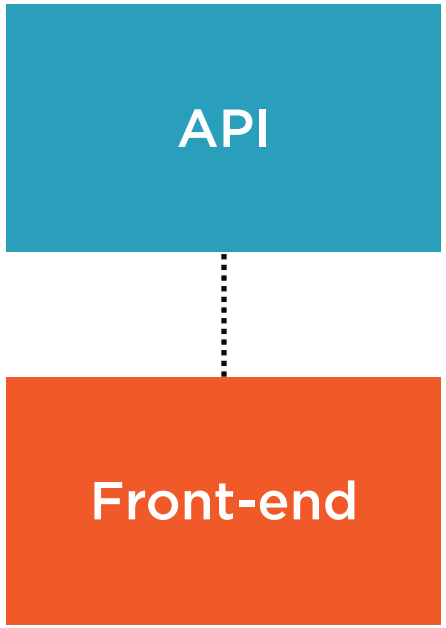
Monolithic Flow



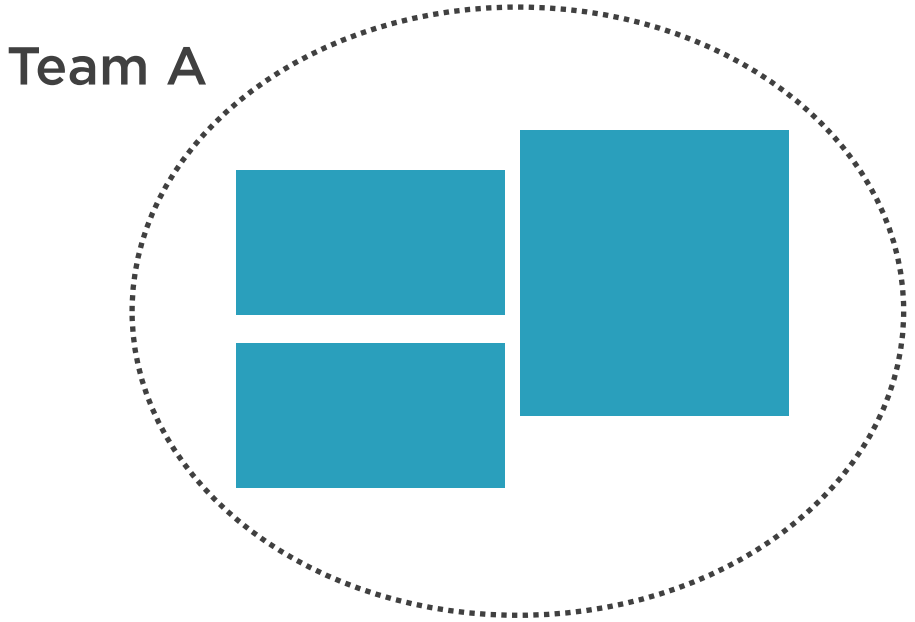
Downsides of Monoliths

- Hard to maintain modularity
- Bigger == more complex
- Tends to get bigger
- Scaling out
- Fault tolerance
- Updates to new technologies
- Multiple teams

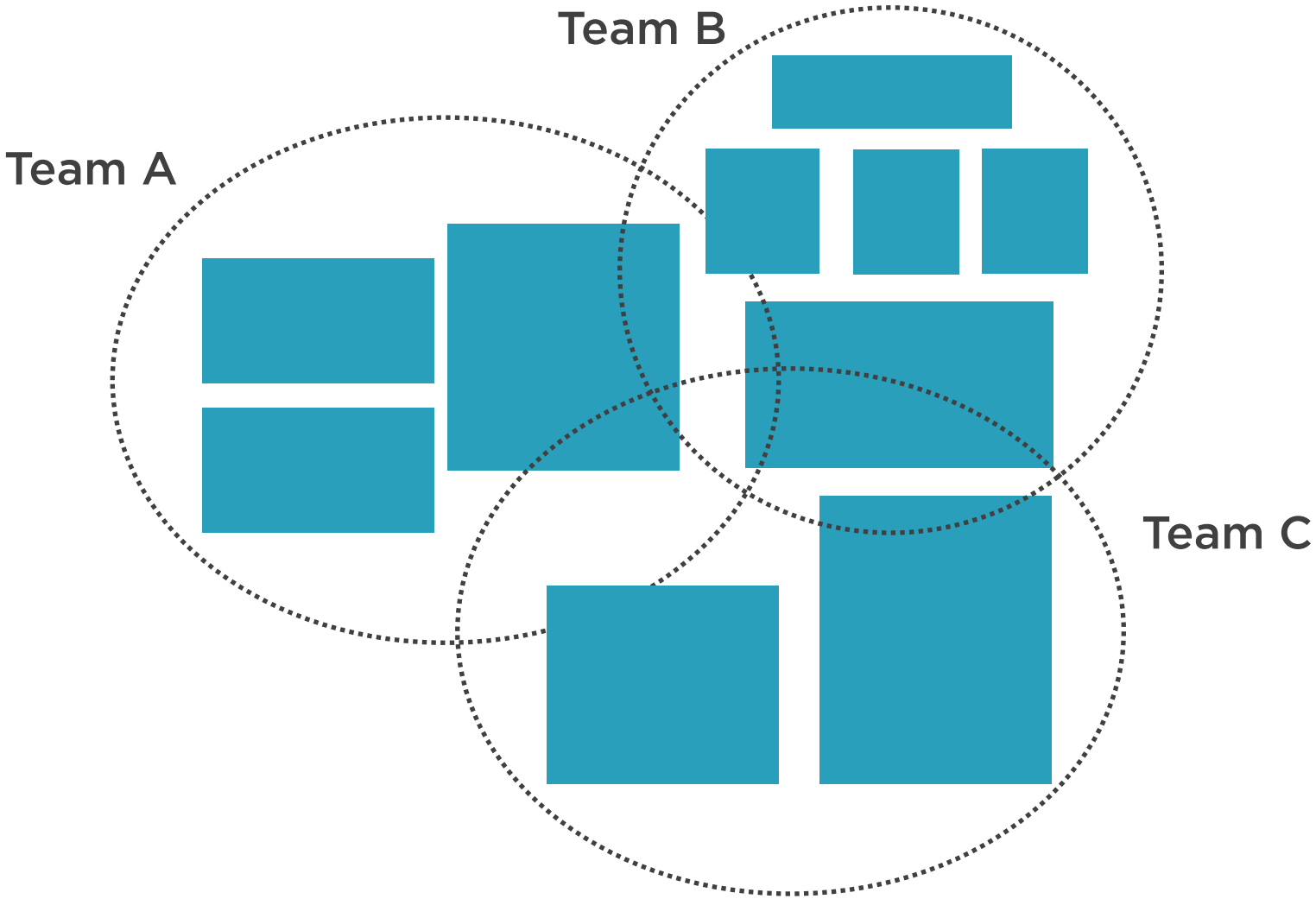
Distributed Monolith



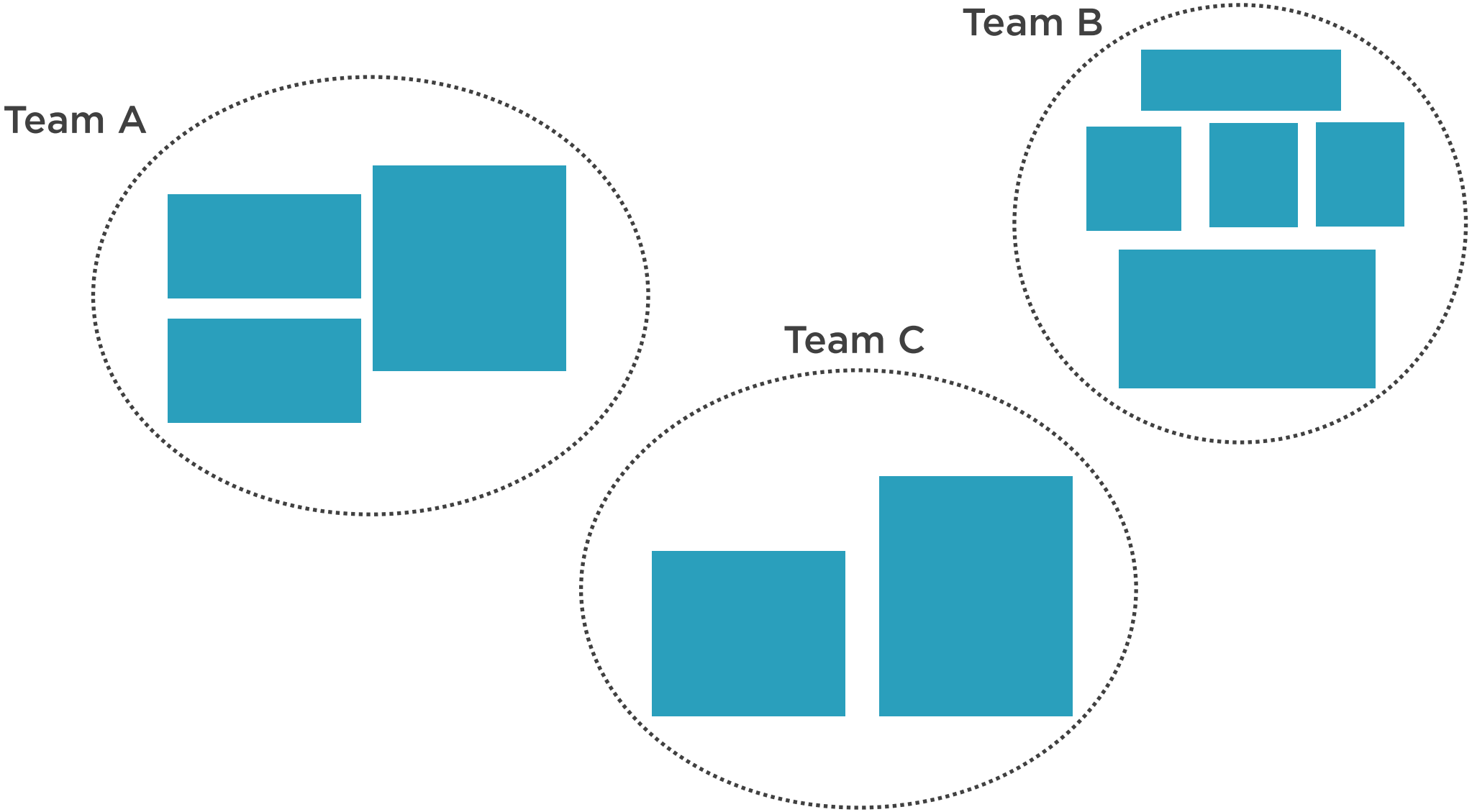
Starting Out with a Monolith

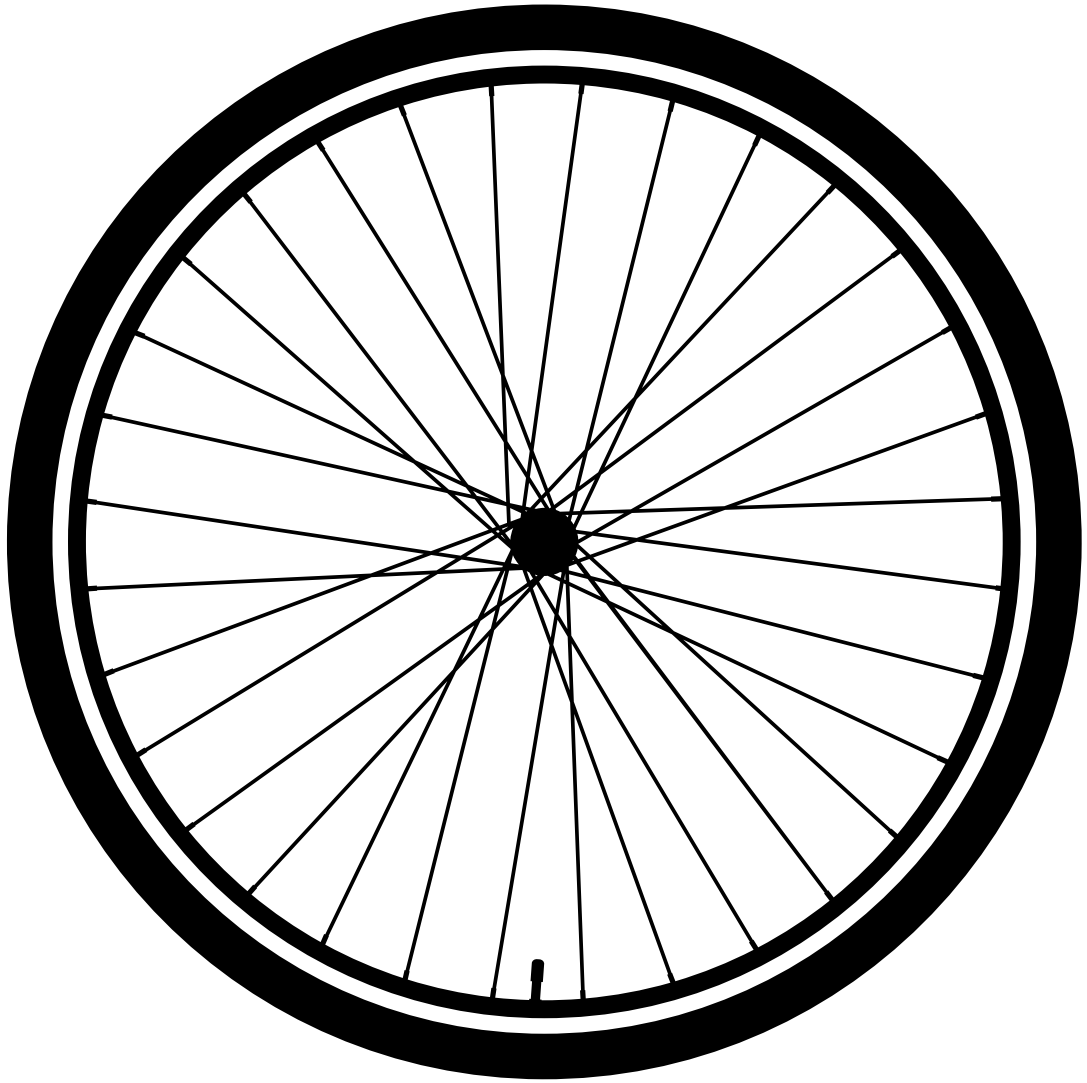


The Growing Monolith



Microservices

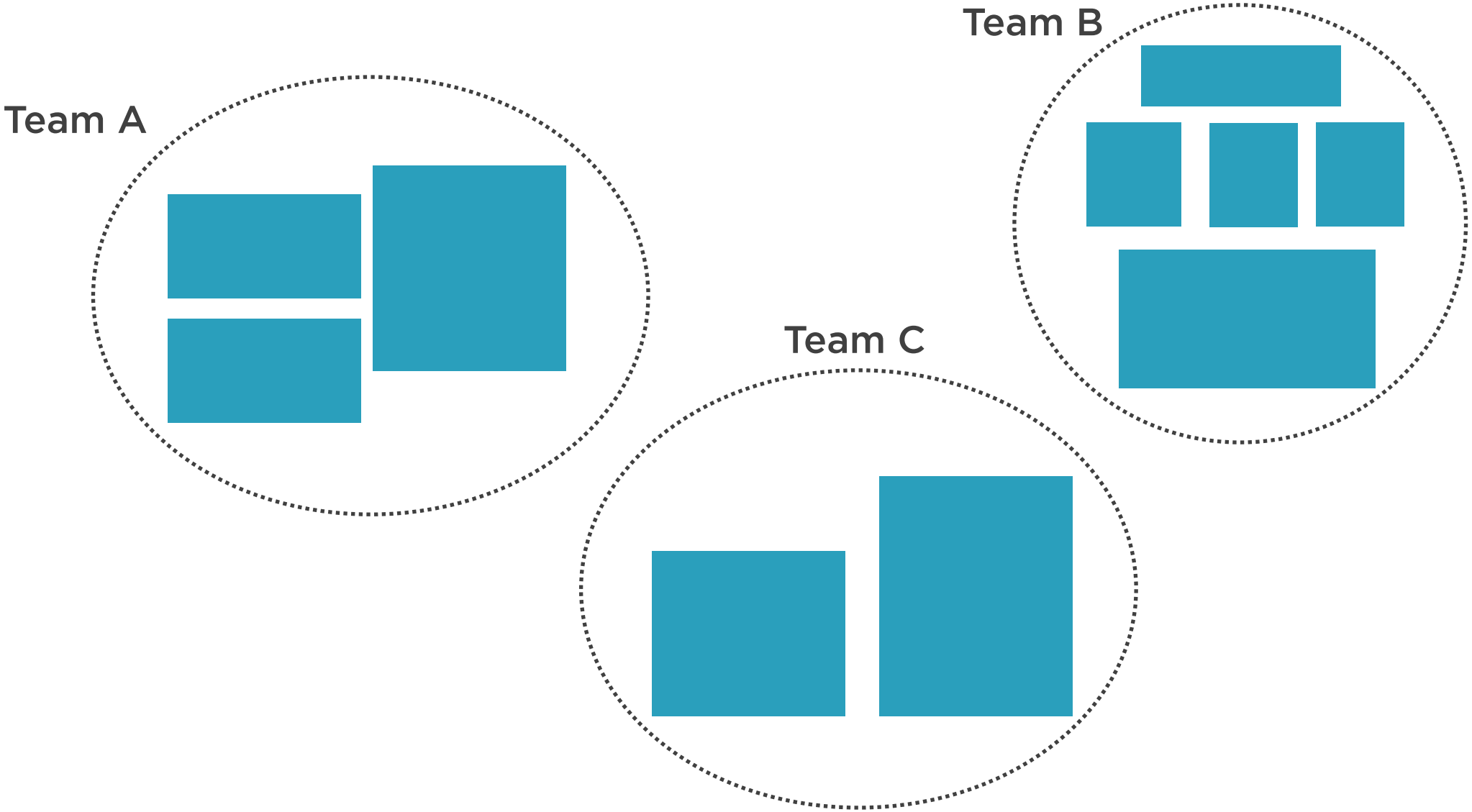




Reasons for Microservice Architecture

- Very large and/or complex applications are easier to develop
- Highly available
- Individual microservices are scalable
- Development experience better when many people collaborate
- Teams can use various technologies
- Fault isolation

Microservices



Organization Readiness

Project-based approach doesn't work

Devops

Ownership

Ground rules across teams

Microservices: The Downsides

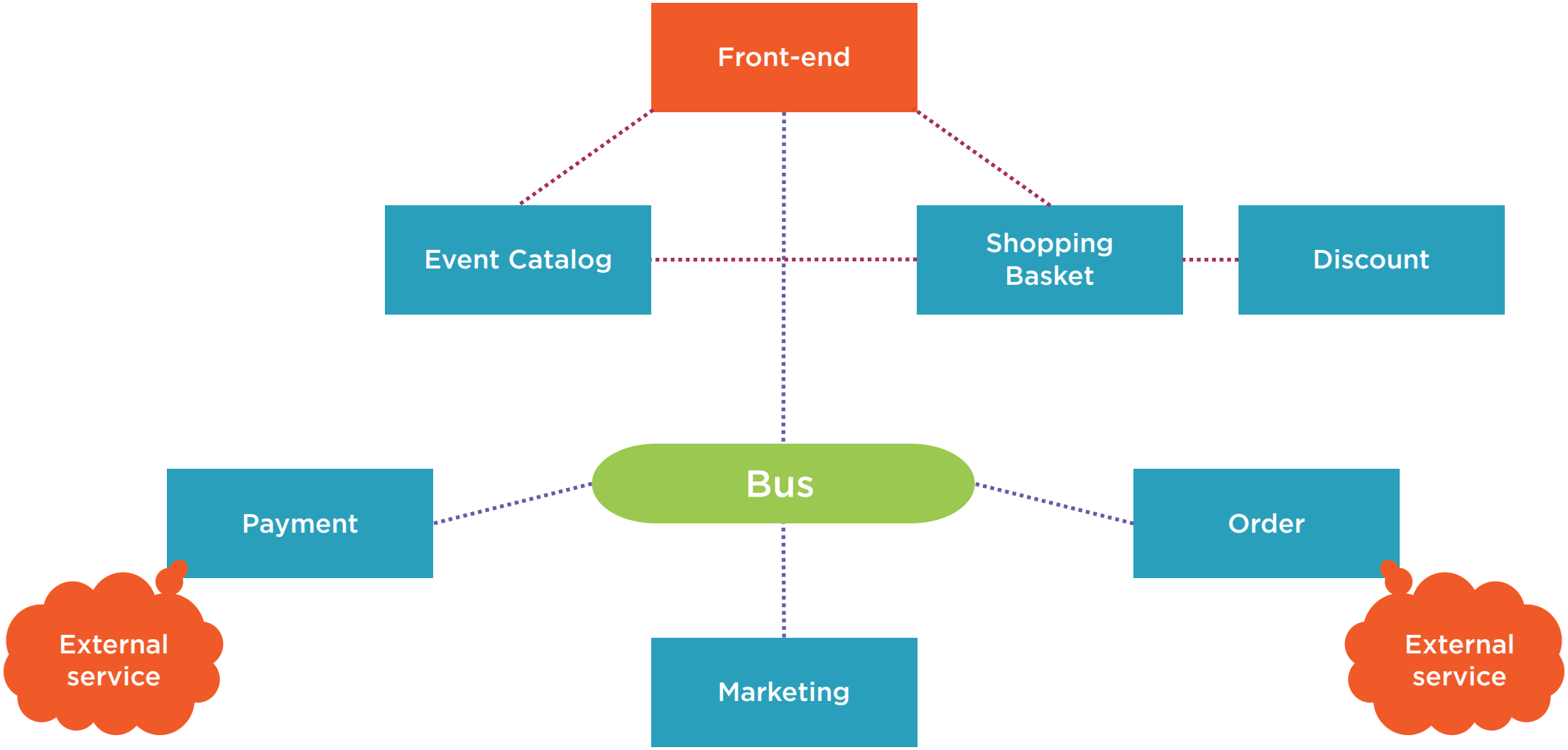
The solution as a whole is more complex

- How do all parts stick together?
- Deployment and monitoring
- Requires team skills beyond coding
- Properly sizing microservices

Eventual consistency

Most applications don't need a microservices architecture

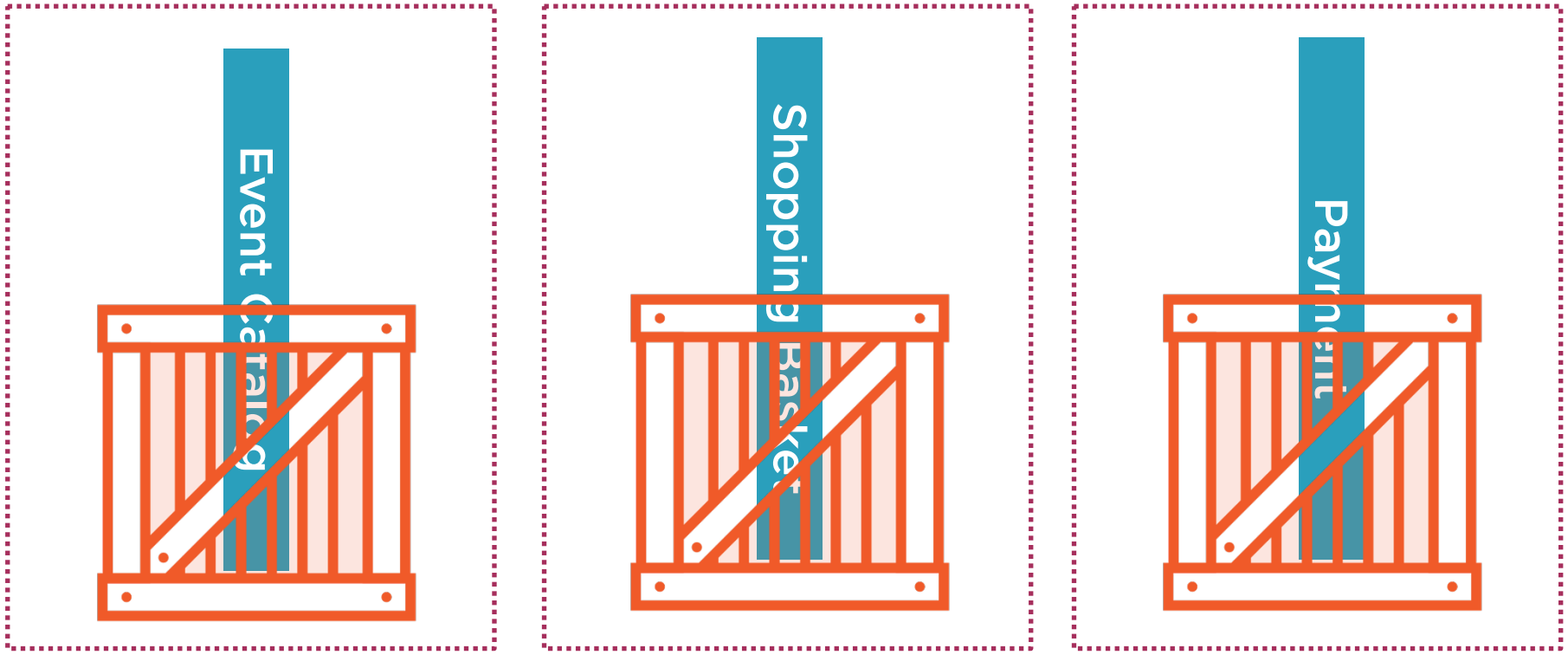
Complete GloboTicket Solution



Why ASP.NET Core?

- Performance
- Stability
- Footprint
- No dependencies
- Platform flexibility
- Tooling
- REST and gRPC frameworks
- Monitoring

The Container



Why Run in Containers?

Portability

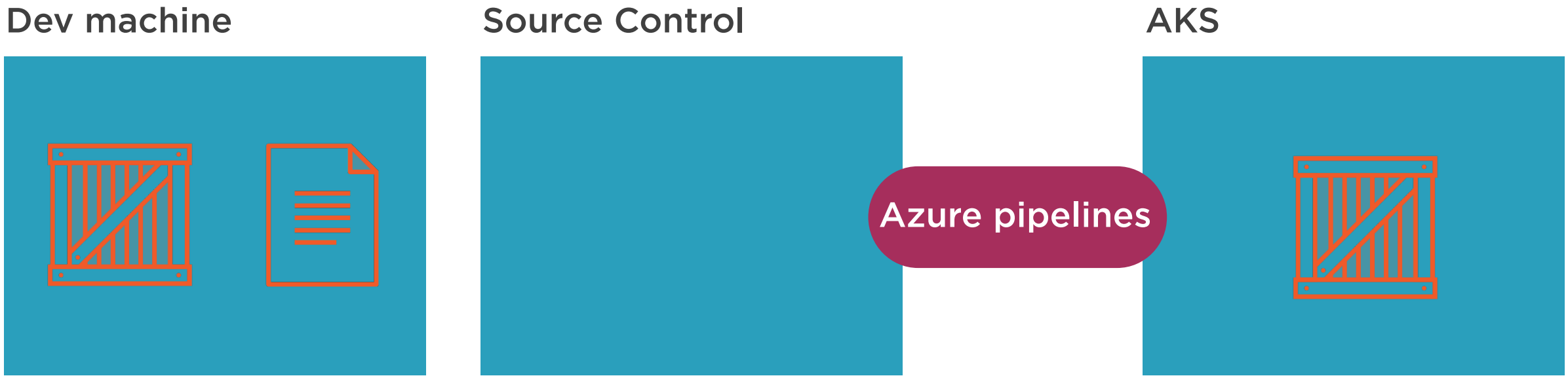
Scalability

Identical experience on dev machine and in production

The Container Orchestrator

- Safely deploy a new version
- Configure multiple instances
- Multiple servers
- Basic container monitoring
- Functionality expandable by adding more containers

Deploying Containers

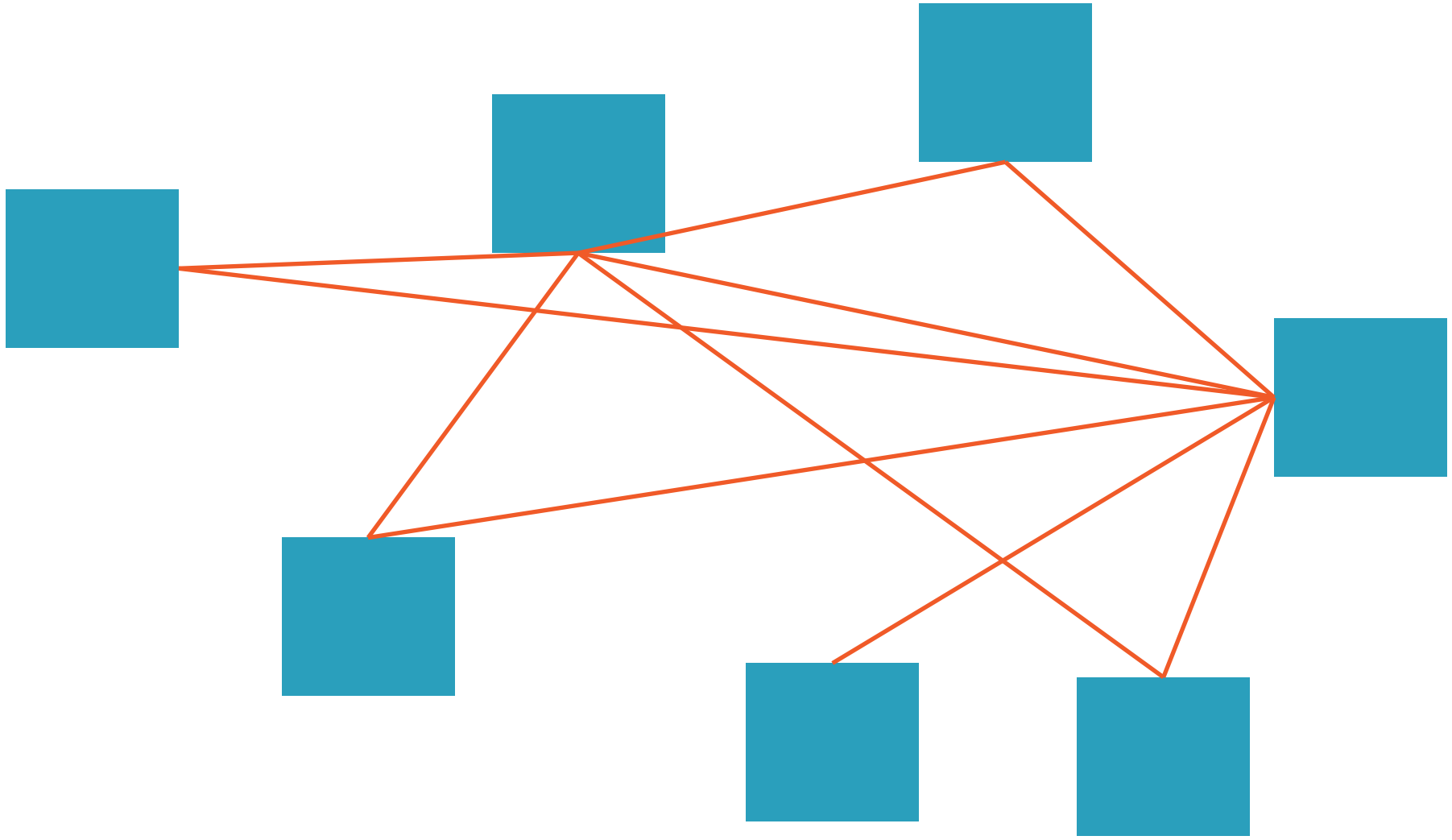


Service Bus and Transport

Service Bus = Rebus

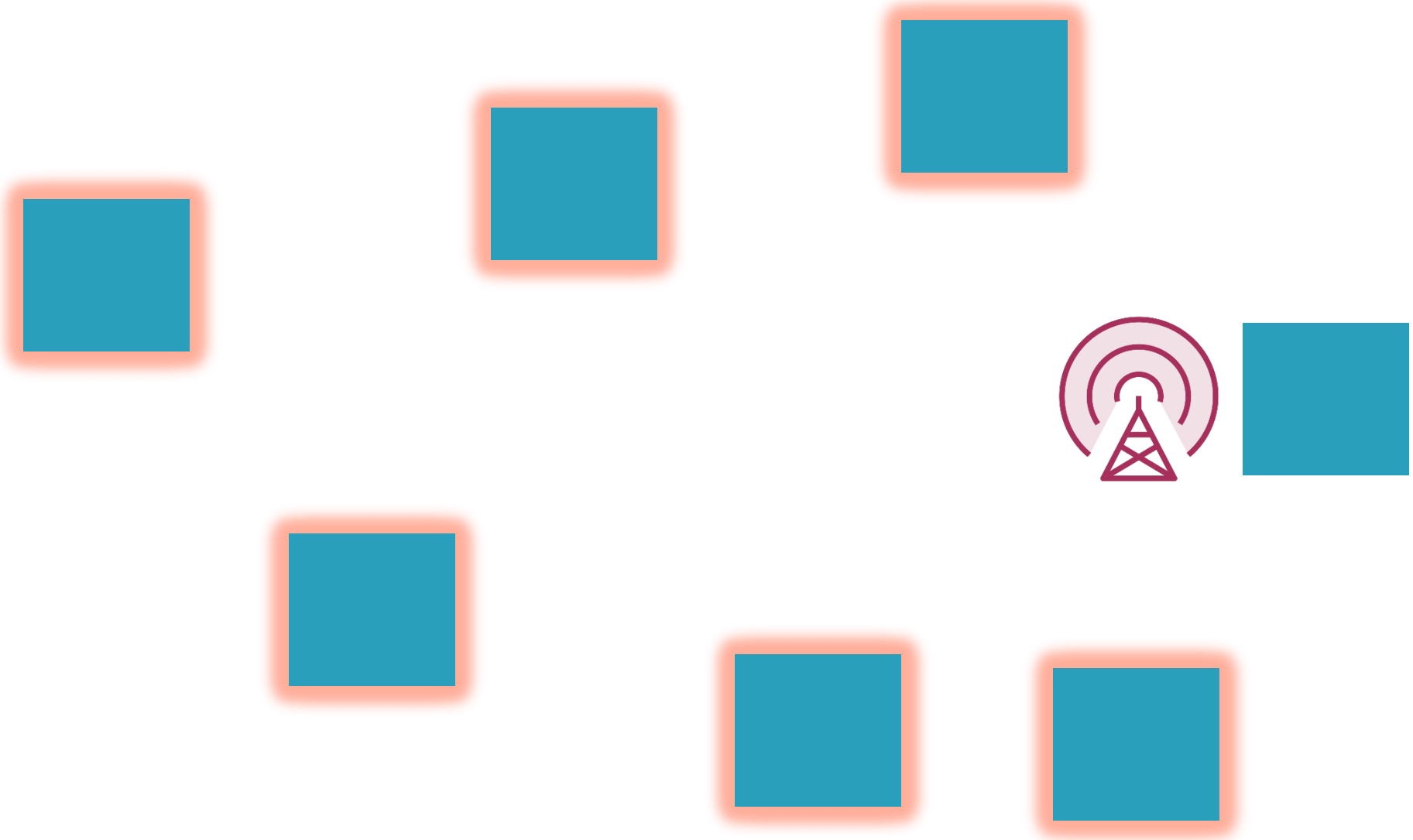
Transport = Azure Storage Queues

Messaging with Many Microservices



Messaging with Many Microservices

Messaging with Many Microservices



Service Bus and Transport

Service Bus = Rebus

Transport = Azure Service Bus

Upgrading the Transport

- Guaranteed first-in-first-out (FIFO)
- Support for message transactions
- Dead lettering
- Guard message duplication
- Sending batches

Service Bus and Transport

Service Bus = NServiceBus

Transport = Azure Service Bus

Why NServiceBus?

More reliable

- Tested
- Documented
- Supported

Tools

Scaling Applications with Microservices and Nservicebus 6

Summary

- Use microservices only when needed
- ASP.NET Core is great
- Containers and orchestrators
- More reliable transports and service bus