WHITE PAPER

Pivotal Practices: Application Transformation

By Matt Russell

Pivotal

Table of Contents

Pivotal

Why You Need Application Transformation

Most of the world's developers work on legacy applications: products and services that have been built, maintained, and updated over long periods of time. Pivotal customers who want to move to the cloud often have a complex legacy app portfolio that is tightly coupled and sparsely documented.

Large organizations also develop layers of manual processes designed to minimize risk and ensure compliance. As a result, software releases are often infrequent, high-ceremony events that require heroism and brute force.

Your legacy portfolio can be gradually transformed to cloud-native in order to incrementally reduce time, cost, and operational inefficiencies while maintaining security, resilience, and compliance. Updating and automating processes, in parallel with transforming apps, will reduce the pain and risk of continuously releasing changes while still meeting enterprise-grade requirements.

Pivotal's App Transformation Approach

App transformation is challenging because it requires app teams to develop cloud technical skills in parallel with making major changes to core software development lifecycle (SDLC) processes. Organizations must dedicate their most talented people to the transformation effort, while ensuring that the entire legacy application portfolio continues to operate.

Pivotal has helped dozens of organizations on their app transformation journey. Our approach defines incremental steps that gradually increase the cloud maturity of your app portfolio, the automation in your SDLC, and the knowledge of your team.

Objectives and key results (OKR) measuring cloud maturity, automation, and skill building are defined and tracked for every step. Objectives define where you want to go; key results are metrics to measure the progress you are making in that direction.

Tenets of App Transformation

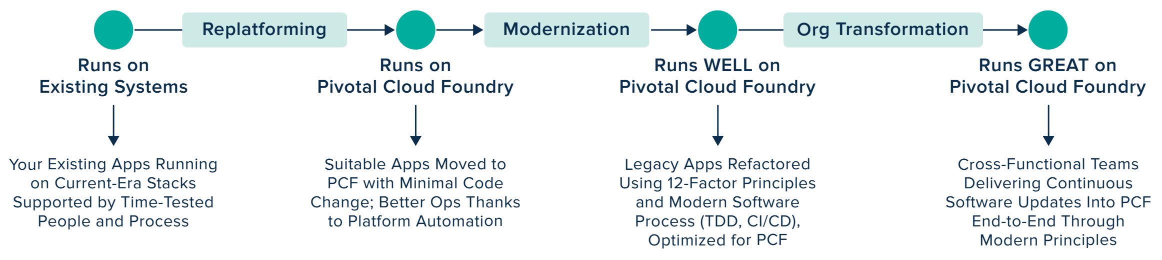
Pivotal has developed four key tenets of app transformation:

1. **Start small.** Even if your portfolio contains thousands of apps, start with a single business unit and a handful of apps.
2. **Learn by doing.** Build skills by transforming the first apps and developing a cookbook of transformation patterns that are then used to transform more apps.
3. **Break big things into small things.** Incrementally transform larger and more complex apps by breaking them down into smaller parts.
4. **Automate everything.** Use test-driven development, continuous integration, and deployment.

Phases of App Transformation

Pivotal defines a cloud-native application as one that runs on a Platform-as-a-Service—such as Pivotal Cloud Foundry (PCF)—and embraces horizontal, elastic scaling. Pivotal customers have moved thousands of applications to PCF over the last few years, from modern Java Spring and .NET Core to legacy systems that span hundreds of containers.

Pivotal defines three phases of app transformation to cloud-native: replatforming, modernization, and organization transformation. The cloud maturity of your app portfolio, as measured using 15 technical factors, gradually increases during each phase.


Replatforming
Modernization
Org Transformation
Runs on Existing Systems
Your Existing Apps Running on Current-Era Stacks Supported by Time-Tested People and Process
Runs on Pivotal Cloud Foundry
Suitable Apps Moved to PCF with Minimal Code Change; Better Ops Thanks to Platform Automation
Runs WELL on Pivotal Cloud Foundry
Legacy Apps Refactored Using 12-Factor Principles and Modern Software Process (TDD, CI/CD), Optimized for PCF
Runs GREAT on Pivotal Cloud Foundry
Cross-Functional Teams Delivering Continuous Software Updates Into PCF End-to-End Through Modern Principles

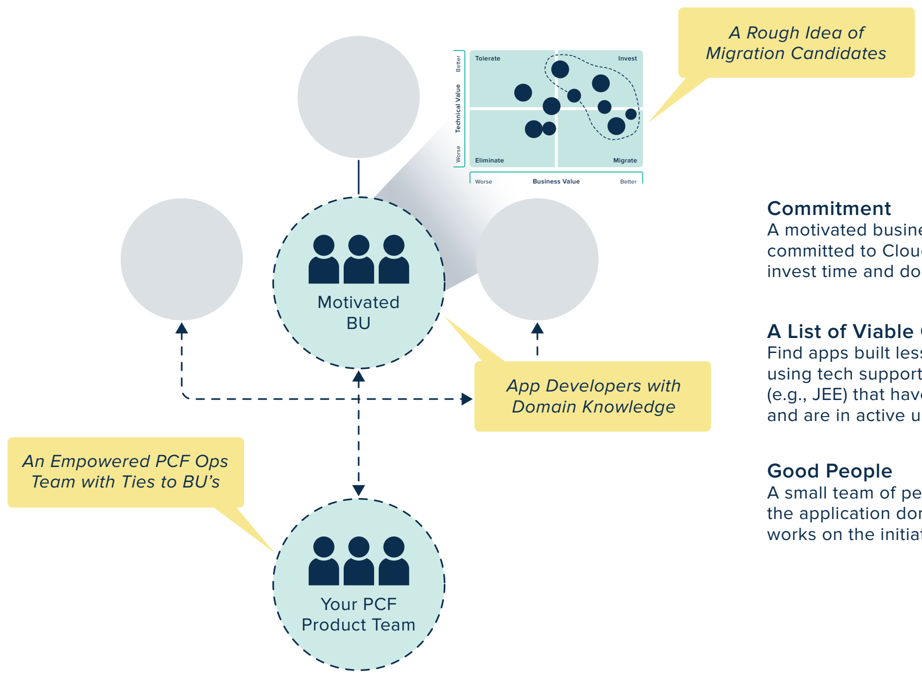
Replatforming involves moving a small set of existing applications to the new platform. The effort required to migrate these applications should be low to moderate, and the replatformed apps should perform as well, or better, than before. However, replatformed apps often do not yet comply with all 15 factors.

During modernization, the level of cloud maturity of replatformed apps increases, while larger and more complex apps are also transformed. More complex nonfunctional requirements—like performance, security, and compliance—will be met during modernization.

Organization transformation reconfigures your organization by forming small, balanced teams that deliver software products from idea to production. Organization transformation is beyond the scope of this document, but is also an incremental process that starts with a single team.

Prerequisites for Your App Transformation Journey

Pivotal's experience has shown that putting the following team and technology prerequisites in place greatly increases your chances of successfully transforming your app portfolio.


A Rough Idea of Migration Candidates
Tolerate
Invest
Eliminate
Migrate
Technical Value
Better
Worse
Business Value
Motivated BU
App Developers with Domain Knowledge
An Empowered PCF Ops Team with Ties to BU's
Your PCF Product Team

Commitment
A motivated business unit with leadership committed to Cloud and a willingness to invest time and dollars in transformation

A List of Viable Candidates
Find apps built less than seven years ago using tech supported by a PCF buildpack (e.g., JEE) that have business relevance and are in active use

Good People
A small team of people that understands the application domain(s), and which works on the initiative in a dedicated way

Organizational Commitment

The app transformation journey often begins locally, within a single business unit, before spreading across an organization. The leadership of this business unit must be motivated to move to the cloud and willing to invest time and money into the transformation.

An app transformation sponsor is usually a CIO or senior executive who controls a relevant budget and is responsible for transformation within the target business unit. The sponsor must have the motivation and the political capital to unblock legacy policy, processes, and other obstacles standing in the way of the app transformation team.

A List of Viable Application Candidates

You need a basic understanding of the cloud suitability of the target business unit's application portfolio in order to choose the first set of applications to migrate to the platform. These apps will be selected based on a combination of technical, business, and economic factors.

An App Transformation Team

You need a small, dedicated team with a product owner and developers familiar with the applications to be transformed. Ideally, the team should also include developers who have experience with the target platform and cloud-native architectures. The team must be empowered to quickly make decisions without lengthy ticketing processes and signoffs.

Product Owner

Represent business interests through backlog prioritization and internal coordination to unblock encountered issues by the team to ensure maximum project velocity.

Project Anchor

Hands-on technical leaders who pair with product owners on backlog concerns, guide technical practices, oversee quality and do technical work.

Developer

Skilled architect/developers who know the existing app and underpinning stacks being worked on as they grow Cloud-Native skills and experience by doing the work.

Recommended roles within small cross-functional teams

Production-Grade Platform Infrastructure

The platform should be operated by a balanced, dedicated platform product team. App transformation projects that start soon after platform installation require frequent interaction with the platform team because platform features (buildpacks, stem cells, etc.) will need to be tweaked to fit application requirements.

An Automated Continuous Integration (CI) Pipeline

At a minimum, a basic CI pipeline should be available. If the platform is mature, existing CI pipelines and features provided by the platform team can be used. These pipelines may be extended by the app transformation team.

How to Transform Your App Portfolio

Pivotal measures the cloud maturity of an app using 15 technical factors. In 2012, a team at Heroku defined the 12-factor app, a manifesto for building applications for cloud platforms. Pivotal has updated the 12 factors to 15 factors, which are prioritized and adapted to use the latest technologies and best practices.

The new factors added by Pivotal include API-first development, monitoring, authentication, and authorization. Pivotal also provides recommendations on how to implement particular factors using PCF (e.g., managing dependencies using buildpacks).

Usually Easy	Moderate Effort	More Effort
V. Build, Release, Run	I. One Codebase, One App	III. Configuration
XI. Logs	II. Dependency Mgmt.	IV. Backing Services
X. Environmental Parity	IX. Disposability	VI. Process
VII. Port Binding	XII. Administrative Process	XV. Security*
	VIII. Concurrency	
	XIII. API First*	
	XIV. Telemetry*	

Effort required to implement Pivotal's 15 Factors

More effort is required to comply with some factors than others. Factors requiring easy to moderate effort are tackled earlier in the app transformation journey.

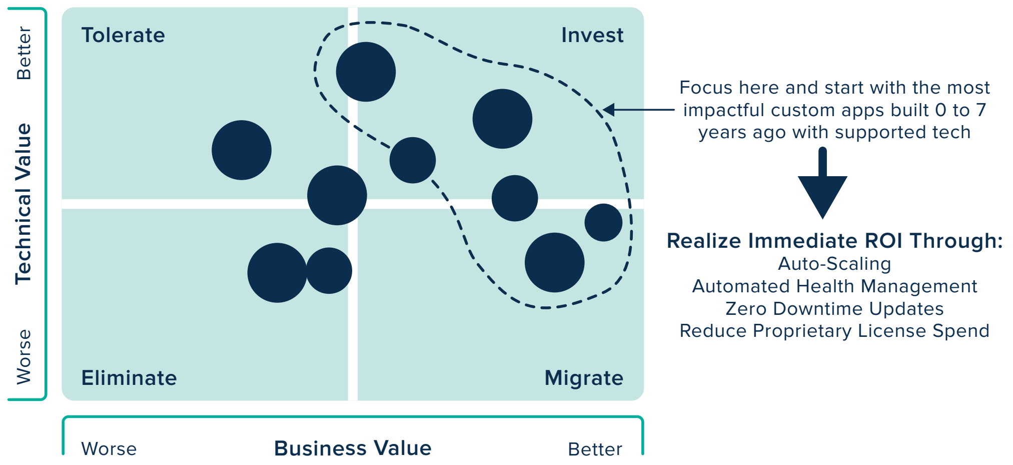
Select Apps for Replatforming

Pivotal recommends selecting a set of 5–10 apps to be replatformed based on a combination of technical, business, and economic factors. The vast majority of applications built within the last seven years can be replatformed with minimal technical effort.

The chosen apps should also be representative of the full portfolio. Companies in regulated industries such as banking and healthcare should ensure that these apps can pressure test as many policy, security, and regulation constraints as possible.

TIME

When the app portfolio is large, a TIME (Tolerate, Invest, Migrate, Eliminate) analysis is often used to prioritize a batch of apps for replatforming. A TIME quadrant map shows the business value of each app versus the technical effort and risks required to replatform it. Ideally, the selected apps should have high business value and require low-to-medium technical effort to replatform. TIME will narrow the scope to maximum 10–50 apps that will then be scored using SNAP analysis.


Tolerate
Invest
Eliminate
Migrate
Technical Value
Better
Worse
Business Value
Worse
Better
Focus here and start with the most impactful custom apps built 0 to 7 years ago with supported tech
Realize Immediate ROI Through:
Auto-Scaling
Automated Health Management
Zero Downtime Updates
Reduce Proprietary License Spend

TIME Map

15-Factor SNAP Analysis

Snap Not Analysis Paralysis (SNAP) is used to score the technical cloud suitability of application replatforming candidates. A simple SNAP is a 15–30-min exercise covering app usage, data architecture, app architecture, and configuration. SNAP results in a replatforming score for each candidate app capturing whether a low, medium, or high level of effort is needed to move it to the cloud. TIME and SNAP results are combined to choose the first 5–10 apps to be replatformed.

Replatform Applications

Make the minimal testable changes to each application required to run it on PCF.

The first phase of app transformation moves the set of applications chosen above to PCF. A replatformed app should work identically, or better, on PCF than on the existing platform, but often conformance with only four to six factors from the full 15 factors is necessary to move the app. In parallel, make any changes required to deploy using the automated CI pipeline.

Work Tracks

There are three parallel work tracks during replatforming. OKR should be defined for each work track.

App replatforming: The application transformation team will work in week-long sprints. A first app can generally be replatformed within hours or days. OKR for this work track typically measure the number of apps or user stories that have been replatformed.

Process transformation: Work through the process and policy issues required to get the replatformed apps to production. This might include release management, environmental issues (e.g., network, firewall, DNS), security (e.g., OSS, credential management, code scans), application (e.g., logging, health monitoring, configuration management), risk (e.g., standards adherence, regulation), and business issues (e.g., business validation, training readiness). This analysis may result in a new CI pipeline or usage of PCF features like logging and monitoring. OKR might include CI/CD pipeline improvements to increase delivery speed or reduce time to production.

Pattern cookbook: Popular technologies like Java and .NET use fairly standard architectures, and employ common messaging patterns. As a result, the technical challenges you solve (e.g., how to modify JBoss code to run on PCF) can be documented in a set of patterns that can be reused broadly across the enterprise, helping to accelerate and de-risk future app transformation efforts. Once 10–20 apps have been replatformed, the patterns identified are usually sufficient to transform thousands more apps. OKR should cover documentation of these patterns.

Automate Testing

Many app transformation teams fall victim to a centralized QA organization that cannot accommodate a faster rate of change. The solution is to automate unit, integration, acceptance, and smoke tests in order to minimize manual testing.

There may be little to no automated test coverage for your existing application portfolio. All new code written should include unit and integration tests. All existing code ported to PCF should include smoke tests (including smoke tests for backing services) and potentially acceptance tests.

Where possible, integrate testing into the CI pipeline so that it becomes a standard part of release management. Push replatformed applications to production and start to transition the central QA role to a more exploratory, functional testing role. Document your new testing practices. They should become standard practice throughout your organization as the app transformation effort grows in scope.

Continuous Integration

Many large organizations have release processes that take weeks or months and have often become complex and opaque over time. Draw a value stream map of the end-to-end path to production (for the replatformed apps only) in order to identify opportunities to reduce waste through automation. We've seen cases where an eight-month release process was initially reduced to a few weeks—and eventually to days—using value stream maps.

A full tool chain must be available in order to push to production. This may be an existing tool chain provided by the platform team, a new tool chain selected by the app transformation team, or some combination of the two.

Modernize Applications

Make testable changes to applications to make them "run well" on PCF, aiming for 15-factor compliance. Break monoliths down into microservices.

Modernized apps run on PCF with high to full 15-factor compliance, including metrics and monitoring, backing services, automated failure testing, and elastic scale. Replatformed and new apps from the app portfolio can be modernized during this phase. Multiple business units may start transforming their own applications using the pattern cookbook developed during replatforming.

Some app teams maintain a single monolithic application rather than a suite of apps. Monoliths are broken down into microservices by moving successive slices of the app to the cloud. New microservices and replatformed slices will continue to work seamlessly with legacy code until the entire monolith has been moved.

It's essential to have a mature PCF platform in place to support modernization. A balanced platform product team should manage the platform and provide increasingly sophisticated automation for testing, patching, upgrades, and release management.

Decompose Monoliths into Microservices

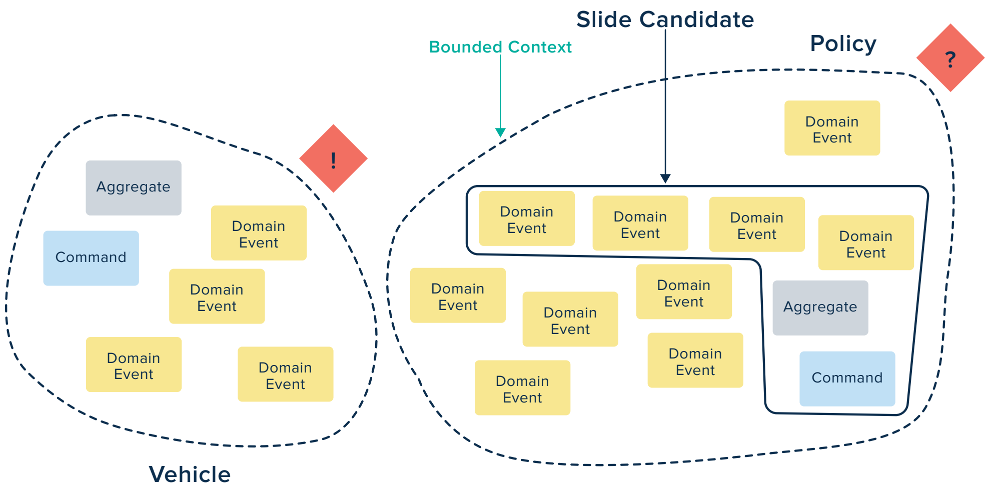
Monoliths are broken down into lightweight microservices that can scale horizontally on a cloud. Monoliths may span multiple subsystems and data sources and dozens of deployables across a network of servers and mainframe backends. Monoliths often also contain elements that cannot move to the cloud without a complete rewrite.

Decomposing an app that may contain millions of lines of code can seem intimidating. Pivotal uses a step-by-step process that starts by building a high-level domain model for the app and ends with actionable work in the form of user stories mapped to releases. Code is reused where possible, refactored when necessary, and rewritten only if it makes good business sense.

Domain-Driven Design

Pivotal recommends Domain-Driven Design (DDD) and Event Storming as practical approaches to defining the boundaries of new microservices. A domain is a sphere of knowledge or activity. Understanding the core business domain of a monolith means understanding the problem the app exists to solve.

A domain model captures a shared understanding of the business domain of the monolith and a shared vocabulary to describe it. Business and technical staff should collaborate to create the domain model. Any sizable enterprise application will have a core domain and several subdomains or supporting domains that interact with each other.



An example domain model

The model describes domain events: things that happen in the business domain rather than in a technical system (e.g., a customer buys a ticket). Triggers, also known as commands, cause events. Aggregates accept commands and process events. To model a business process, events are arranged in a time sequence called an event flow.

A bounded context describes a domain as a group of aggregates with an explicit interface. A complex system consists of several bounded contexts (e.g., in an ecommerce system, you might have order, delivery, and billing-bounded contexts). Bounded contexts or aggregates can often be separated into microservices.

Event storming is a conference room activity used to build a domain model with event flows, commands, aggregates, and bounded contexts.

Slicing

After building a domain model, the monolith is split into vertical slices that can be moved to the platform one by one. Slicing horizontally splits functionality into architectural layers (e.g., user interface, web server, data layer). A vertical slice touches every layer of the architecture but implements only a sliver of functionality.

For example, the vertical slice, “Allow a user to login with a password,” might add username and password fields to a user interface, implement server-side logic, and update the last login field on the database record. Slicing vertically is one of the toughest mind shifts to make for a team new to agile because it requires developers to interact with areas of the app with which they may be less familiar.

Vertical slices are identified by choosing short, domain event flows in the core domain and defining the vertical architectural components required to produce those events. Slice by slice, the app transformation team translates the domain model into a software architecture including microservices, APIs, message queues, etc., that will run on the platform. Finally, a set of user stories is defined and mapped to releases or MVP.

As vertical slices of the monolith are moved into PCF, the system will combine the new microservices, with appropriate traffic routed to them, and legacy code that is integrated using various bridge and queuing techniques.

Decomposition Steps

Here is the full sequence of steps Pivotal recommends to decompose a monolith:

1. Define Objectives and Key Results (OKR).
2. Event storm the app and identify bounded contexts.
3. Pick several short domain event flows in the core domain.
4. Use C4 diagrams to identify a vertical slice of components to produce events in the flow.
5. Brainstorm a target architecture to implement the vertical slice.
6. Perform SNAP analysis to score the effort needed to make the slice 15-factor compliant.
7. Create a backlog of prioritize user stories tied back to OKR.
8. Map user stories to MVP or releases.

How to Know if You're Succeeding

Quantitative measurement is critical and we recommend defining—and continuously refining—OKR for each step that covers process, time, and cost improvements. However, there are some general signals that show that your app transformation effort is succeeding:

- More software releases this quarter than last quarter.
- Release management efficiency: Lower lead and process time, fewer steps and handoffs.
- Improved operational metrics for transitioned apps: MTTR, MTBD, support tickets, etc.
- Improved security: Faster patching, zero downtime upgrades, etc.
- Infrastructure usage: Higher density compute, auto-scaling and licensing reductions.

Pivotal helps organizations all over the world to build agile app transformation plans. However, we do not have all the answers; we learn as much from our customers as we teach them. The successful app transformation journeys of Pivotal customers such as Liberty Mutual have informed much of the above advice. If you have insights about app transformation you would like to share, please contact us at **info@pivotal.io**.