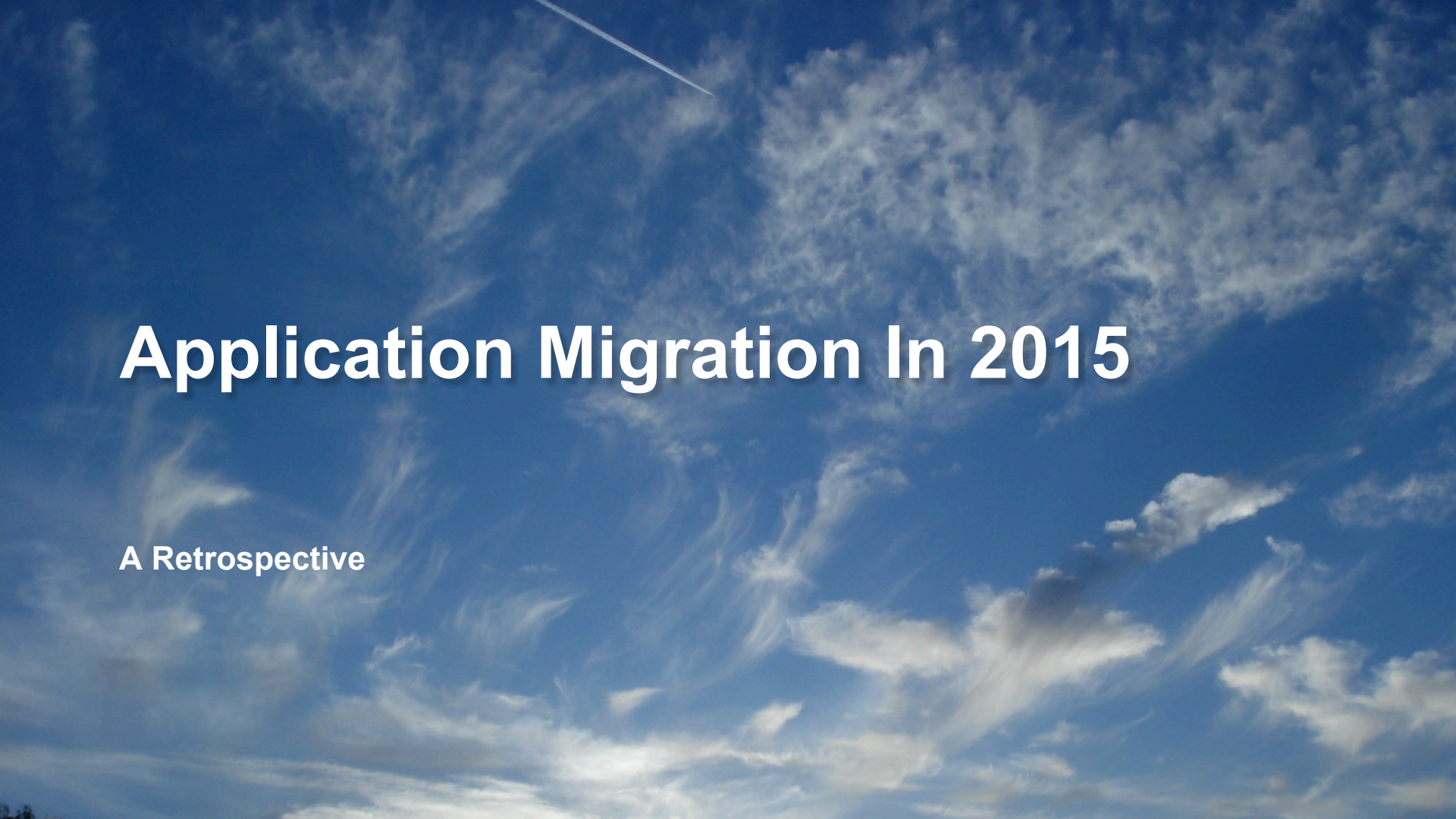

Application Migration In 2015
A Retrospective

WebSphere Liberty Profile Buildpack++

- Packaging offline Buildpack process is deliberately obtuse
- Create all you can eat buffet package or setup a Feature repository
- By default ships with the IBM JDK
- Supports applications (*.ear, *.war, *.jar) and server packages
- Auto-reconfiguration of server.xml based on subset of managed services
- Liberty runtime is decomposable into several packages
- Supports Java EE7 WebProfile + Java 8 by default
- Complicated configuration of the app done via server packages
- Well suited to old school WebSphere Application Server ND apps
- LBP injects VCAP_SERVICES variables into runtime-vars.xml automatically

Pivotal

- Run Java EE apps using TomEE
- Least Evil of all Buildpacks
- TomEE Resources Auto Configuration
  - Targets relational data services connection properties
  - Creates or modifies `WEB-INF/resources.xml`
- Developed in collaboration with SAP (`@violetagg`)
- `https://github.com/cloudfoundry-community/tomee-buildpack`
- Provisions Apache TomEE 1.7.3 JAX-RS profile
- No JCA, JMS, JAX-WS
- Other all-you can eat packages
  - TomEE+ TomEE PluME

Recommendations

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

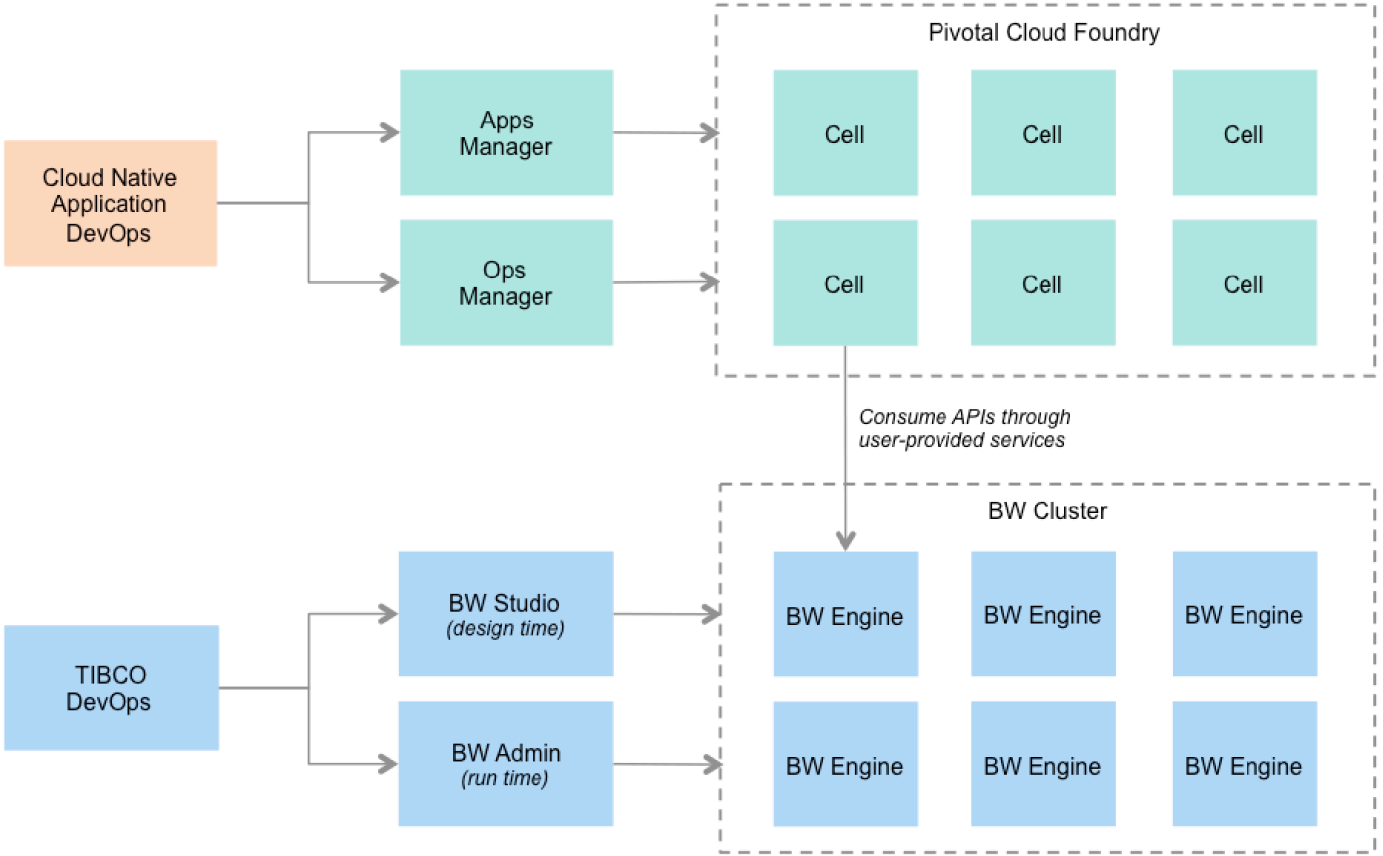
Integration concerns will be handled by implementing the app with Spring technologies(Spring Cloud Stream, Spring Cloud Data Flow, Spring Batch, Spring Reactor)

Support for horizontal concerns will be provided by Cloud Foundry

5 Step Evolution of the ESB to the Cloud

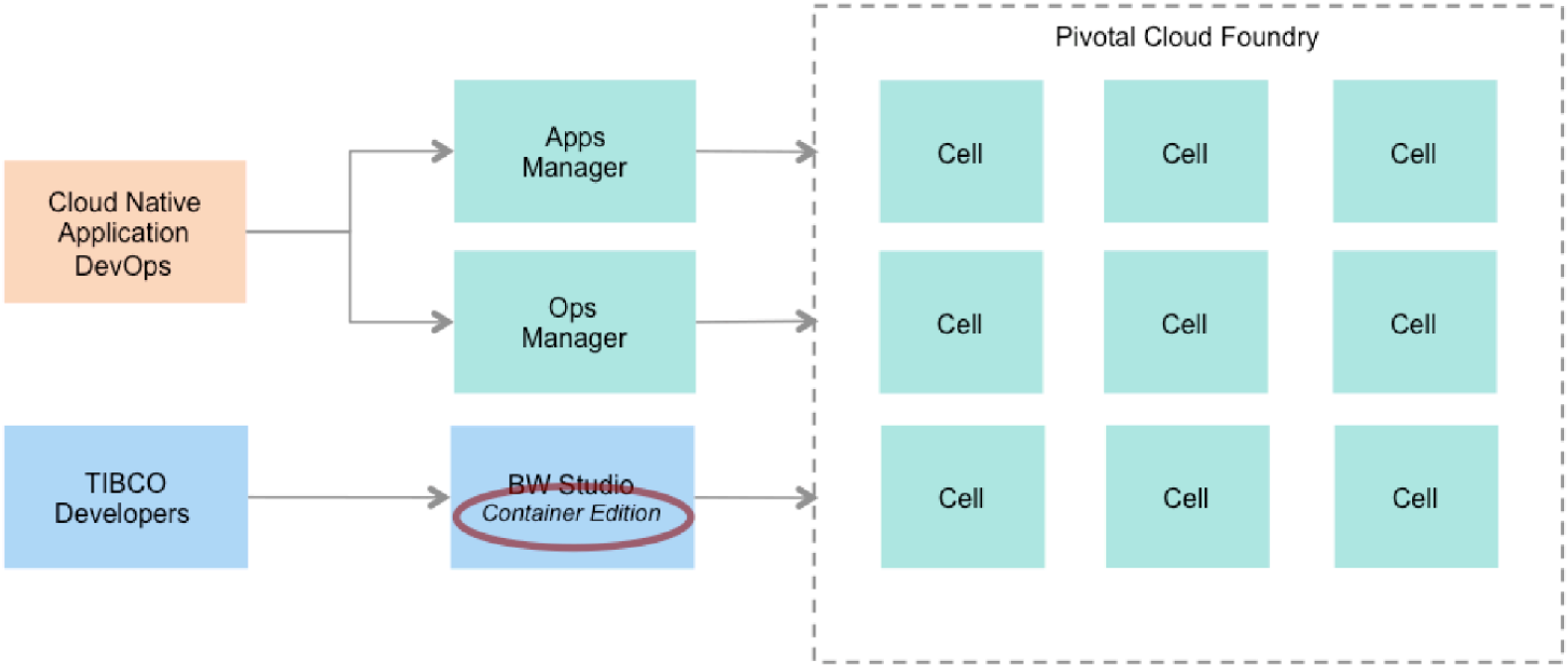
- Phase 0 : Co-exist
- Phase 1 : Lift And Shift
- Phase 2 : Refactor
- Phase 3 : Replace
- Phase 4 : Transform

0 Co-exist



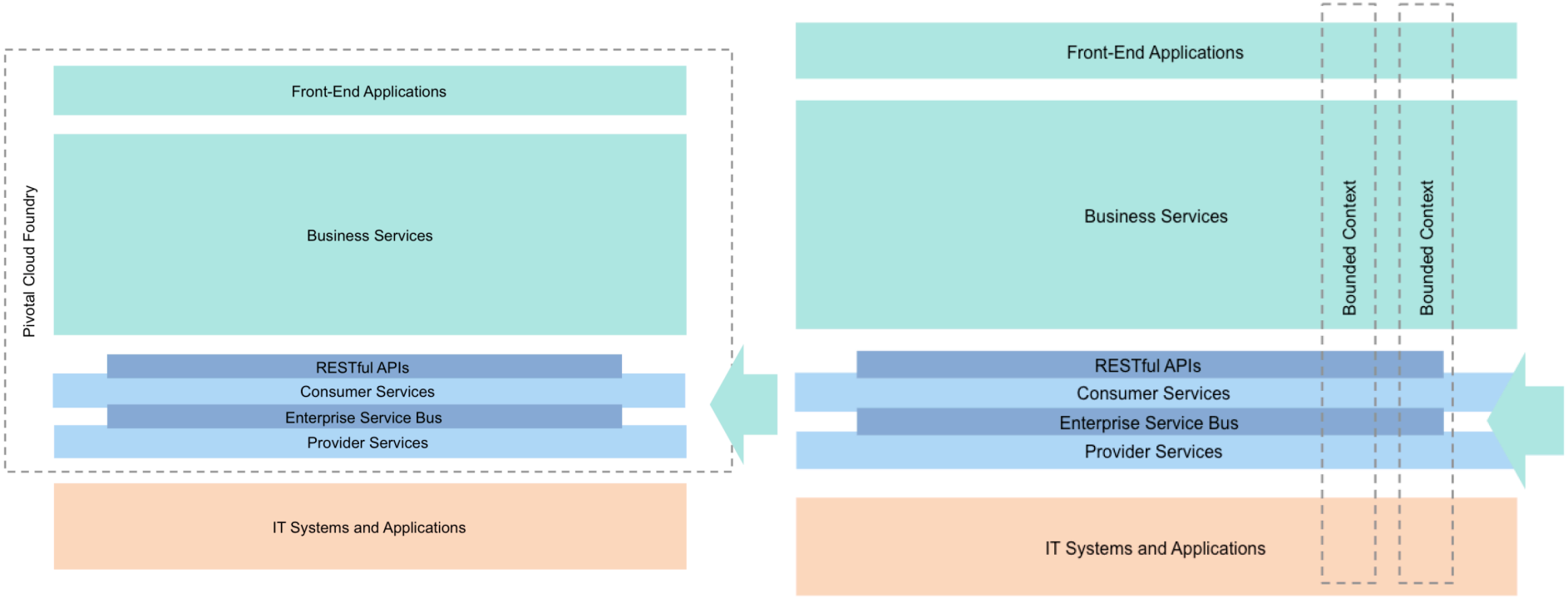
Pivotal

1 Lift & Shift



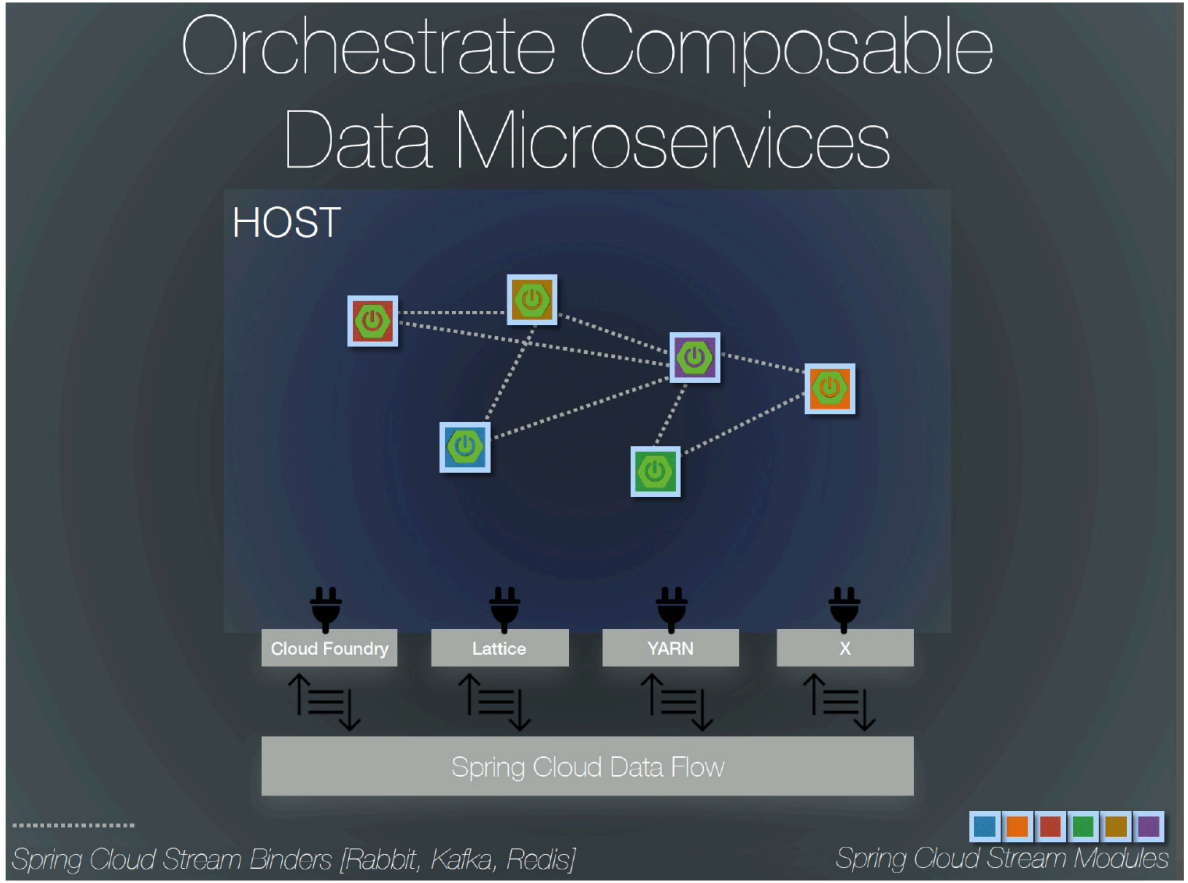
Pivotal

2 Refactor



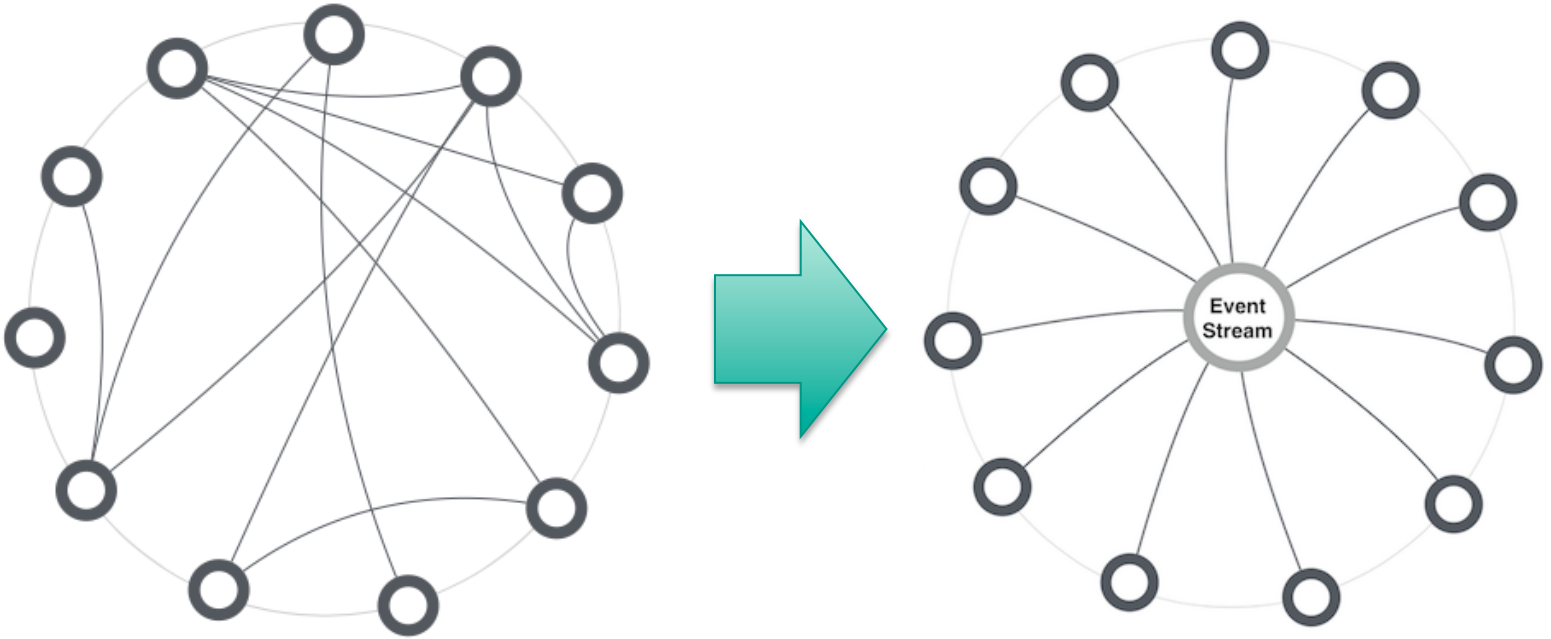
Pivotal

3 Replace



Pivotal

4 Transform



Java Buildpack Experiences

- Most customers fork the JBP
- Perceived Native Memory Leaks
  `JRE_MEMORY_HUERISTICS heap:60, native:25, stack:228k`
  `MEMORY_LIMIT – (HEAP + METASPACE + 100M) between 100 & 150M.`
- Diego App Migration Issues
- Security concerns with credentials in VCAP_SERVICES
- Too much magic between Spring Boot, Buildpack and Auto-Configuration

Pivotal

Traditional Ways of Transforming

- Typically Led by a Large SI on a Multi-Phase Program
- Always Starts with Analysis using Tools and/or Surveys
  - Consultants who use tools and cast a wide net over everything
  - Deliver an expensive report and phased approach to success
- Long projects with big budgets and large batches of work
  - Failure is slow; it takes time and a lot of money to see the problems
  - Business value is slow; measured returns often take years

Which App or Apps Should I Select ?

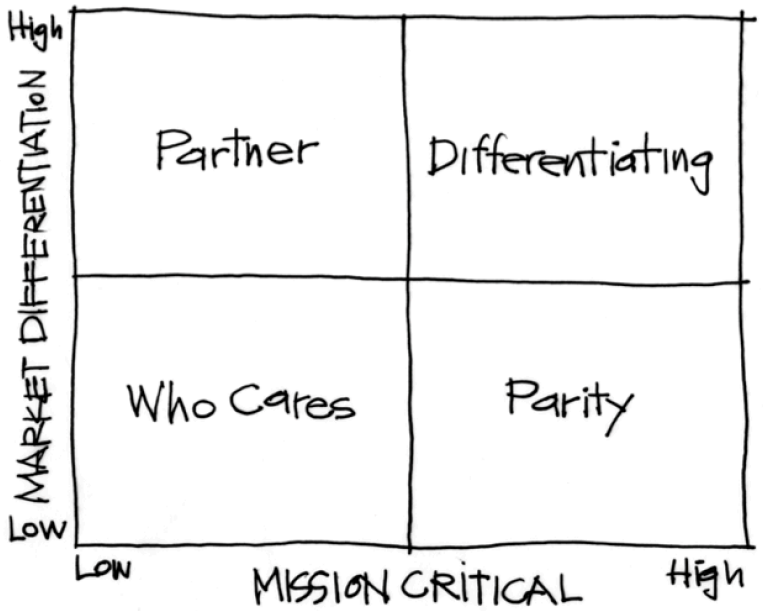
What We Believe

- Detailed Multi-Year Roadmaps Rarely Become Reality
- Break Big Things Into Small Chunks Worked Incrementally
- Keep Your Feedback Cycle as Short as Possible
- Automate Everything You Can (TDD, CI/CD)
- Build New Skills Through Pairing and By Doing
- Failure is Okay, Experimentation Informs Strategy
- Start With “One Thing”

IT DOES NOT MATTER

It is the practice that is more important than the apps itself

Cloud Native	• Micro-service Architecture and Principles • API first design
Cloud Resilient	• Design for failure • Apps are unaffected by dependant service failure • Proactive testing for failure • Metrics and monitoring baked in • Cloud agnostic runtime implementation
Cloud Friendly	• Twelve Factor App • Horizontally scalable • Leverage platform for HA
Cloud Ready	• No file-system requirements or uses S3 API • Self contained application • Platform managed ports and addressing • Consume off platform services using platform semantics



Pivotal

Transform to Cloud Native Applications

TRANSFORMATION

APPLICATION REPLATFORMING

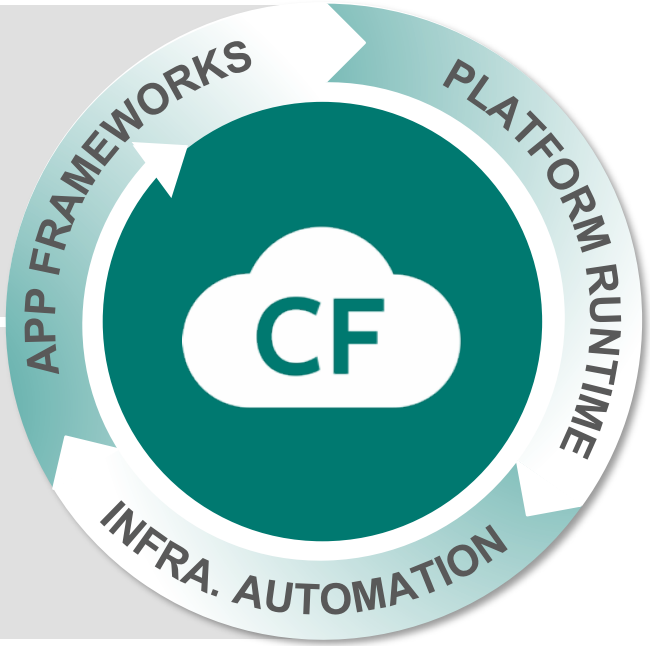
Move 10's of Existing Apps to Pivotal Cloud Foundry With Minimal Rework

- Find Suitable Apps
- Make Minimal Code Changes
- Enable Continuous Delivery
- Move Tested Apps to PCF
- Establish Repeatable Process
- Enable Cloud Native Skills

APPLICATION MODERNIZATION

Modernize a Complex Legacy System Into Cloud Native Architecture

- Plan Data & App Architecture
- Reshape Legacy Code
- Build Microservices
- Enable Continuous Delivery
- Use Pivotal Methodology
- Enable Cloud Native Skills

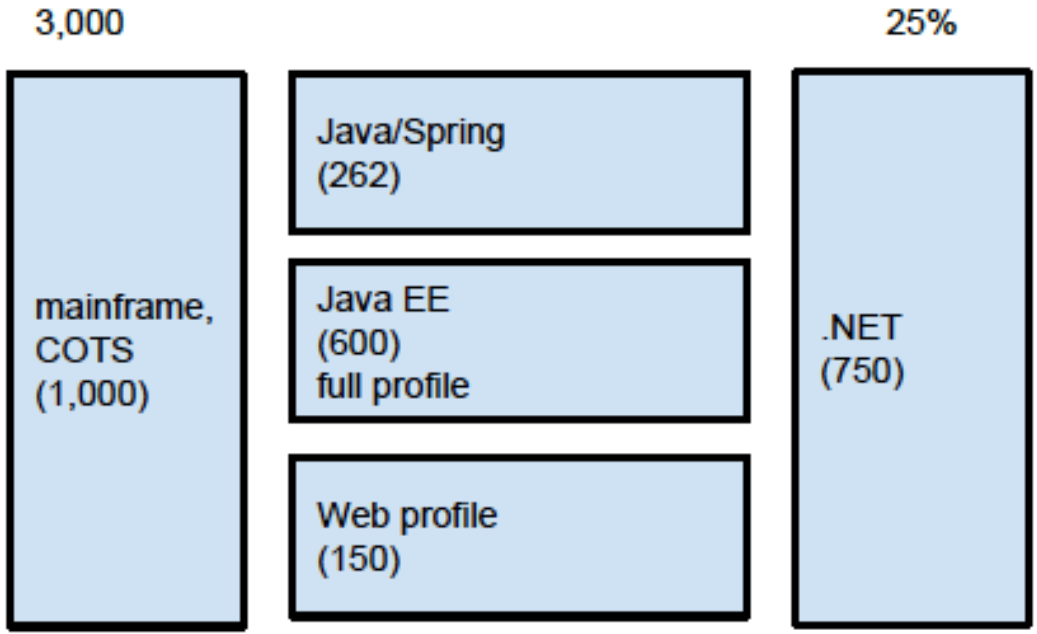


Pivotal

EPIPHANY

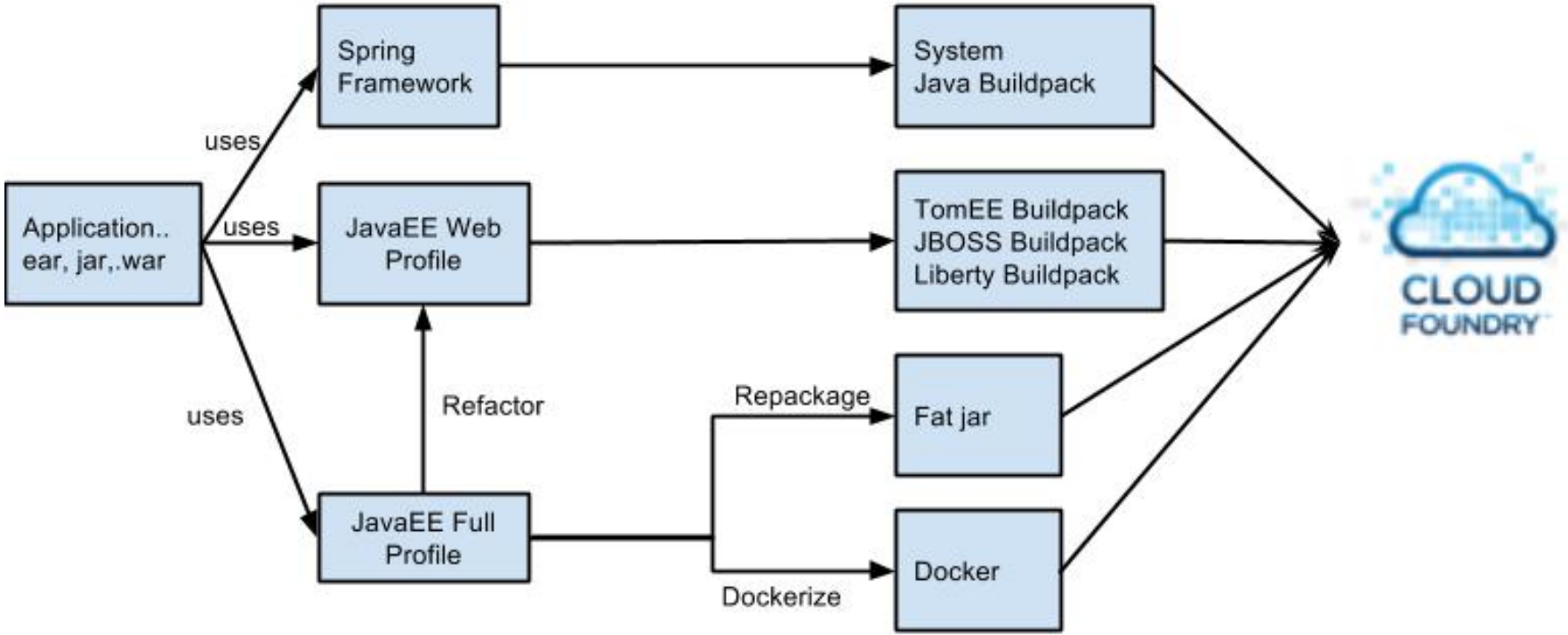
1. Create a Map

Application Portfolio

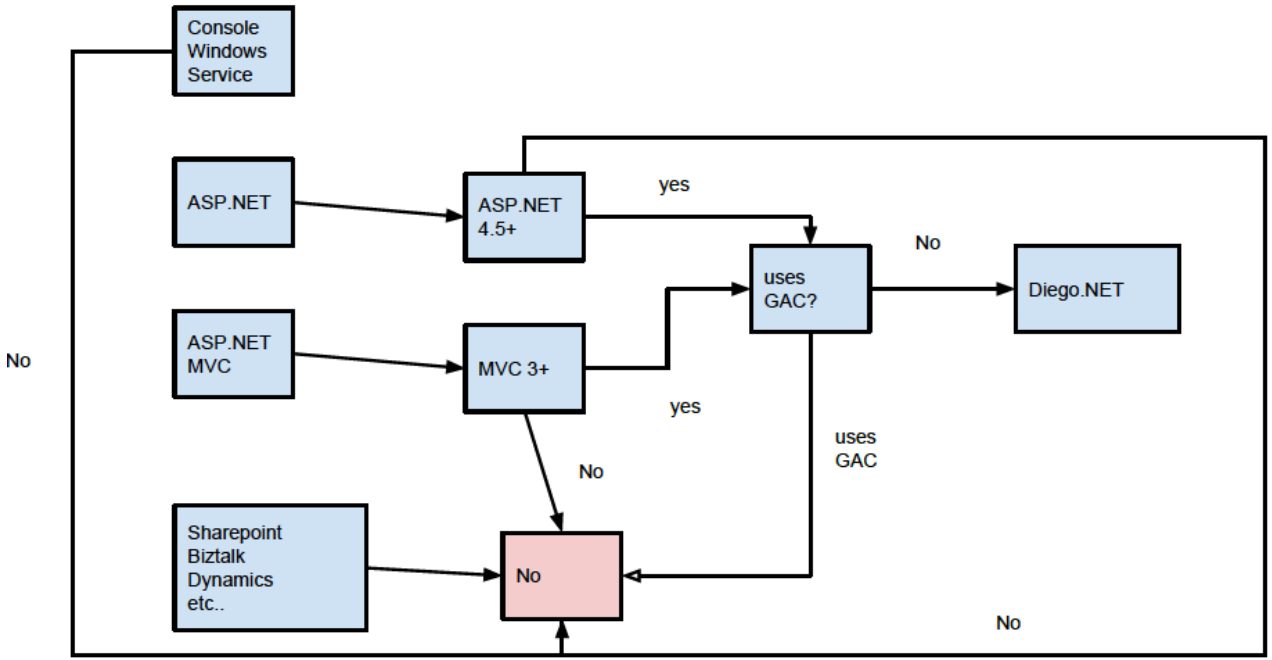


2. Chart a route

Java App Replatforming Flowchart



.NET App Replatforming Decision Tree



Pivotal

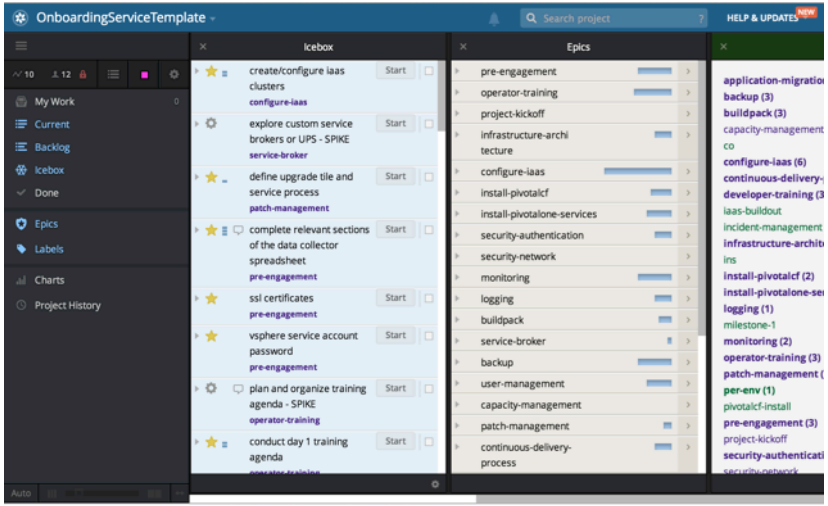
3. Start Driving

cf push cf push cf push cf push cf push cf push ...

- Degree of Statefulness ?
- Reliance on Persistent File System
- Configuration
- Long Startup/Shutdown Time
- Semantic ordering between apps
- non-HTTP Inbound Protocol routing
- Logging to destinations other than STDOUT/STDERR
- Non Standard Security
- App Server based Clustering & Administration
- Distributed XA 2-phase commit Transactions
- Batch Processing
- Specialized Hardware Needs
- OS Based Monitoring
- App Packaging
- Size of the App
- Native Code Libraries

How We Do Projects

https://github.com/pivotalservices/app-transformation



We Use The Pivotal Methodology

- Projects start with a set of pre-kickoff inception activities
- Backlog templates are modified to fit project context
- Scope is shaped and prioritized daily with the team

Our Engagement Toolkit Maximizes Customer Touch

- Pivotal Tracker for project management and reporting
- GitHub for code management and collaboration
- Slack for messaging, search and knowledge sharing

Our Job is Software Success

- Best practice architecture configuration
- Skills and process enablement

WE ACCELERATE SUCCESS BY HELPING CUSTOMERS LEARN BY DOING

I DO > WE DO > YOU DO

Pivotal


Pivotal®
Transforming How The World Builds Software