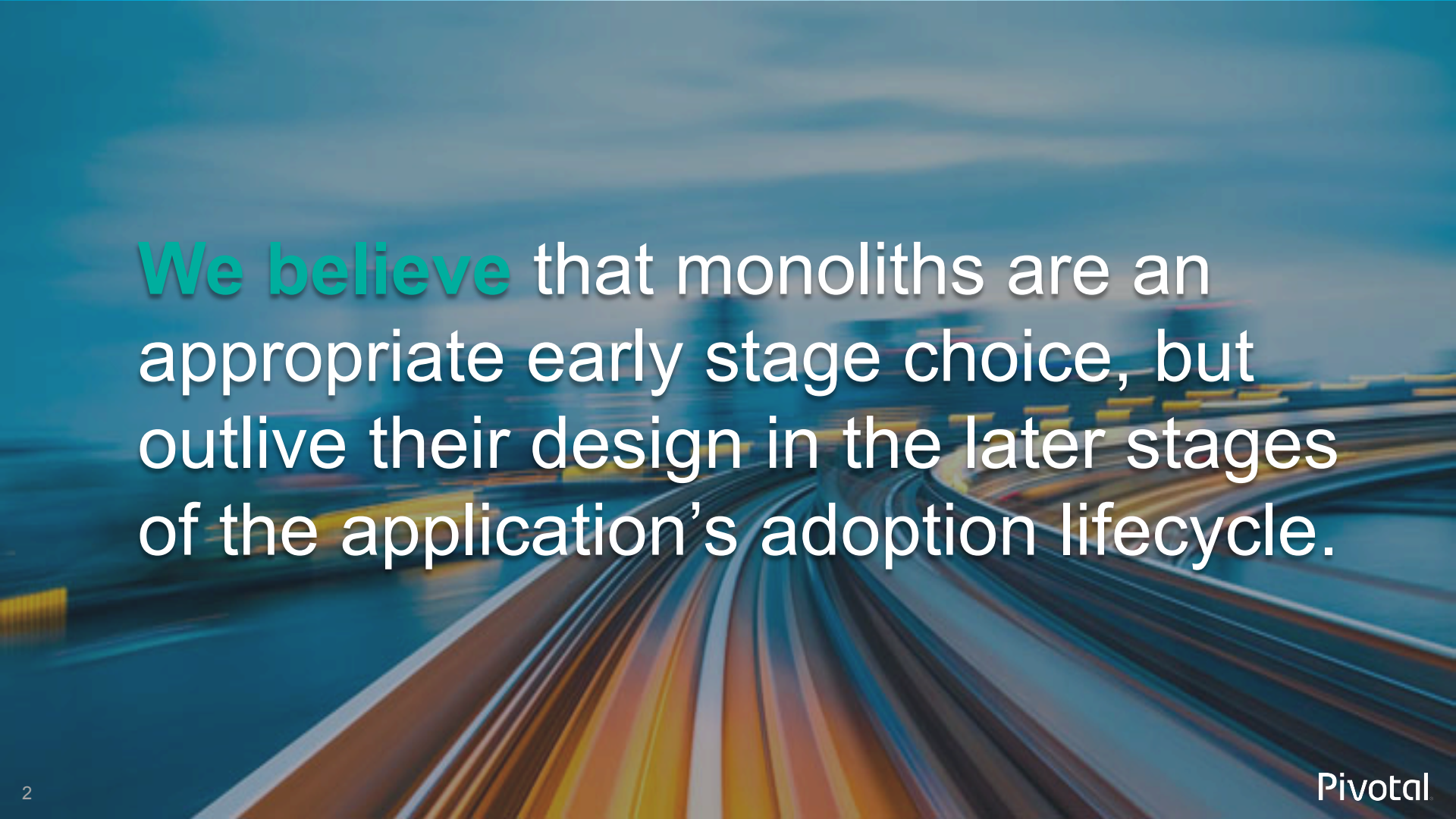
Pivotal

Breaking The Monolith

Migrating Your Legacy Portfolio to the Cloud with Spring and Cloud Foundry

Rohit Kelapure, Pieter Humphrey


We believe that monoliths are an appropriate early stage choice, but outlive their design in the later stages of the application's adoption lifecycle.

Legacy Portfolio Realities

Things we hear:

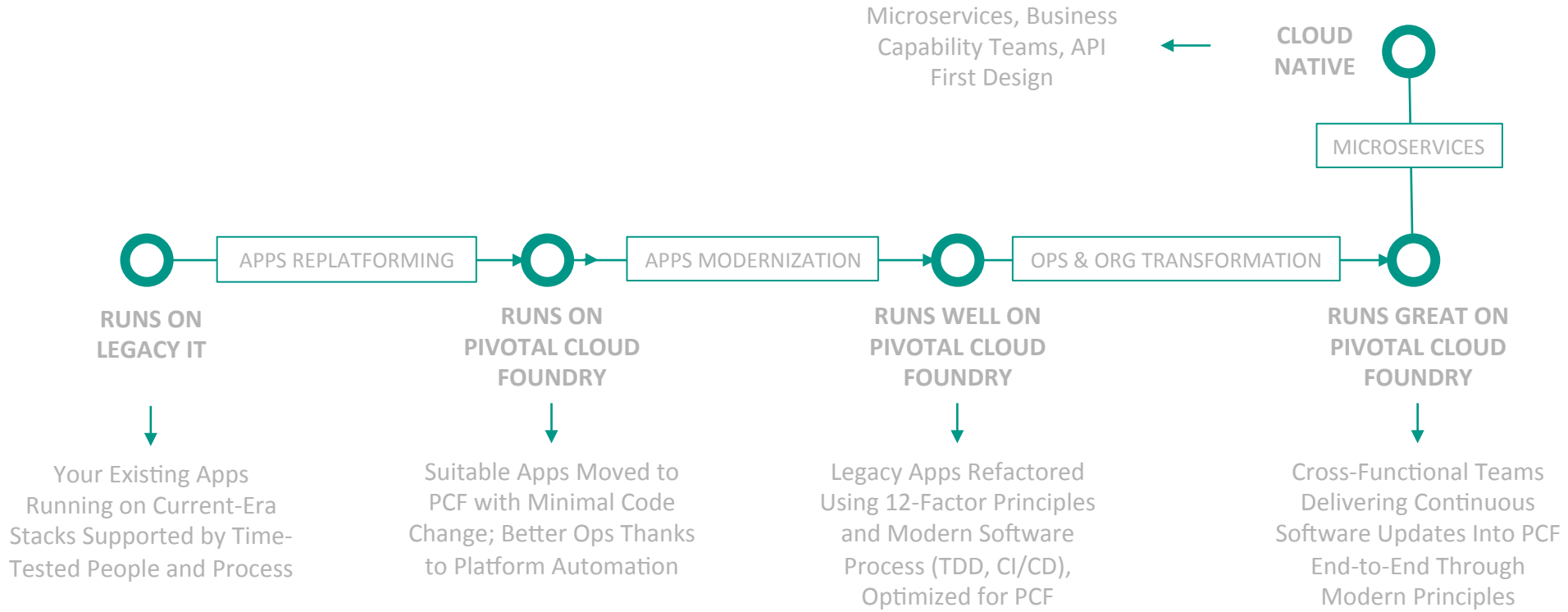
- Our portfolio is **complicated**, **documentation is sparse** and it's a mix of many things
- Architecture is often **tightly coupled** code and **complex dependencies**
- Many people spend their days **working with legacy technology**; they lack new skills

What we believe:

- Pivotal Cloud Foundry is the **comprehensive Cloud Native platform** to build the future
- You **get value** by **building apps on it**
- You **see more value** by moving existing apps

Pivotal

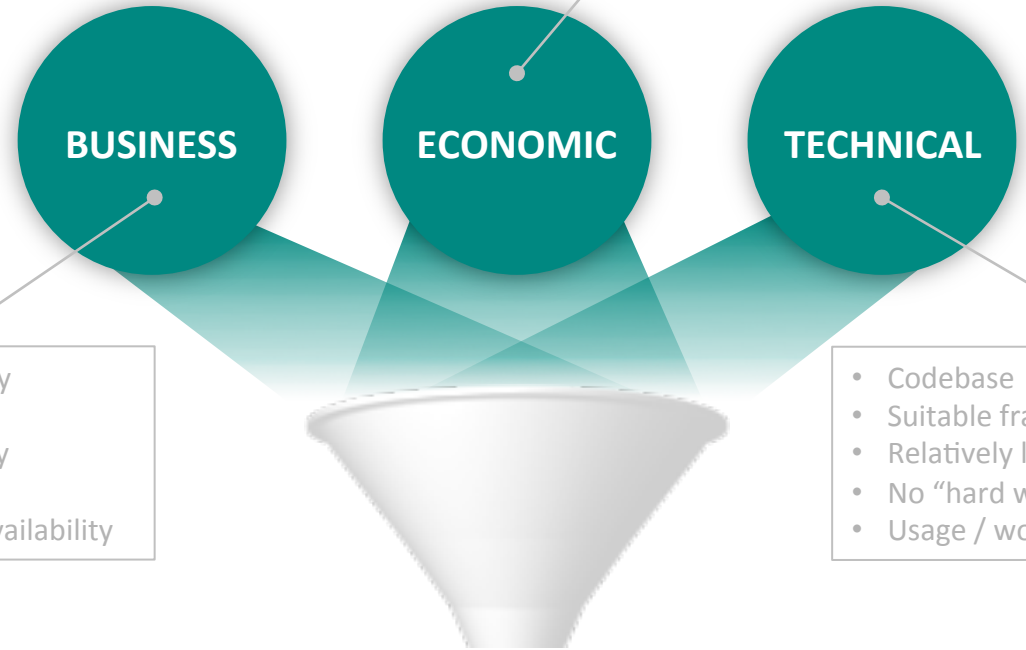
The Cloud Native Applications Journey



Pivotal

Choosing Candidate Apps

- Licensing costs (HW / SW)
- Time-to-market implications
- Revenue impact-fullness

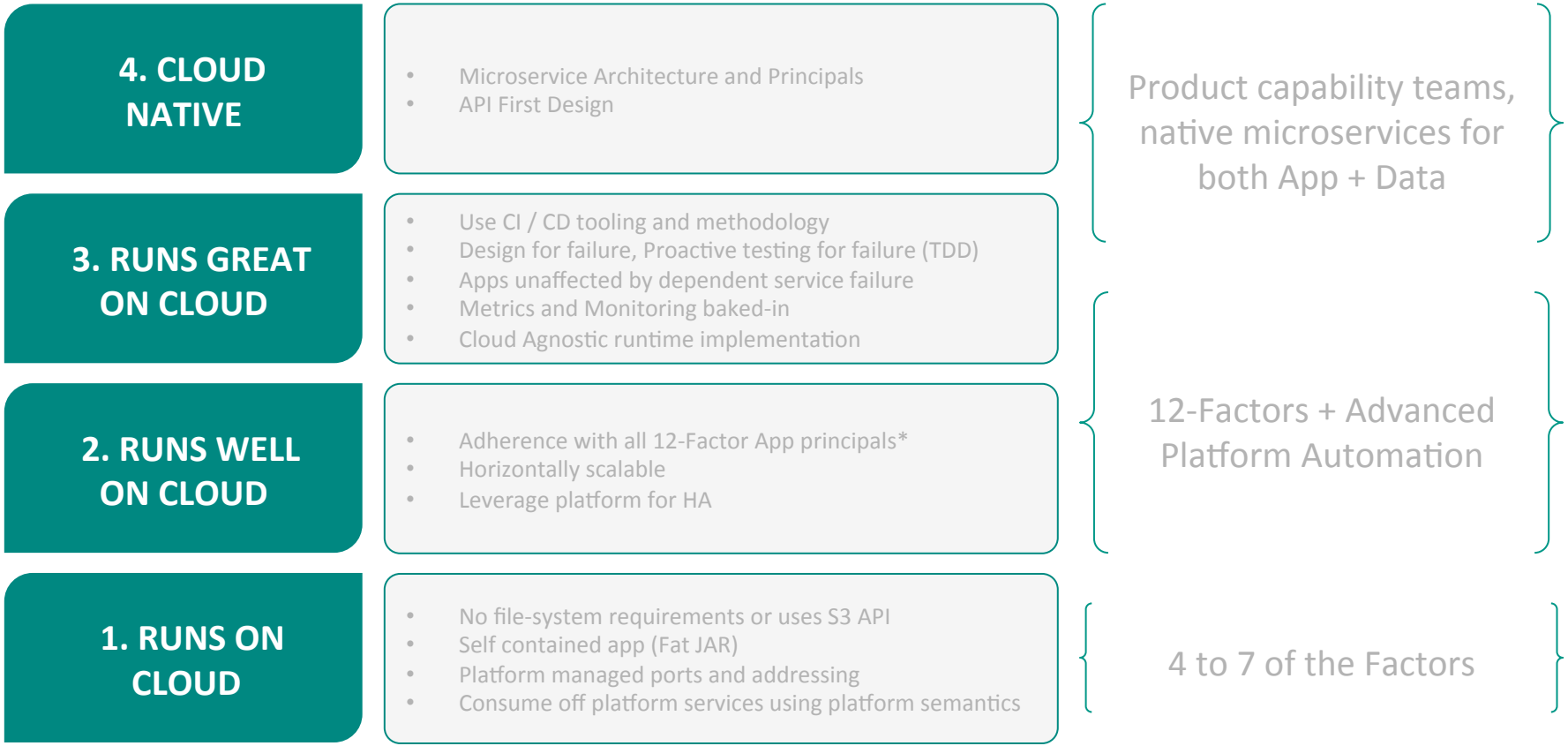


- Business criticality
- Risk tolerance
- Change frequency
- Lifecycle stage
- Domain expert availability

- Codebase
- Suitable framework / runtime
- Relatively lightweight footprint
- No “hard wire” file system dependencies
- Usage / workload

Pivotal

A Legacy Maturity Model



* We believe there are more like 15 “factors” that exemplify a true, “Cloud Native” application… more later

Pivotal

Why Bother Migrating Monoliths to PCF?

BENEIFTS

- Cloud Portability – free to run on any Cloud
- Significant operational efficiencies – reduce headcount
- Auto-scaling; efficiencies for spikey workloads
- Automate patching, upgrading, and lifecycle management
- Standardize on dev stacks and platform-provided services
- Constant innovation of platform capabilities
- Management and monitoring through the platform
- Runtime consolidation and reduction of multiple vendors

Pivotal

Traditional Ways of Tackling Modernization

- Lots of Upfront Study in a Multi-Phased Program
- Uses Tools and/or Surveys to Gather Information
  - Consultants who cast a wide net over everything
  - Delivery of an expensive report and phased roadmap definition
- Long projects with big budgets and large batches of work
  - Failure is slow; it takes time and a lot of money to see problems
  - Value is slow; measured returns often take years

Recommended Migration Path

4. CLOUD NATIVE

- Microservice Architecture and Principals
- API First Design

3. RUNS GREAT ON CLOUD

- Use CI / CD tooling and methodology
- Design for failure, Proactive testing for failure (TDD)
- Apps unaffected by dependent service failure
- Metrics and Monitoring baked-in
- Cloud Agnostic runtime implementation

2. RUNS WELL ON CLOUD

- Adherence with all 12-Factor App principals*
- Horizontally scalable
- Leverage platform for HA

1. RUNS ON CLOUD

- No file-system requirements or uses S3 API
- Self contained app (Fat JAR)
- Platform managed ports and addressing
- Consume off platform services using platform semantics

Work up from #2 to #4 at your own pace

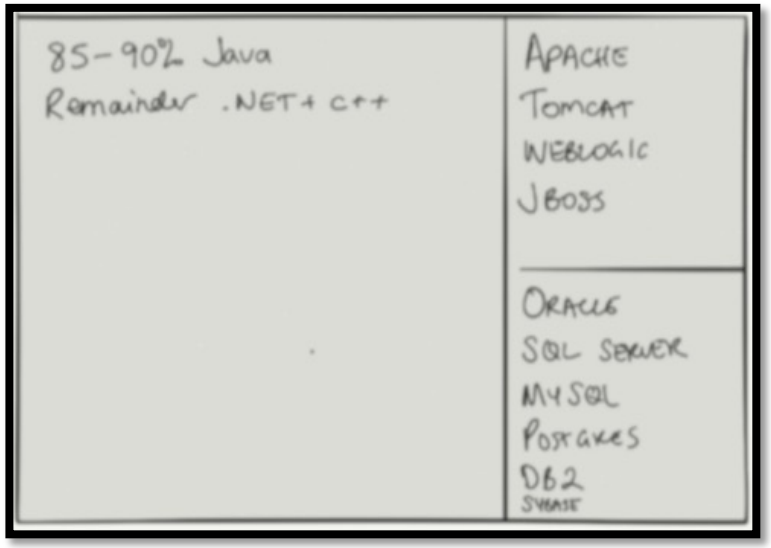
Start here

Pivotal

To begin

Don’t Plan Everything; Start Small and Let Your Work Inform the Strategy

SCOPE → DISCOVER & FRAME → TIME BOUND PILOT



Pivotal

Getting to Cloud Native

Pivotal

The 12-Factors

http://12factor.net/

A “manifesto” of sorts published in 2012 by a team at Heroku

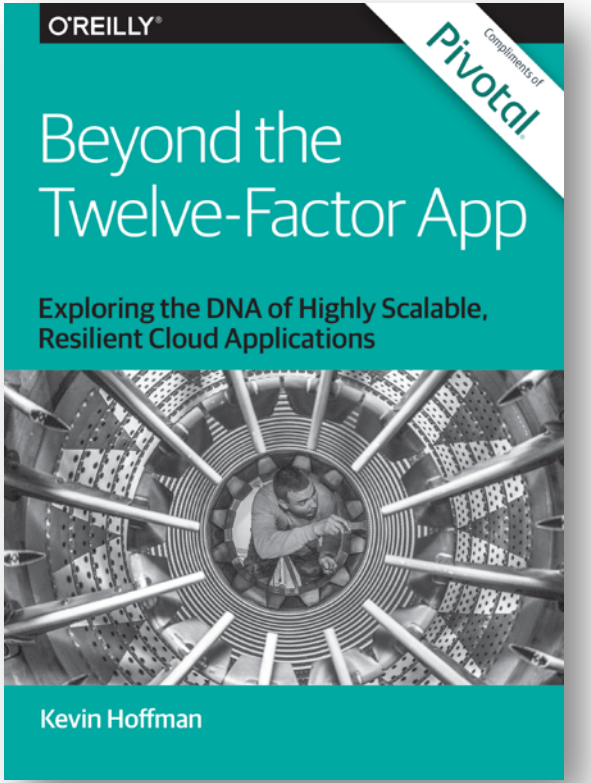
The goal of these 12-factors is to teach developers how to build cloud-ready applications using **declarative formats for automating setup**, had a **clean contract with underlying operating system** and were prepared for **dynamic scaling**

VALUE, APPROACH

I. One Codebase, One App*	= **Time to Market**; find the seams; use good SDLC practices
II. Dependency Management*	= **Dev Productivity**; standardize & remove surprises
V. Build, Release, Run*	= **Release Mgmt Hygiene**; use CI/CD automation /w PCF
III. Configuration*	= **Release Mgmt Hygiene**; move to environment vars
XI. Logs*	= **Real-Time Metrics**; use PCF features; stdout / stderr
IX. Disposability	= **Auto-Scale**; move slow processes to backing services
IV. Backing Services	= **Resiliency / Agility**; use circuit breaker; loose binding
X. Environmental Parity*	= **Reliability**; use well architected PCF, get parity
XII. Administrative Process	= **Reliability**; move to backing service(s), expose as REST
VII. Port Binding*	= **Ops Efficiency**; use PCF features like routing, scaling, etc.
VI. Process	= **Cloud Compatibility**; move state to backing service(s)
VIII. Concurrency	= **Auto-Scale, ZDD**; design for cloud, use PCF features

Pivotal

Looking Beyond 12-Factors



- 12-Factor App Was Published in 2012
  - In context of Heroku
  - A LOT has changed
- This Book Provides New Guidance
  - With emphasis on Enterprise Java & PCF
  - Adds 3 new "factors"
    - API First
    - Telemetry – APM, Logs, Domain-Specific
    - Authn / Authz – Security First Design
- A Must Read for Cloud App Architects

Pivotal

Evaluating Migration Suitability

WON'T RUN ON CLOUD

- Accepts inbound connections for non-HTTP protocol
- Application Container Hosted Clustering
  - Relies on container-based clustering for resiliency and scale
  - Relies on container for shared state (see stateful process)
- Stateful Process
  - Uses in-memory cache (in-process)
  - Could cause corruption or data loss if application was restarted without warning
- Filesystem I/O (Reading files from disk; Writing files to disk; I/O with NFS mount)
- Logging to any destination other than **STDOUT** or **STDERR** (console)
- Use of distributed transactions of any kind, including **XA**
- Extremely long application startup and shutdown time – those measured in minutes
- Use of Java properties files/.NET web.config files
- Use of hardcoded configuration – URLs, Credentials, Database connection information, Queues and Topics
- Nonstandard security - App relies on a nonstandard security mechanism that conflicts with standardized Siteminder security in customized buildpacks
- Batch Processing - Autosys or cron invoke shell or batch scripts to invoke ad hoc functionality

Start by Replatforming Suitable Applications

Work with One Group; Move “10s” of Apps in 10 Weeks

TYPICAL ACTIVITIES

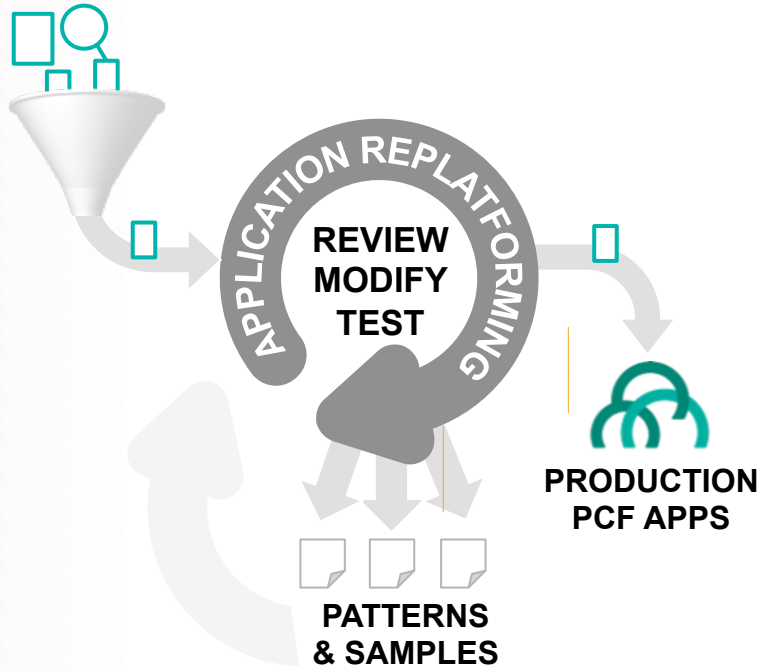
Discovery & Framing
- Suitability Workshops; Technology Planning
- Backlog Development, Grooming & Prioritization

Platform Extensibility
- Buildpack Engineering; Apps Configuration

Test Automation
- Build CI/CD Pipelines; Automate Testing for App & Backing Services (e.g. Connectivity, Perf.)

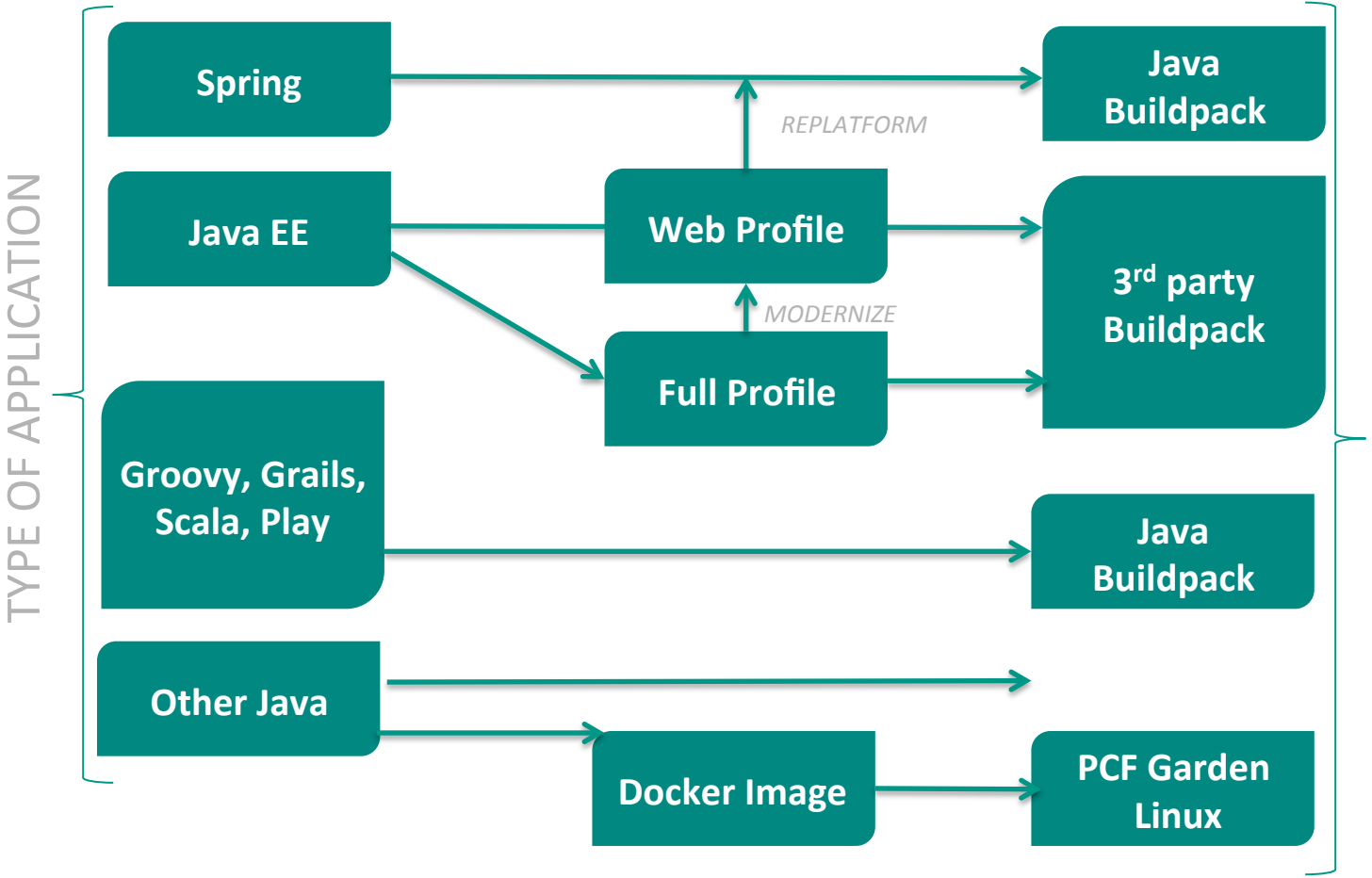
Refactoring
- Backing Service Location / Configuration
- File System Usage, Messaging, etc.

Process and Documentation
- CI Everything (Including Docs)
- Reference Patterns Informed by Work



Pivotal

Replatforming Java apps



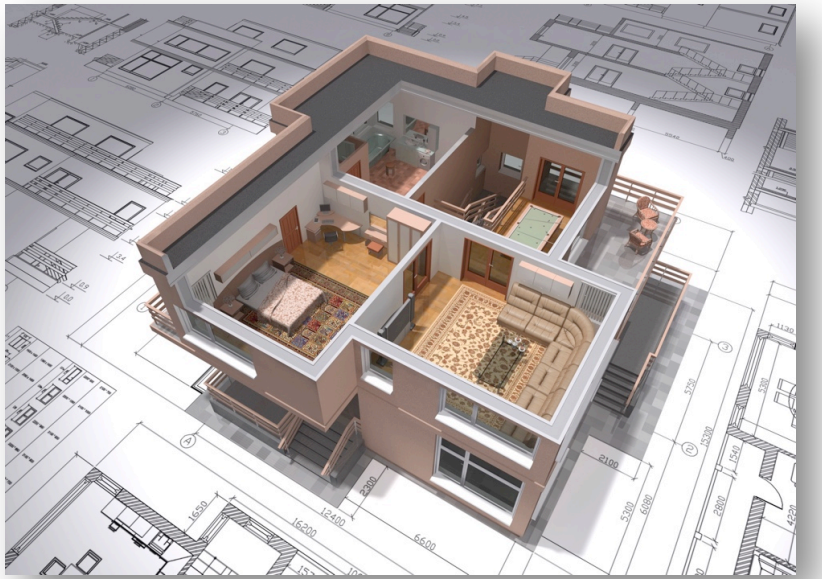
Pivotal

Replatforming to Spring Boot

- Introduce Spring Boot Dependencies
- Introduce Cloud Profile that can read configuration settings from VCAP_SERVICES
- Application should be able to run on PCF and on standalone Tomcat
- Modify the spring boot packaging to use executable jar/war
- Profile Cleanup - A separate profile for every environment / easy and clean
- Tune your CI / CD pipeline to take full advantage of Spring Boot

Breaking the Monolith – Picking Seams

- Stability and Point of Evolution
- Inbound and Outbound Coupling
- Tools - Xray, JDepend, Structure101
- Databases & Data Stores
- Transaction Boundaries
- Modes of Communication
- Team Organization and Structure
- Use Cases/User Journeys
- Business Processes
- Verbs & Operations
- Nouns & Resources
- Separated models for reading and writing



Pivotal

Generic Bounded Context Refactoring Recipe

BOOT + HEX ARCHITECTURE

- Search for all the call sites into the bounded context
- Analyze the current interfaces exposed by the bounded context
- Define the required ports for the bounded context
- Analyze external dependencies used by the bounded contexts
- Define the required adapters for the bounded context
- Analyze how the bounded context will stay in sync with the rest of the system
- Define what domain events the bounded context will emit
- Define what events the bounded context will listen for
- Copy and paste the code from the old project s
- Create a new spring boot project for hosting the refactored code
- Add the newly configured project to the CI / CD Pipeline
- Write unit and integration tests
- Cut and paste code and refactor it
- Iterate until done

Mikado Method

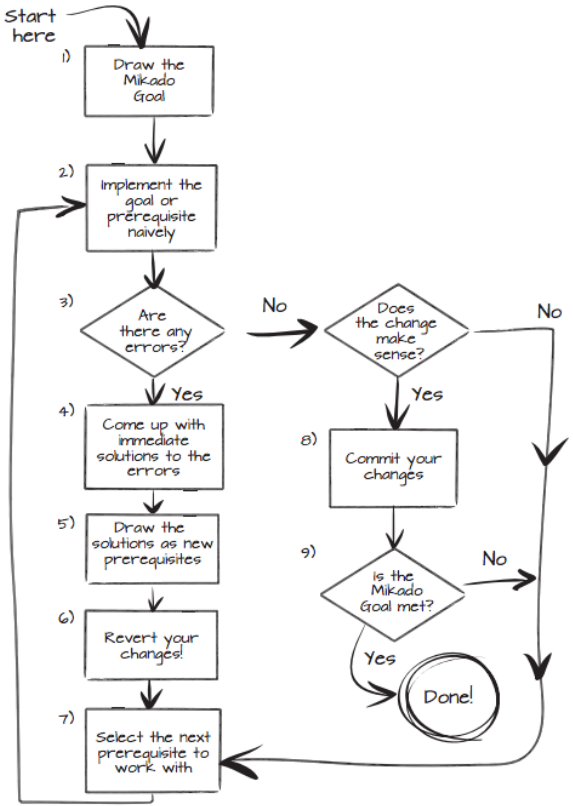
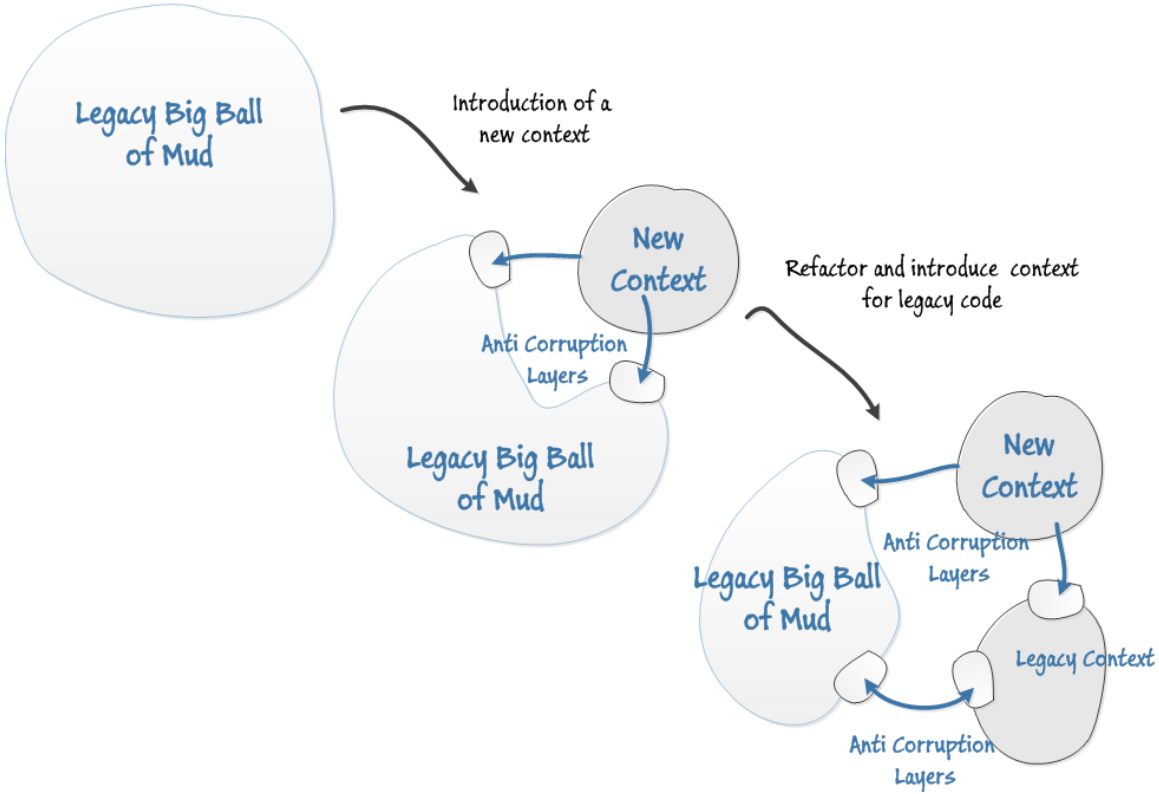


Figure 1.3 A process chart describing the Mikado Method

https://www.manning.com/books/the-mikado-method

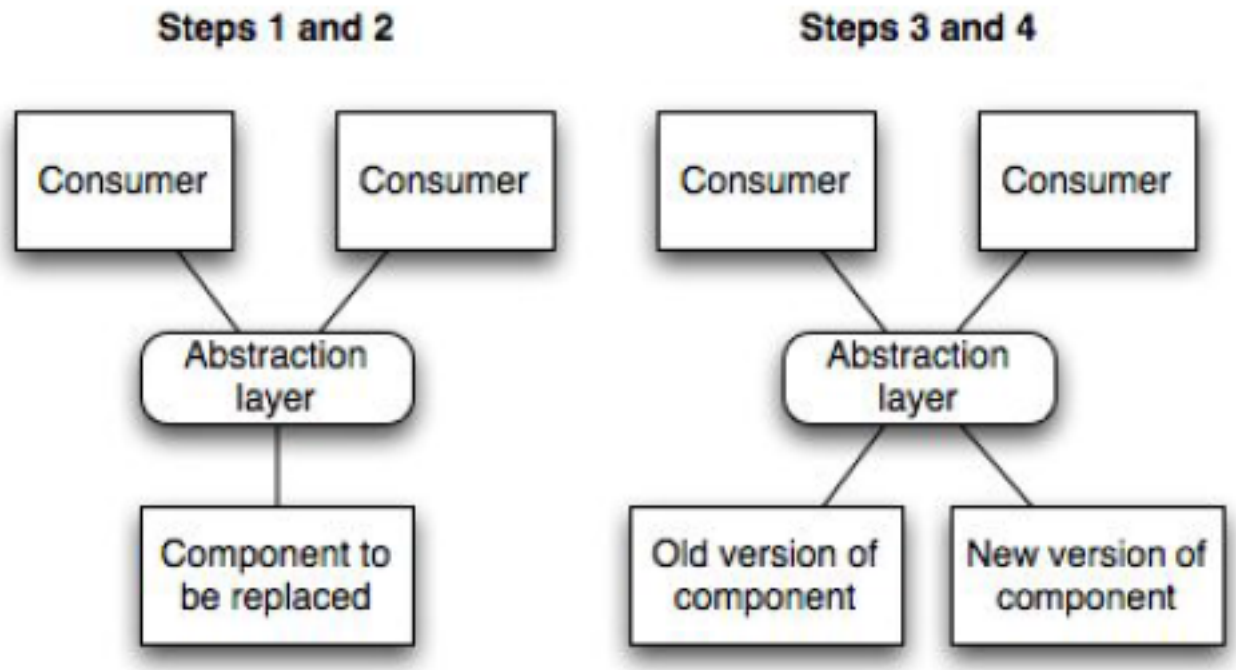
Pivotal

Anti-Corruption Layer

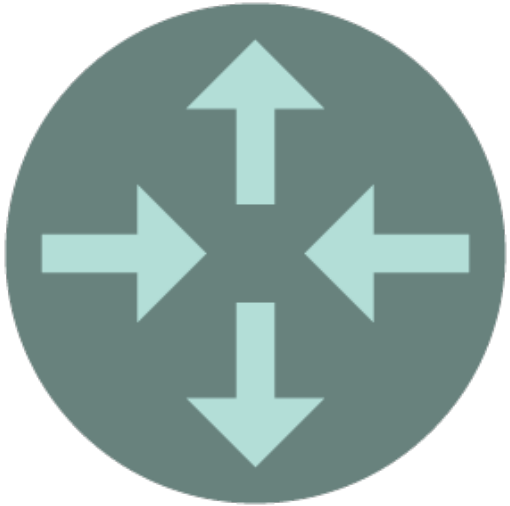


https://leanpub.com/Practicing-DDD

STRANGLING THE MONOLITH



Smart Routing



Dynamic Routing

Service Migration

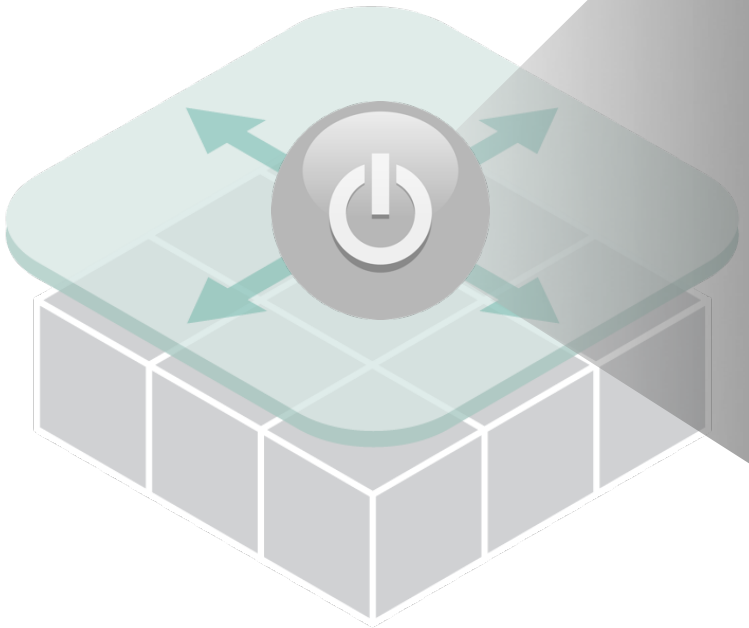
Load Shedding

Canary Testing

Active/Active Traffic Management

Dark Launching/Feature Flags

Wrapping software features in a way that let you turn them on or off



- **Why?**
  - Private beta release
  - Commit your code in logical chunks
  - Release a new feature to all your users at a specific date
  - Not confident in how stable or how scalable a new feature is
- **How?**
  - **Boolean** – Feature will be on or off
  - **Percentage** – Certain % of Users, Cookie, Random, Group
  - **List** – User ID, Group ID, Organization ID, …
  - **Identity** – Always on! and cannot be turned off.
  - **Nil** - Always off! and cannot be turned on.
- **Cloud Foundry Constructs**
  - `cf scale, Configuration Server, Route Services`

Pivotal

Migrating Data

- Tools – SchemaSpy – graphical relationship viewer
- Tools - Liquibase, Flyway, jooQ to Auto apply bundles of database refactorings
- Require a transition period during which both the original and new schemas exist in production
- Expose a Facade service to encapsulate DB changes
- Move logic and constraints to the edge aka services
- Implement retry and compensations
- Database Transformation Patterns cataloged in “Refactoring Databases” seminal book by by Scott J Ambler and Pramod J. Sadalage

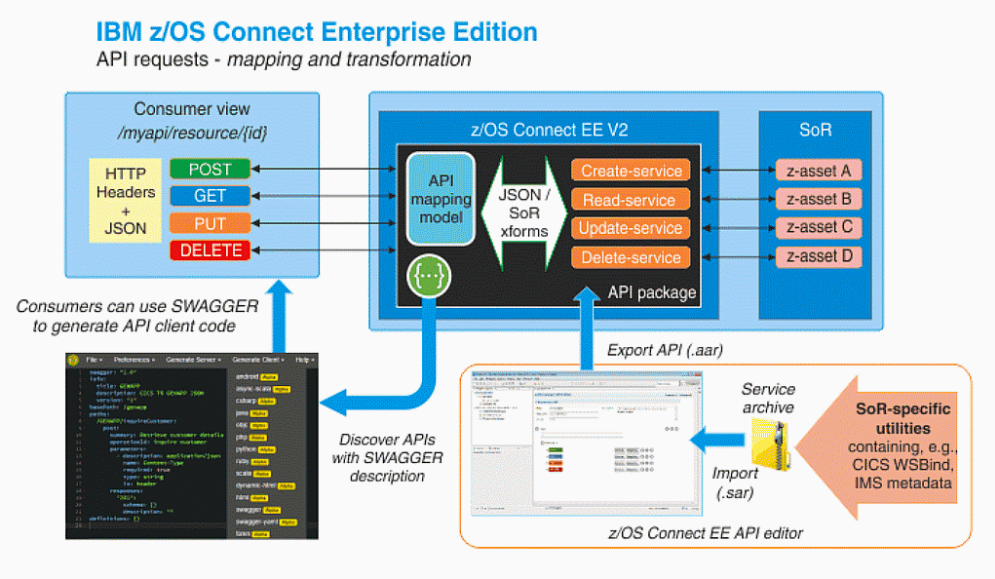
ESB to Microservices

DECENTRALIZTION

- Follow a phased approach to migrating ESB composite and provider services
- Business logic should reside in Java apps and only fundamental ESB functions like legacy adapters and pure transformation and mediation should be handled by the ESB
- Where existing ESB services do not already exist, start greenfield net new development with a pure microservices based approach
- 5 Step Evolution of the ESB to the Cloud
- 1. Co-exist 2. Lift & Shift 3. Refactor 4. Replace 5. Transform

Modernizing Batch

- Address Concurrent Batch and Online
- z/OS v2 Connect
- Eliminate needless Data Movement
- Eliminate file transfer and unnecessary app integration
- Scheduling and Job Management
- Technical Solutions
- Leverage Distributed Batch



Pivotal

A word about organization structure

- Conway's Law asserts that organizations are constrained to produce application designs which are copies of their communication structures
- Leads to unintended friction points.
- Evolve your team and organizational structure to promote your desired architecture
- Break down silos to foster collaboration
- Your technology architecture is then isomorphic with your business architecture.

Application Level Eventual Consistency

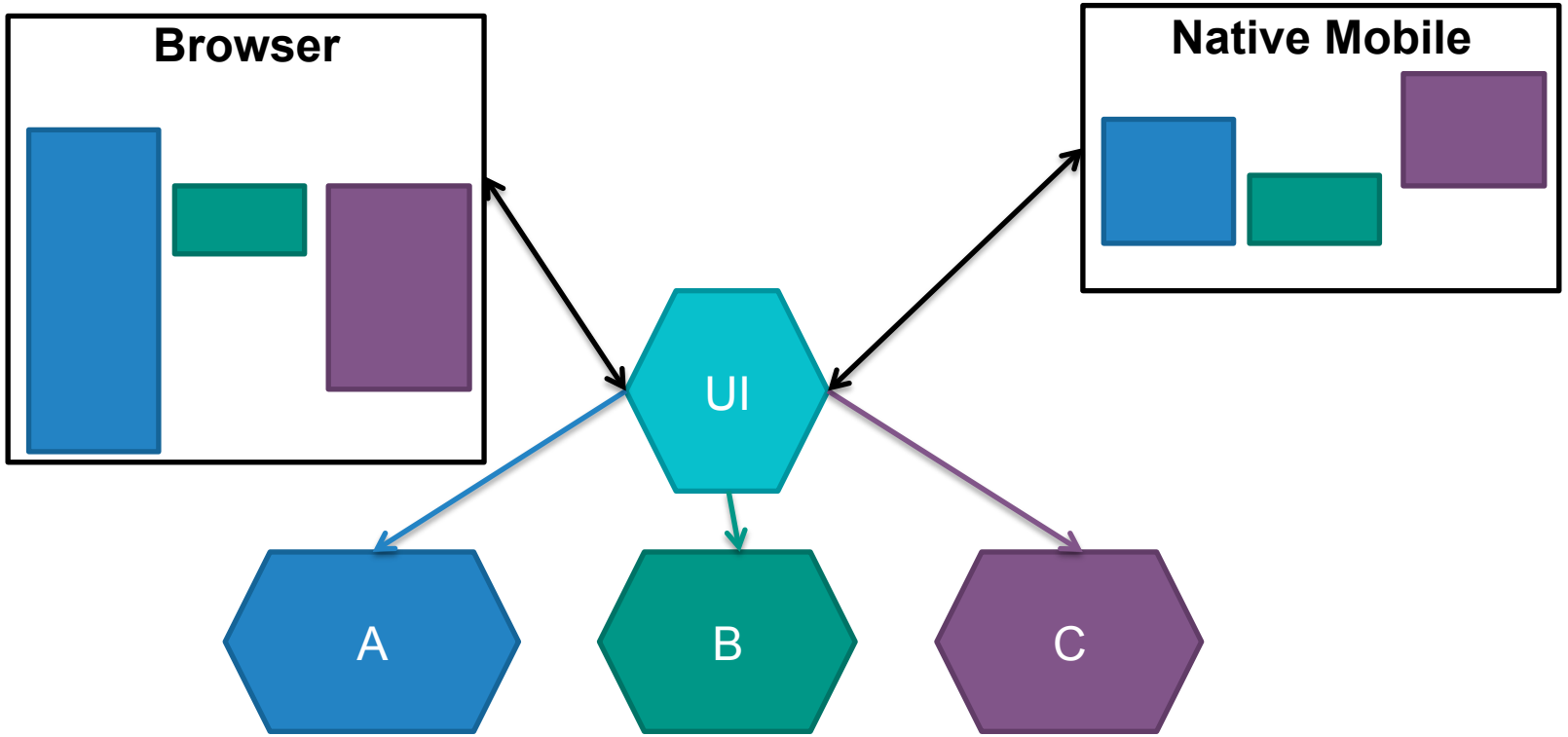
Persist operation state in the client microservice and track to success or compensate

- Databases SQL or NoSQL Gemfire
- Queues (RabbitMQ, JMS, WebSphereMQ, Tibco .. etc)
- Spring State Machine
- Java 8 Completeable Futures

- “Distributed transactions in Spring, with and without XA” from Dave Syer
  http://www.javaworld.com/article/2077963/open-source-tools/distributed-transactions-in-spring--with-and-without-xa.html

Monolithic Edge UI Gateway

Make a UI Microservice that is exposed to end users and have it serve up the UI?



Pivotal

Foreign Keys Constraints

How are Foreign Key Constraints Validated Across Table is Different Bounded Contexts

- Enforcing Foreign Key Constraints between microservices becomes an application level problem to be handled by the microservices rather than the database

- Usage of Immutable Stable URI's to identify Foreign keys can be helpful

Shared Static Data Becomes Code

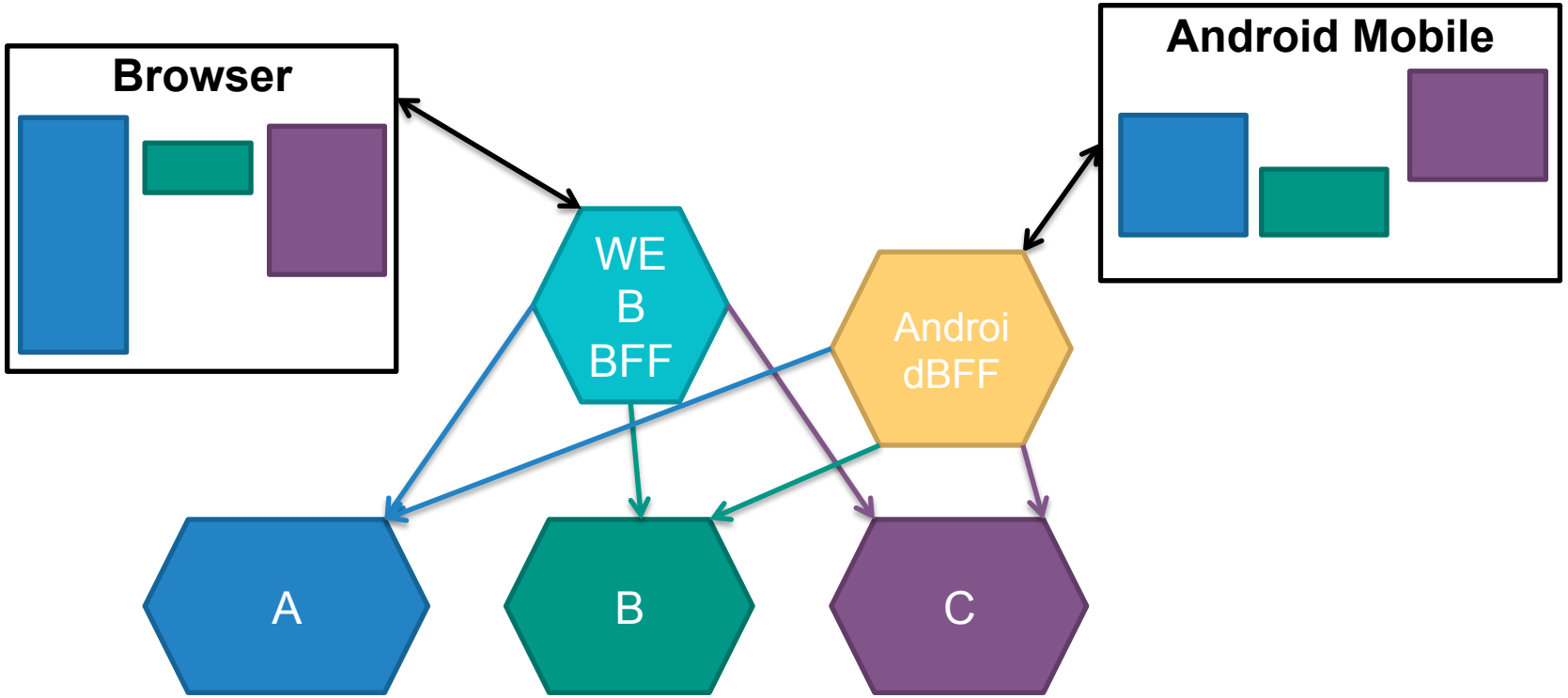
Turn Static Shared Data into Code Accessible via dependency manager

```
public enum CurrencyCode
{
    CAD("CAD"), USD("USD"), EUR("EUR");

    private final String isoCode;
    public final Currency currency;
    public final MathContext displayContext;
    public final RoundingMode ROUNDING_MODE = RoundingMode.HALF_EVEN;
    public final MathContext computeContext = new MathContext(6, ROUNDING_MODE);

    private CurrencyCode(String isoCode)

    public String getCode()

    public static CurrencyCode parse(String isoCode)

    public BigDecimal round(BigDecimal amount)
}
```

Back End For Front End

Extend each UI experience with a dedicated backend component for UI

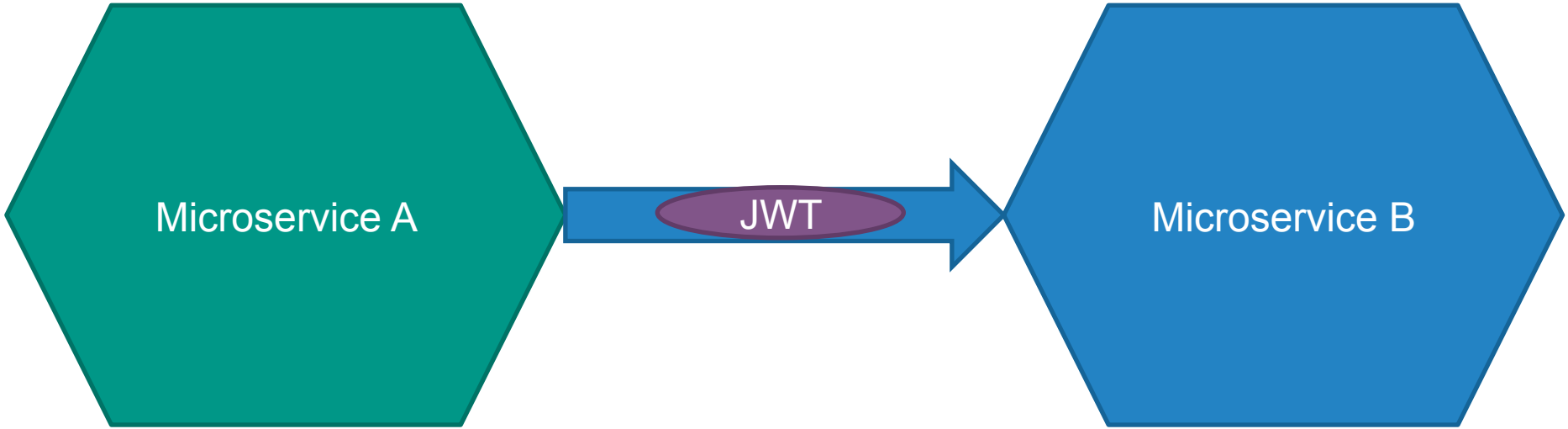
http://samnewman.io/patterns/architectural/bff/



Pivotal

Use JWT Tokens

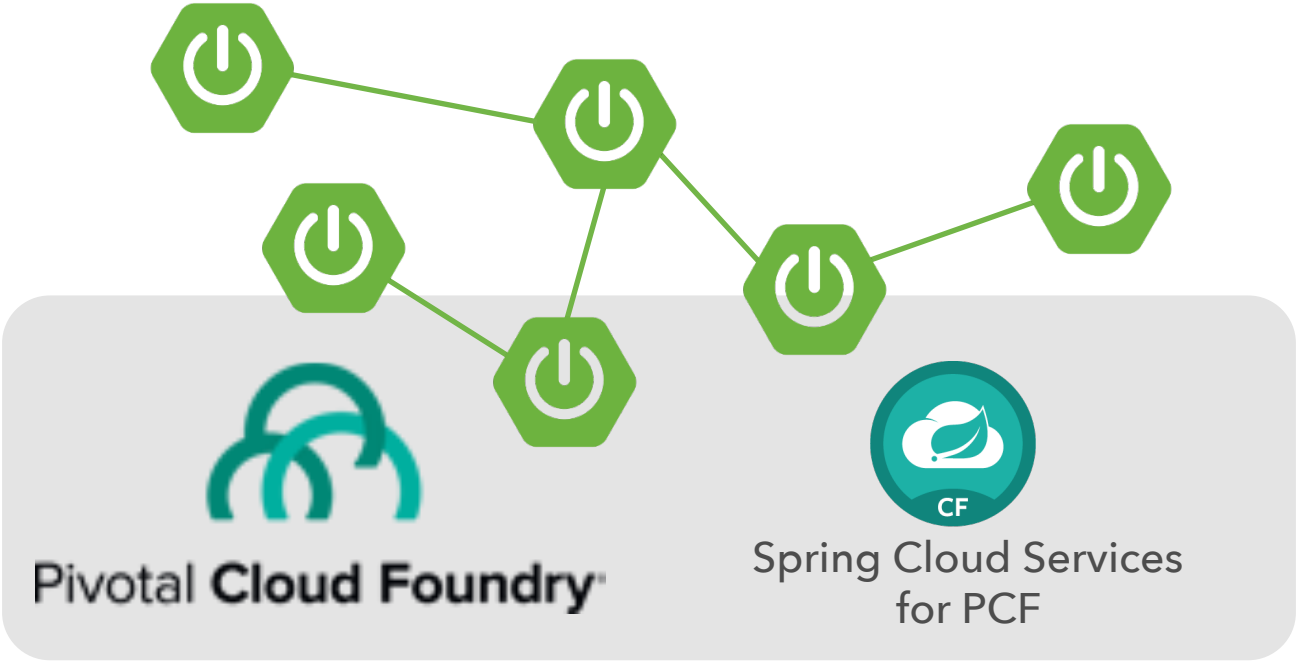
Use JSON Web Token to pass user info between microservices



JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

Distributed Systems are Hard!

Pivotal has got you covered!



Transforming How The World
Builds Software