Pivotal

Going Cloud Native

Presenter

Monolithic Motivation...consider

Typical Context

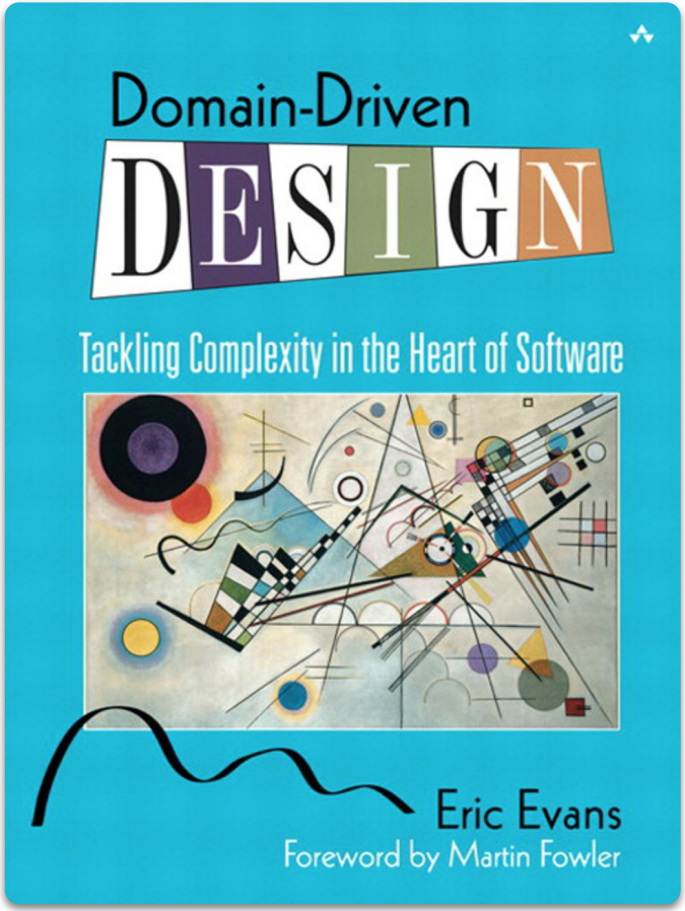
You're developing a non-trivial application that must support a variety of clients such as web, native mobile, mobile web, tablets and wearables. The application has to expose external 3rd-party and internal integration points. The typical request-response flow is to receive a request, validate, perform business logic, access data and/or exchange messages with other systems then return a response to client.

Classic Problem

A rolled up deployment couples together all parts of a software and thus is bulky, hard to change and even harder to deploy with velocity. Force equals Mass times acceleration holds true. Given a relatively stable number of Forces such as teams and technology and an increasing Mass of software we see by natural law Acceleration (delivery) will suffer.

Domain Driven Design

- A community based approach for designing software that models the complexity of the real world
- Kicked off in 2004 by Eric Evans
- Basic principles are
  - Crunch Knowledge
  - Communicate with a shared language
  - Constantly bind reality with model and model with implementation
- A practical microservice design methodology



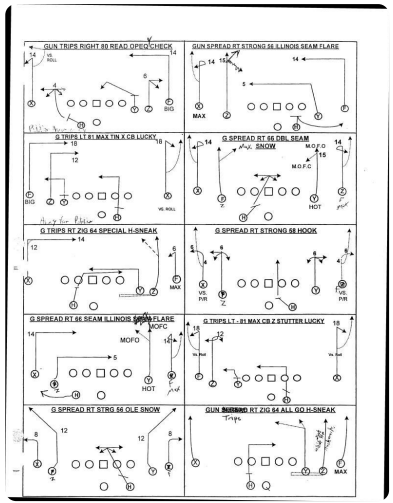
Domain Driven Design

Can help ...

- Software development as a profit center
- Drive a thought first approach culture
- Develop a clear mental model
- Improve collaboration between businesses
- Code quality and testability

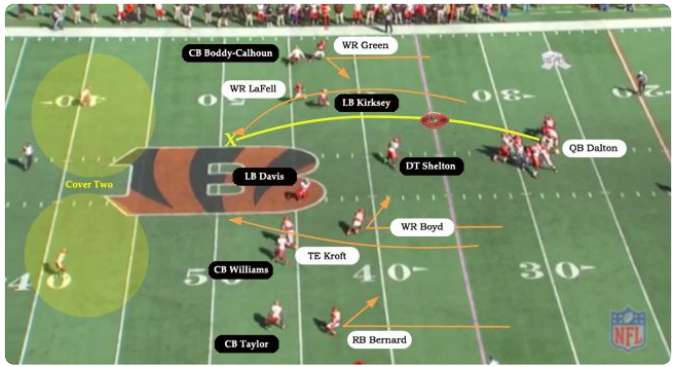
Domain Driven Design

2 Main parts



Strategic

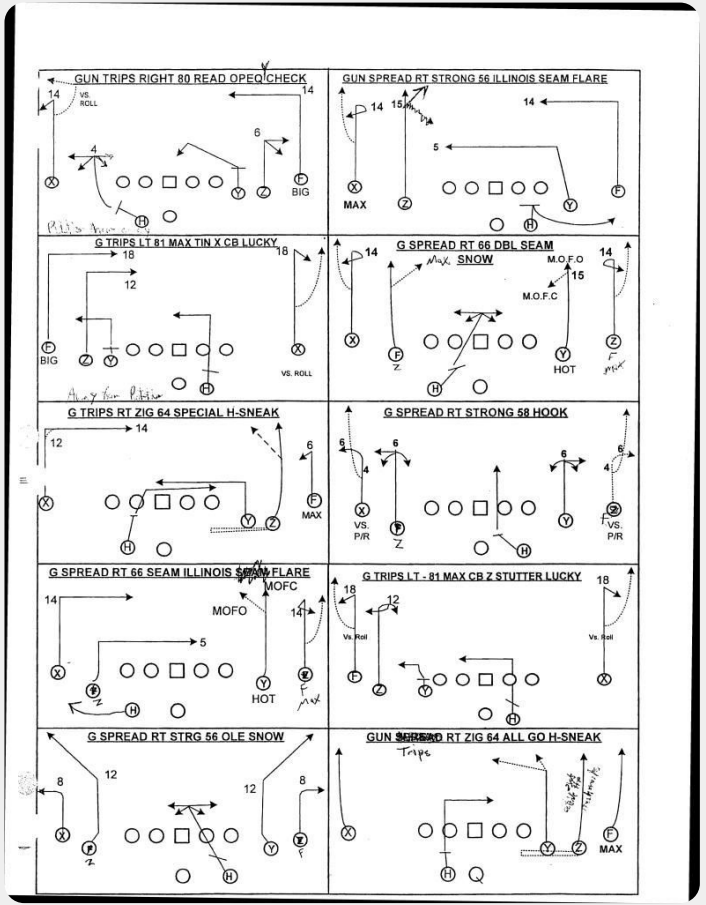
Tactical



Pivotal

Strategic Design

- Domain
- Subdomain
- Bounded Context
- Ubiquitous Language
- Context Maps

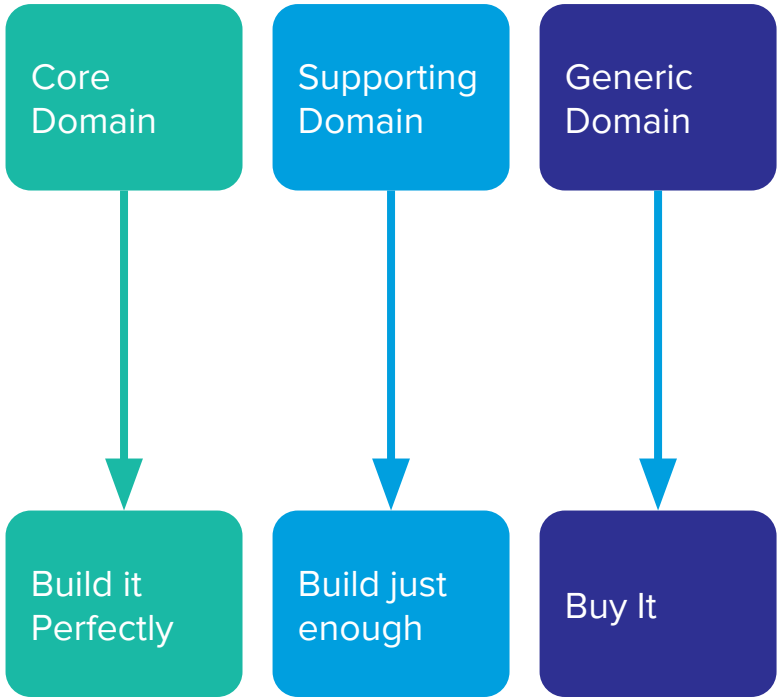


Pivotal

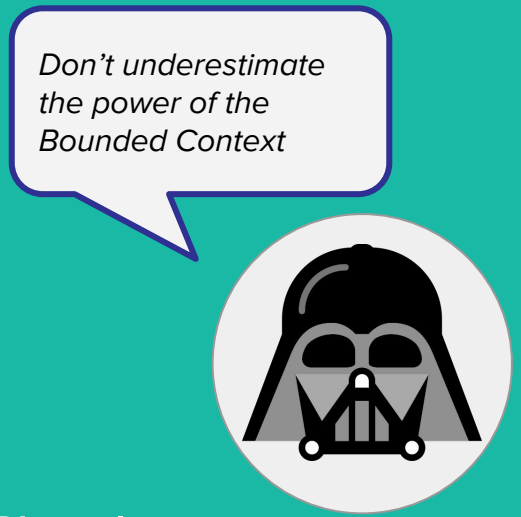
Domains

- Domain is always org specific
- Captures the unique and fundamental element of the business
- Composed of sub-domains & bounded contexts

Subdomains



Strategic Design

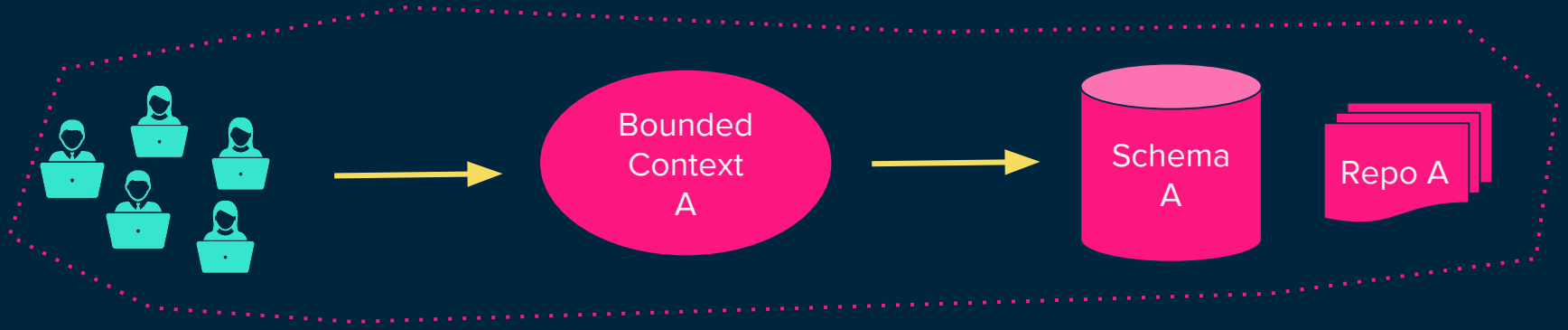


Bounded Context

- Semantic contextual boundary where your model is implemented and a Ubiquitous Language spoken
- Separate Bounded Contexts should be separate software artifacts
- Each component inside the Bounded Context has a specific meaning and does specific things
- Developed to distinguish your organization competitively from all others
- Choose wisely what should and shouldn’t be part of your Core Domain

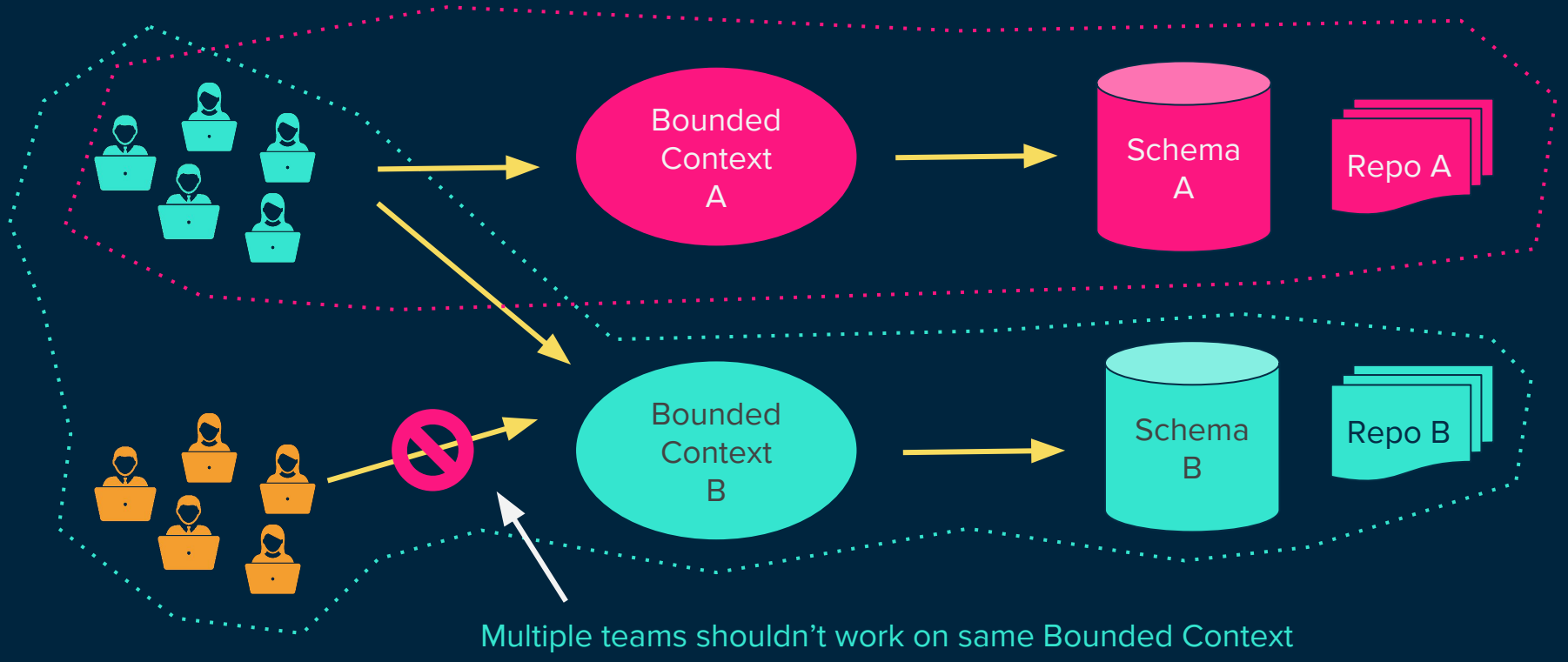
Strategic Design : Bounded Context + Teams + Repos

One team owns one or more Bounded Contexts

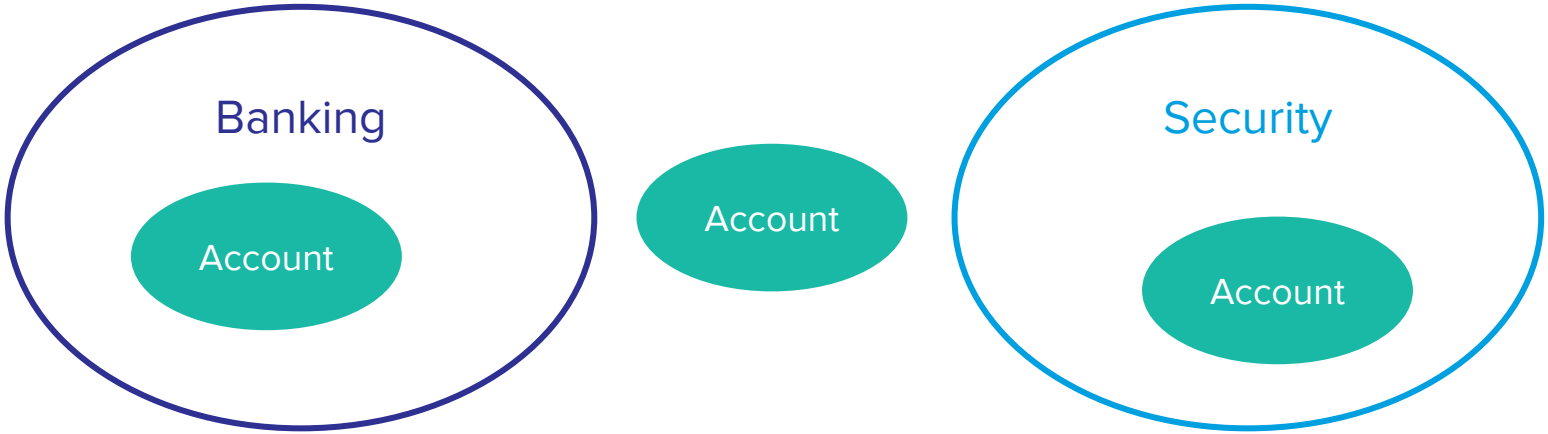


Strategic Design : Bounded Context + Teams + Repos

One team owns one or more Bounded Contexts

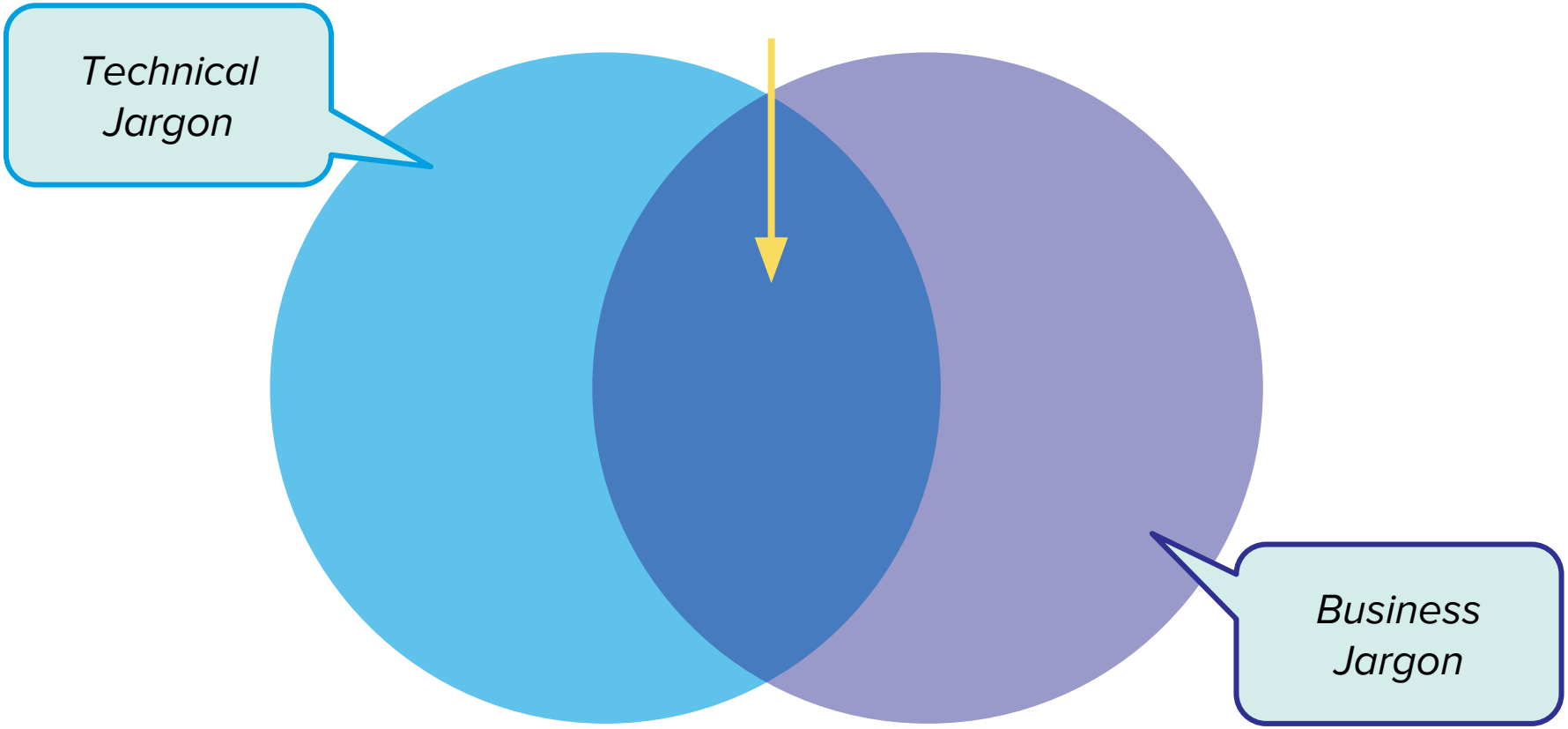


Bounded Context



Context is important!

Ubiquitous Language



Cloud Native Design - 12 factors

Codebase
One codebase tracked in revision control, many deploys

Dependencies
Explicitly declare and isolate dependencies

Configuration
Store config in the environment

Backing services
Treat backing services as attached resources

Build, release, run
Strictly separate build and run stages

Processes
Execute the app as one or more stateless processes

Port Binding
Export services via ports

Concurrency
Scale out via the process model

Disposability
Maximize robustness with fast startup and graceful shutdown

Dev/Prod Parity
Keep dev to prod as close as possible

Logs
Treat logs as event streams

Admin Processes
Run admin and management tasks as one-off processes

Cloud Native
So What?

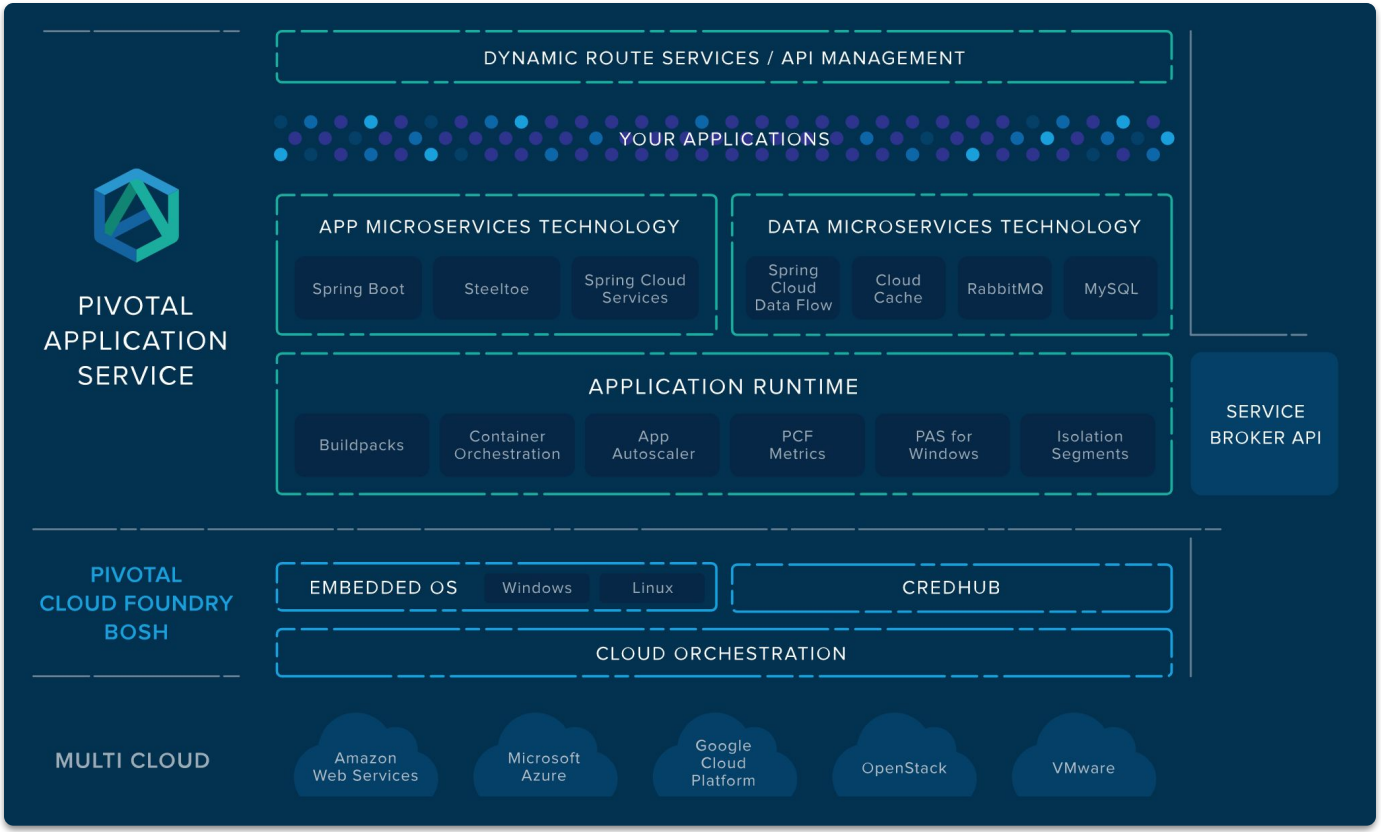
Because you can
build
great Software...like

Cloud Native Implementation

#1 Codebase 	#2 Dependencies 	#3 Configuration 	#4 Backing Services 
#5 Build, Release, Run 	#6 Processes 	#7 Port Binding 	#8 Concurrency 
#9 Disposability 	#10 Dev/Prod Parity 	#11 Logs 	#12 Admin Processes 

Pivotal

Pivotal Application Service



Cloud Native Evaluation

Cloud Native

- Microservice Architecture
- API first design

Cloud Resilient

- Design for failure
- Apps are unaffected by dependent service failure
- Proactive testing for failure
- Metrics and monitoring baked in
- Cloud agnostic runtime implementation

Cloud Friendly

- 12 Factor apps
- Horizontally scalable
- Leverage platform for HA

Cloud Ready

- No file system requirements
- Containerized
- Platform managed addresses and ports
- Consume Platform services

Transforming How The World Builds Software