THE CLOUD NATIVE JOURNEY

Michael Coté

Pivotal

REPORT

The Cloud Native Journey

So, How Does One Eat The World With Software?

The term "Cloud Native" has risen in popularity of late and is being used to describe organizations that are using new cloud technologies and techniques such as Pivotal Cloud Foundry and DevOps to dramatically change how their businesses run. Rather than treating IT as a supporting function, these Cloud Native enterprise are using IT as the core enabler for their business models.

In doing so, they are not only making what were previously analog processes "digital," but they're enabling the business to take full advantage of the flexibility of software and, thus, making their businesses' processes more flexible. Marrying up your business to software may seem risky: after all, as the Standish Group extensively covers, we've seen high rates of software projects struggling, if not outright failing, over the years. In contrast to traditional development technologies and methods, the new Cloud Native approach that focuses on smaller batches, is supported by cross-functional teams, and is delivered on highly automated cloud platforms improves the overall quality of software development across industries.

Most of Pivotal's customers are organizations looking to take advantage of this Cloud Native approach and are on various stages of the journey to become Cloud Native Enterprises. This has given us the chance to observe how numerous enterprises are transforming and how they use custom written software to run their businesses. These organizations are responding to the need to become a software defined business—the pressure from "Silicon Valley" to use agility, cloud, data, and devices to disrupt how business is done. Those pressures are well covered elsewhere, and I wanted to start collecting the highlights of *how* these companies have been changing and what they're doing. This concept is often called the "journey," which is a euphemism for "it takes a lot of hard work and time—you won't just succeed overnight." Similarly, building your own home or raising children is a "journey." There are no easy answers, things "go wrong" often, but setting up a process of continual learning and improvement is what ends up addressing the problems, resulting in custom built houses and functioning adults.

> This white paper explains what Pivotal customers and other organizations have been doing to make sure their "kids" stay out of jail. First, I'll briefly discuss the goal—the reason companies are looking to become Cloud Native enterprises. The rest of the white paper looks at three modes of operating, new projects ("greenfield"), existing projects ("legacy"), and, finally, transforming the company's culture. My hope is that you can use this overview to understand why the "journey" matters, what it looks like, and how to start planning for the journey to becoming a Cloud Native enterprise.

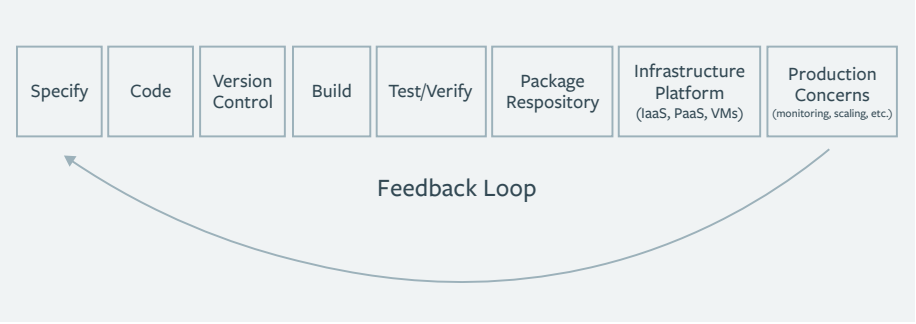
PART 1

The Goal: Shorter Cycles To Increase Quality And Grow the Business

Our customers want to use Pivotal Cloud Foundry to improve how IT is used within their organization. To run and improve their business and their customer experience, they must become really good at writing and managing their own software. To do this, they are looking to improve the quality of the software life-cycle—from thinking of a new idea to developing it to running it in production. As Andy Zitney of Allstate puts it, their goal is to think of a new software idea on Monday and have it running in production by Friday.

I like to reduce the goal down to speeding up the software delivery cycle, as shown in a diagram I use a lot:

Continuous Delivery Pipeline



Your goal is to make moving code through that pipeline, safely and reliably, as quick as possible. A good goal is every week, but, if you can master small batches and put in all the process and technology rigging needed, you should shoot for daily deployments.

This is the goal to keep in mind as the short and medium term goal for your organization—reducing the full cycle time it takes to get new features into production. The long term goal is to improve not only the quality of your software delivery process, but the quality of your business.

But, first, you need to get your software delivery process in order.

CHECK-IN TO SEE WHAT SITUATION YOUR SITUATION IS IN

As mentioned, we see three types of organizations embark on this journey. To get started, your company must know where it is with respect to capabilities and obligations. By knowing the current state, you understand where you are starting the journey, and it helps you determine the next steps and the problems you will likely encounter. Ultimately, it guides your strategy.

To date, I use these three buckets to describe the types of companies—Greenfield, Legacy Management, and IT Transformation. They're not perfect, and often overlap: most organizations will have all three going on to a degree.

GREENFIELD DEVELOPMENT

These groups have very little existing IT process or staff to work with. Some might say, to deal with. They're given freedom to do anything they want and are often starting from nothing. Most of these companies are smaller, but sometimes they're a team at a large company that is given the freedom to completely ignore everything and start fresh—a regular skunk-works situation.

They want a platform for developing net-new applications and have little concern about integrating with or managing existing applications. This is a fun group. Their challenge with using Pivotal Cloud Foundry (or any cloud platform!) is that they get excited about building their own custom built platform instead. They resist the need to delegate infrastructure management and cloud orchestration concerns to the cloud platform.

And why not? These teams like building software, but building their own cloud platform tends to trap the team into becoming cloud platform builders. Instead of building the business applications their organization needs, they spend much of their time and money re-inventing an existing wheel. These teams are often focused on "day one" problems (building and finally deploying the first version of the application), and they put off the tediousness of "day two" problems (managing the application in production, ensuring that it's easy to update as needed, and providing enterprise governance and controls if they're stricken with that kind of maturity).

To me, the challenges here are all about establishing a process of moving fast without building up too much technical debt. Teams must also realize that they're being tracked and closely monitored for their success. They need to "market" themselves appropriately. If these initial greenfield projects work, the rest of the organization will likely want to replicate the efforts. If the efforts fail, the thinking behind these projects will be quietly swept under the rug. That may stifle "going cloud" for the rest of the organization.

LEGACY MANAGEMENT

These groups have a full portfolio of existing IT applications to maintain and grow. There are many "obligations" owed to the past, and they often operate under many more constraints than the other two types of teams. Their challenge with switching to

a Cloud Native approach is planning out how to methodically "slice off" parts of their existing applications and re-platform them as Cloud Native applications. These teams are metaphorically tasked with rebuilding the jet engine mid-flight.

These legacy teams are often looking for fixes to lingering, systematic problems they have (the relational database can no longer scale) and the effects of too much technical debt ("our system is so burdened and fragile that it takes weeks to do a release"). These teams have a challenge—all of their time is taken up simply keeping applications up and running. This leaves little time to work on new functionality. Worse, when there is time to add in new functionality, the legacy system is so ponderous (and often poorly understood) that change takes much longer than it should.

To me, the challenges here are about risk management—knowing when to keep doing "the wrong thing" despite the allure of "the new thing." Eventually, these teams have to choose either to "give up" or "go for it." If the risks of making changes are too high, they must quarantine the applications in questions. Or, if the risks seem acceptable, the teams have to start systematically re-platforming and re-writing the backing services and applications themselves.

IT TRANSFORMATION

> *"With Pivotal we have minimized our innovation cycles and can now respond to changing market demands faster than ever before, both of which allowed us to develop a profound software culture in our transformational journey into a digital company."*
>
> —Christoph Hartung, Head of Connected Cars, at Mercedes-Benz

Somewhere between greenfield and legacy, there are groups tasked with actively changing how IT is done at their company. These teams are often driven from the top-down and own the entire application portfolio at their organizations. They're responsible, at a portfolio level, for building new applications and also maintaining existing ones. What makes them different than just a combination of greenfield and legacy teams? They are actively looking to convert most everything over to the new platform, not just keep the old things up and running or let greenfield applications run without any constraints. They want a common, shared platform for doing everything. I think of CoreLogic as the earliest, most representative Pivotal customer here, but as you can imagine, most Pivotal Cloud Foundry customers are looking at this path to meet their business goals. Whole industries, like automotive, are taking a Cloud Native approach to changing how their business operates.

Getting to that kind of software culture requires a common platform and product development philosophy for all of their applications. They are, indeed, on a long, multi-year journey to move everything, new and old, to that platform. The challenges they face reflect this. First, they have to "rein in" the new application developers and have them develop on the standard platform rather than using whatever they like. Second, they have to push the legacy teams to be more aggressive in changing the architecture of legacy applications. Sometimes, they need to retire legacy applications entirely, completely rewriting them. To me, the challenges here are all about change management for the application portfolio and the "culture" of IT in their organization. This is the hardest thing an IT department does. However, this bucket has the largest payoff—naturally, because it's going after everything.

In the rest of the paper, I'll spend time detailing different strategic and operational tactics for the three types of journeys, and I'll also look at some of the higher-level technical changes that Pivotal Cloud Foundry customers are doing to be successful. At the end, I'm hoping you'll get a sense for what it looks like to be a Cloud Native enterprise.

PART 2

The Purity And Tyranny Of A Blank Screen: The Greenfield Journey

WHAT'S GREENFIELD?

When I think of "greenfield," it describes application development projects that have little, if any, connection to existing systems. Unlike "legacy" situations (which I'll will cover next), there is no existing code to evolve into the next release. The product team has the thrills of blankness in front of them—a blank white-board to start planning, blank post it notes to start tracking requirements and stories, and blank screens waiting to be filled with code.

In many ways, this is the dream situation. Developers have little in the way of constraints and no technical debt. We read the most about these types of projects, but, as anyone who regularly talks with "enterprises" will tell you, it is the least common in the field. However, getting the projects right is often required for broader IT transformation. Your success with greenfield projects will give you credibility and experience needed to do more. At the same time, failing at them can quickly slow down any future planes.

Some of the ideas here will apply to all types of cloud journeys (like having full test coverage) but I've pulled them together here because they are particularly applicable to greenfield projects.

THE THREE STRATEGIC GOALS OF GREENFIELD PROJECTS

Before I get into some general advice for greenfield projects, let's go over their goals. The overall goal is to deliver a product, but we need to think beyond that. You need to think ahead to what you do after the greenfield journey. How can you ensure that your greenfield work helps improve and transform the whole company over time? In that "what's next?" sense, the goals of doing greenfield projects are:

1. **Earn trust by showing success.** Pick an appropriate, valuable, but low risk project to experiment on and learn from. It will be hard to continue expand into larger transformation work if you fail.

2. **Establish a continuous delivery pipeline and cloud platform.** These will be some of the core technology assets you will re-use after your greenfield experiences. Once setup, they will enable you to release smaller batches of code on a more frequent basis—one of main operating points on the Cloud Native style.

3. **Learn by doing and educate the rest of the organization.** Much of your work is the hard task of learning a new way of planning, managing, working, and interacting with customers. "Culture," as the kids call it, isn't easy.

GET YOUR HEAD STRAIGHT & SET REALISTIC EXPECTATIONS

Using Cloud Native technologies is the simple part—changing how your organization uses those technologies is the hard, grueling part. Give yourself the space to learn, fail, learn, fail, and start succeeding. Doing this with a large project is dangerous and not required. Start with a small project and grow from there. Overly ambitious, amateur knife throwers get cut a lot, or worse, tend to impale others while learning.

While the benefits of a Cloud Native approach are wondrous with its innovate new features and power to deploy software safely multiple times a day (beating the competition!)—it takes work to get there. Plan out a few quarters. Allocate resources to the greenfield project, but stay conscious of the fact that you're learning. Set expectations for everyone, including upper management, that things will not go perfectly at first.

SELECTING YOUR FIRST PROJECTS

If you're a small team, or a small company, selecting the project to work on is likely easy. You probably just have one application, so select that! In larger companies, there are often 100's, if not 1,000's of application and projects to pick from. Pick one with direct value to customers—one where you will get feedback once you deploy it (i.e., people will use it a lot, it won't just be shelf-ware). You also want to pick a small enough project so it can get into production in a short amount of time, let's say 3 months at most. Finally, if things go poorly, you want it to be a somewhat low profile project so you can sweep it under the rug.

This last point is no doubt contentious to the purer minded out there, and I can sympathize. We should strive for truth and transparency! I'm sure you're lucky enough to be in a corporate structure that rewards the value of failing (read—learning), but think about your peers. Many are not so lucky and work in caustic corporate cultures that punish any type of failure by "promoting" the former VP of "Having a P&L" to VP of "Special Projects." In such environments[1], you're given the chance to advance to the next place on the board by success. So, you will want to pick projects accordingly. Of course, "failing fast" should help you change said caustic corporate culture around...hopefully. While a bit dated, the 2010 booklet, *The Concise Executive Guide to Agile* has helpful details for picking your initial projects.

Alternatively, there's another project type to choose—what I like to think of as a "moribund" project. It fits all the criteria above but already exists and just needs to be shown some love. One of our customers, Humana, did this. Their Vitality project wasn't getting the engagement levels they wanted—just 3% of potential users. They wanted to triple engagement. As they detailed in their keynote at this year's CF Summit, they increased engagement to over 30% after reviving it with a more agile and Cloud Native approach. This success was parlayed into two other, small-but-important projects and the company is now on the path to transform how applications are delivered company-wide.

STAFFING THE PROJECT

Team organization is a big change with Cloud Native approaches. It's easy to dismiss the funny-sounding talk of "two pizza teams" that Cloud Native pioneers like Amazon have recommended for so many years, but the sizing and composition of the product teams is critical.

Not only do you want to keep teams small—somewhere between 6 and 12 people—but you want them to have all the skills needed for the full life-cycle of the product, including development, testing, design, and operations. Keep in mind—this team size will be possible because the infrastructure provisioning, configuration, and ongoing management will be powered by a highly automated cloud platform.

The team needs to be dedicated to the project, full time. For many companies, this is a hard pill to swallow. Everyone is so busy with various projects that they can't be dedicated to just one thing! Well, if operating in that way is getting you the delivery speeds, product quality, and uptime you think you should be having...perhaps there's no need to read on... sounds like you have it all figured out! To put it more gently, dramatic Cloud Native results require some dramatic changes. Dedicating a small, skilled team is key, and it works.

1 For more insights on how to run your OODA loops in caustic cultures, see Tina Nunno's excellent *The Wolf in CIO's Clothing*.

To cite one of many examples from Pivotal's customer-base, CoreLogic has proven that this mentality gets results:

> When planning the first product developed on Pivotal Cloud Foundry, CoreLogic allocated a team of 12 engineers with four quality assurance software engineers and a management team. The goal was to deliver the product in 14 months. Instead, the project ultimately required only a product manager, one user experience designer and six engineers who delivered the desired product in just six months.

The other "people concern" is assigning one individual as the key stakeholder in charge of the project. At this stage, you will need someone who can be decisive and cast the deciding vote on any given decision to keep things moving. If you're moving from a culture where people are used to being told exactly what to do (most corporate cultures, I'd argue), without a definitive person in control, people will fall back to their "bad behavior" of doing nothing when there are not clear directions. Thus, if there's not an individual who has ultimate responsibility, you're likely to see lax discipline and attention to roadmaps. As you move into the later stages of going Cloud Native, your organization will take on more and more shared responsibility driven by clear goals from management across departments, but, in the greenfield stage, it's good to ensure that everything is going well by assigning an individual to that role.

USE THIS CHANCE TO CHANGE YOUR PROCESS

Many large companies are looking to do Cloud Native development, but they are not following best practices for agile development, continuous delivery, and DevOps. Indeed, a recent Institute for the Future study of 3,600 business leaders showed how only 25% of respondents felt their companies were agile enough. This of course leads to a declining value of IT to business. Read: not good if you're IT in these software eating the world days!

So, if you're lucky enough to take a fresh start, be as strict about adopting agile practices as possible. They are best practices because they work. Start by defining a weekly release cadence, using whichever variant of agile works for you, practice open, shared project management (likely with a Kanban-inspired tool like Pivotal Tracker), automate as much as possible, and, most importantly, follow the continuous learning loop of trying something new for a short time and evaluating if it's working or not. Psychologically, large organizations tend to view "process interrogation" as a sign of management weakness. This couldn't be further from the truth! High performing IT departments are continually studying, tweaking, and learning how their process works. Low performing IT departments are rigid and never question process.

To that end, I recommend learning through immersion, for example, by working with Pivotal Labs. We can thoroughly soak a team in agile product development, and many customers rely on introducing our culture to reinforce new development habits. There's

a wealth of information out there on this model: consume it and try it out. This is your chance to try something new if you're in that 75% of people who feel like their organization is not agile enough.

The recently published *Leading the Transformation* has helpful advice on figuring out the wicked problems of test coverage across different domains if you find yourself stuck. Also, keep in mind that the highly automated and speedy capabilities of a Cloud Native platform will also speed up your ability to increase your test coverage. In general, there's little excuse to skimp on automated tests, and you'll certainly damage your overall project agility if you do so.

CLOUD NATIVE DEVELOPMENT: 12 FACTOR APPS

With the benefits of that blank screen staring at you, you can start your application architecture and development in true, 100% Cloud Native style. In addition to enabling all of the deployment speed and operational benefits of a Cloud Native approach, a greenfield situation allows you to road-test new development styles. The guiding principles for development style are encapsulated in the 12 Factor App.

The 12 factors describe design principles that ensure your application will be Cloud Native. For example, you can't depend on local storage, configuration should be injected into your application on-demand rather than shipping with it, and apps should be built as stateless processes, to name just a few. It's important to hew closely to these principles because they form a sort of "contract" between the cloud platform and your application. If your application obeys the contract, it can be run and managed properly by the cloud platform.

AVOID TECHNICAL DEBT WITH STRICT TESTING

Until you experience the effects of near 100% test coverage, it's hard to appreciate the worth of all that work. Automated test coverage won't solve all your problems, but it will definitely make it much easier to move fast, refactor your code, and add new functionality to your application. If it takes you days (if not weeks!) to test changes, even just a line of code, you're going to slow down your release process. Also, without tests, you'll find it hard to make changes to your code: because you can't quickly and accurately test changes, you'll start to ignore older parts of your code base for fear of breaking functionality. This turns into a spiral of building up technical debt as more and more parts of your application are surrounded by signs that say "don't touch!" Very quickly, you'll be immobilized and unable to change your application's behavior and add in seemingly simple changes. Thus, in a highly automated Cloud Native application lifecycle, you need the ability to quickly test nearly everything. As you're starting with a new project, don't cut corners by cutting down tests. Getting broad, accurate testing in place is key to moving fast and helps set an example for the future.

CLOUD NATIVE ARCHITECTURE: MICROSERVICES

Microservices are the other important development style consideration. This architectural style breaks down applications into as many independent services as possible. It also focuses on the end goal of breaking process and technology dependencies between each service, making them stand-alone. These services could be things like displaying "you might also like" recommendations, storing a user's profile information, or providing a service that queries your inventory system. The idea of decomposing your application into subsystems is not new; what's new about microservices is the emphasis on how decoupled and small those services are not only technologically, but also with respect to people and process.

In the large view, a microservice exists and can evolve on its own. The development team responsible for it can use whatever technologies to implement it that they choose and can follow their own road-map. As with much Cloud Native thinking, this focuses on speeding up release velocity. The size of the service is, well, "micro." Some joke that all the code for a good microservice should be about as tall as your head if held next to a screen. Others size it by how long it should take to re-write the service—3 to 5 days at most. The point is that to maintain agility, you need to keep each component independent and small enough to comprehend quickly and work with easily.

Taking a microservices approach is the right decision in the long term. In the short term, though, because of the drawbacks of microservices, it can be helpful to do your initial project with a more traditional, monolithic approach. This approach allows you to start learning-by-doing a little bit, and then ramp up for your second project. The larger the project, the more important microservices are for project stability and ensuring sustainable agility.

That said, many of the drawbacks of microservices are addressed in Pivotal Cloud Foundry. For example, if you're developing in Java, the Spring Framework has a lot to offer that's tightly integrated with Pivotal Cloud Foundry. Spring Boot can be used to create a self-service, microservices-oriented "tool store" for developers, while Spring Cloud can be used as the foundation for your microservices approach. Check out Matt Stine's book on Cloud Native architectures for more detail.

STAFF GROWTH & RETENTION BY PARTICIPATING IN YOUR ECOSYSTEM

Eventually, it may be hard to retain the key staff responsible for this transformation. With their track record, they will be highly sought after by competitors and technology companies. To abate this, it's important to start "making stars" of these key staff members. It is actually easier than it seems. Just let them engage more publically about what they've been doing.

While some employees will still end up leaving, many more will stay if you allow them to start contributing to open source projects and the ongoing conversation in the Cloud Native community. Equally important is finding new staff to fill your Cloud Native needs. By participating in activities like open source projects and conference presentations, it'll also

help address recruiting needs. And, of course, getting more involved in open source will also get you closer to the code you use and, if you're lucky, improve the quality of it. The point is, you want to become part of the overall community, which means participating as much as possible. Doing so will help tremendously with staffing.

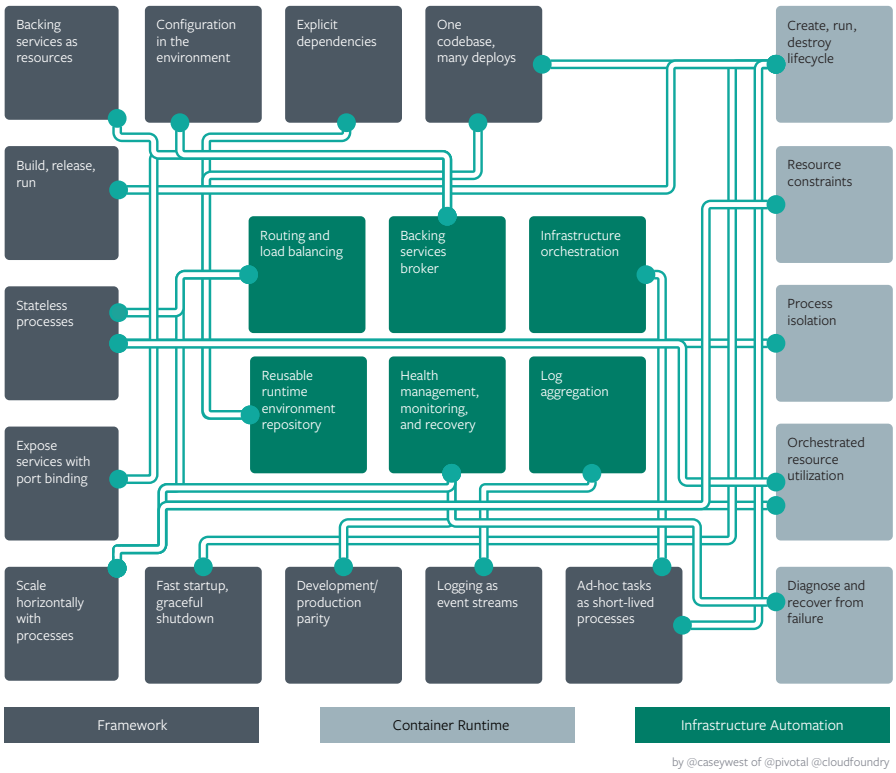
Getting and retaining the right staff is so key to success here that you should start your gardening your team right from the beginning.

GET A PLATFORM, BUT AVOID BUILDING YOUR OWN

The biggest greenfield project misstep and risk I generally see is succumbing to the urge to build your own non-standard approach and cloud platform. What I mean by "cloud platform" is the system that manages everything from the infrastructure up, defines how applications are packaged and deployed, defines how services (like databases, identity, queues, etc.) are provided, and provides capabilities to run, monitor, and otherwise operate in production. And don't forget that you may want the benefits of multi-cloud portability—to move your applications to whatever type of infrastructure you want—private, public, dedicated, "real" cloud, or just plain old virtualization.

Developers and operators love to build all this on their own. It's fun! But in the Cloud Native era, this is akin to writing your own web server, or worse, operating system. Instead, you'll want to save time by using a standardized cloud platform like Pivotal Cloud Foundry that's geared to solve all these problems and let you focus on business value.

The Cloud Native Architecture: *What you need to be cloud native*



by @caseywest of @pivotal @cloudfoundry

See *Casey West's "The cloud-native future"* for more discussion on the needs and components of a cloud platform. Also see *Brian Gracely's discussion of Cloud Native application platforms* for an industry wide view of cloud platforms.

Many of our customers went down the route of building their own platform, relying on automation technologies and containers to create them. This works at first, but cuts off access to the continual stream of innovations you'll get when using an open platform like Cloud Foundry. For example, once Netflix's microservices stack emerged as a solid framework for cloud architecture, Pivotal added it into Pivotal Cloud Foundry. As a negative example, one of our customers had built a cloud platform on their own that only supported Java, and their platform team could never seem to find the time to add other languages.

This last point is key as well. Are you in the business of building platforms or building the software that runs your business? You're going to need a highly automated cloud platform that supports the entire application life-cycle—development, QA, staging, production—and that task is likely best left to a third party (or ecosystem). Using a cloud platform will also force you to follow architectural Cloud Native discipline, like following the 12 factor app guidelines and breaking up your application into microservices. Remember, this stuff is proven to work, but if you deviate from the standard Cloud Native practices, your mileage may vary.

MAINSTREAMING THE GREENFIELD, INTERNAL MARKETING

Finally, as you start demonstrating success with your greenfield projects, consider how you might help the rest of your company. First, you should do some internal communications with lunch-n-learns or even internal hackathons and summits. Go over the stories (good and bad, corporate culture permitting) of projects, explain the processes you've been doing, and hopefully create a conversation with other groups who would like to improve how they deliver software. We've seen Pivotal customers like Allstate and Humana do this to start spreading company-wide improvement. And, indeed, an even larger step is to create dedicated "labs" teams, like Humana has done, to help spread the improvement.

Think back to the three goals above, and you will see why this internal marketing activity is valuable. It will help you build up support to keep going and also helps you spread the benefits of Cloud Native to the entire organization. It's downright responsible and leadership-y, if that's your thing.

WHAT'S NEXT?

After incorporating these ideas, you'll have hopefully succeeded at your greenfield projects and, equally important, have gotten your feet wet learning about the Cloud Native mode of operations or "lifestyle," as we might say in a school of thought that loves to bandy about the word "culture." The next task is to think larger, tackling larger applications which probably necessitate working with "legacy" code and services. This leads to an enterprise view of change, transforming the entire company into a software defined business.

PART 3

The Legacy Cloud Native Journey: Dealing With The Stuff That Makes All The Money

While the term "legacy software" usually has a negative meaning, the fact is that much of what's labeled legacy software is the core money maker for companies. It may be creaky, risky to change, expensive to run, and otherwise full of risk and fear, but, almost by definition, legacy IT is the software that's currently enabling the business and, thus, bringing in the money.

Here, companies get into a problem I like to call the shackles of success—they're trapped in the daily management of their legacy software and have no time to work on new, greenfield software. Worse, they have little time to transform how their organization creates software to support company wide improvements and plans to become Cloud Native enterprises. Depending on how well or poorly your software is designed, "maintenance" (fixing bugs and adding new features to existing software) may also be considered "legacy." If making updates to your software is easy and low-risk, you likely don't think of it as legacy. If making changes is difficult and error prone, you'll think of it as legacy. Whatever your situation—larger organizations have to deal with legacy code and services on the Cloud Native journey.

Let's look at some guidelines for how to work with, around, and possibly start to improve legacy IT as you go along your Cloud Native journey[2].

MANAGE THE PORTFOLIO

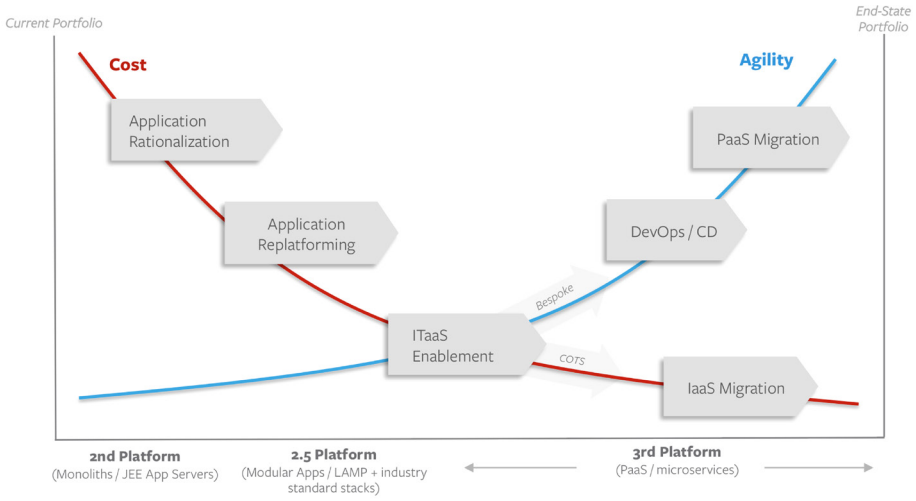
The first step in dealing with legacy software is to ensure that you have proper portfolio and resource management in place. What I mean is that you're aware of all the software you have, its approximate business value, and the expected life span of the software. With this knowledge in place, you can determine how many resources (time, money, IT assets, and corporate attention) to spend on each item. That is, you can determine the priority any given application has. Once you can prioritize assets, you can start making decisions about where and how you run that application (high-end infrastructure that requires a lot of manual, human attention vs. tape backup cold archiving vs. decommissioning).

On the Cloud Native journey, the goal of portfolio management is to free up resources (time and money) to focus on innovative, new software and capabilities. Remember the goal is to put a continuous delivery process in place that will allow you to start delivering software weekly, if not daily, which will allow your organization to start using software as the core enabler for company's business processes and strategy. To meet that goal, you'll need plenty of time to focus on "innovation." Sadly, most companies I talk with have very little time for innovation as they're held fast by the shackles of success—spending too much time managing their legacy software.

2 Josh Kruck contributed to the organizational changes part of this section, while Jared Gordon did the same in the selecting your legacy dance partners discussion

To illustrate what I'm talking about here, let's look at one approach to managing your portfolio.

Application Portfolio Modernization Value Levers



This approach from the EMC Global Services Application Transformation Discipline group is focused on identifying applications whose costs and management attributes can be optimized—basically, it shows the paths that different apps can take. It sorts IT services into six buckets according to their technical profile (e.g., can it run on virtualized infrastructure?). As well, business needs can drive the evolution of the service. For example, some applications are of little value and can be all but decommissioned (e.g., archiving the data for regulatory retrieval if needed), others can be moved from higher cost virtualization to public cloud IaaS, and others require the agility of a cloud platform approach. Part of the analysis assesses if there are merely process and tooling changes needed to improve an application in question—maybe doing the refactoring to get the application running on Pivotal Cloud Foundry is too high for the business value the application provides and simply automating the builds would add enough value and free up enough time. On the other hand, maybe the business value of ensuring very high agility is so high that the costs of "forklifting" the application to Pivotal Cloud Foundry are more than worth it.

What I like about this approach is that it forces you to re-evaluate the IT management and infrastructure needs of your applications. You can prioritize the applications and, thus, how you start spending your time. Not all applications are created equal, and not all of them need a white-glove level of service as they age and decrease in their value to the business. Most IT departments don't seem to operate this way, and, thus, find themselves over-spending on applications that no longer have commensurate IT value.

Honing Your Priority Management Methods

There are many other portfolio management methods, and I've just picked the above as an example. To quickly cite just two other methods—Cutter has an interesting approach that focuses on matching the appropriate methodology to each application. Also, the various interpretations of Gartner's bimodal IT thinking and pace layering can start to feel like a useful approach to sorting your portfolio and determining how to proceed with each asset. Finally, Pivotal's solutions group routinely works with customers to methodically identify and then move legacy applications to Pivotal Cloud Foundry.

Whichever way you choose, make sure you have some way of evaluating your software portfolio and then prioritizing how you deploy resources. To use a predictable quip—if you don't manage your portfolio, it will manage you!

SELECT YOUR LEGACY DANCE PARTNERS...CAREFULLY

Engaging with the beast of legacy can be dicey and risky. So, it's good to verify that you really need to dance. Clearly, if you're working with a core service that's needed for your application, you should work with it—likely using some of the architectural patterns referenced below. However, it can sometimes be wise to leave it alone. Let's look at some red flags for when you should leave legacy alone.

Negative Business Case and Financial Implications

Many enterprise projects are expected to live 2 to 5 years, if not longer. Accordingly, the up-front and ongoing budgeting of that project may depend on little to no additional spend after the first "big bang" release. Unfortunately, this model is antithetical to agile and Cloud Native thinking. Agile thinkers want to take advantage of the inherently plastic nature of software to make frequent changes at will for the benefit of the business. But still, toying with big bang budgeting in this manner may cause your business plan to go into the red. For example, if you incur extra, unanticipated costs to modify legacy software and bring it up to contemporary fashions, or you outright cancel a project before it's been able to "pay back" the investment, you may encounter corporate flack, or to echo the schoolyard days, just plain "get in trouble." Most IT-minded people have an engineering mindset that causes us to pursue the "right" solution to a problem, which is usually good. However, if you sense that you'll have to, as they say, "revisit the business case" for a legacy system you're looking to work with, make sure you can avoid negative consequences. This usually amounts to talking with key stakeholders and the finance department as soon as possible to explain your case.

Agile Resistant Projects

While dated, the 2003 book *Balancing Agility and Discipline* has several good discussions of when to choose waterfall instead of agile approaches. As the application development and infrastructure layers have become more automated and cheaper (that is, cloud!)—many of the concerns in the book have lessened in relevance. However, they offer some cautionary advice about projects that may be best not to monkey with[3]:

1. Changes to the software require close integration across organizational boundaries—if you must work with several different groups to make "simple" changes, you could find yourself in a quagmire. Many of the tenets of microservices address this problem by carefully embracing it, as we've discussed numerous other places.

2. Companywide and/or industry standards can not easily be changed or ignored—if there are numerous self or externallyimposed regulations and standards that must be followed, you may not have the option to change them at all. Battling against these exogenous requirements can become too time consuming. That said, it's common for companies to assume that implementation requirements for audit and compliance will hamper Cloud Native approaches. However, it's often actually the case that they just need to come up with a new implementation to satisfy the original requirements, perhaps even better than the old ones. It's good to investigate what's actually needed to be compliant rather just assume that the situation is impossible[4].

3. The legacy system is not modernized...at all. If the legacy software is essentially a black box and is in no way modernized, trying to change it too much may result in slowdowns and friction. Approaches like the strangler pattern below might help here, or you might be best suited to just quarantine the legacy service and wrap a good, modern API around it. Once you have good portfolio management in place, you will have identified applications that can be taken off the table with respect to software development needs. However, you'll be left with another bucket of applications that are not so easily gotten rid of, like those relatives who don't seem to promptly leave once the holidays are over. These are applications that you need to still evolve and develop, and you may even need to move them to a cloud platform like Pivotal Cloud Foundry. As we'll go over here, there's actually a lot of options.

3 This is my paring down and slight updating of a list of five attributes for projects that would fit poorly with an Agile approach, on page 30. The two items I left off—monolithic requirements and continuity requirements—have more to do with the creation of the software than with integration of other services.

4 For a good example of this, see the discussion about reducing delays in the US Federal software delivery cycles from 9 months to just days.

The Leftover Legacy That Won't Leave

Once you have good portfolio management in place, you will have identified applications that can be taken off the table with respect to software development needs. However, you'll be left with another bucket of applications that are not so easily gotten rid of, like those relatives who don't seem to promptly leave once the holidays are over. These are applications that you need to still evolve and develop, and you may even need to move them to a cloud platform like Pivotal Cloud Foundry. As we'll go over here, there's actually a lot of options.

Some of these applications might require net-new rewrites, in which case I'd think of them more as greenfield, truly "Cloud Native" natives where you have the benefit of a working "prototype" (the existing, legacy application!) to base your new application on. In many cases, though, you won't have the benefit of being able to start from scratch. Even more commonly than being faced with the choice to rewrite a legacy application, you'll have many legacy services that your new applications need to integrate with and rely on, as one of the early Pivotal Cloud Foundry customers, CoreLogic explained in this year's CF Summit talk. The rest of this piece will go over some tips for dealing with those applications and services.

TESTING, WHY'D IT HAVE TO BE TESTING?

One of the more popular definitions of legacy code comes from Michael Feathers' classic in the field, *Working Effectively With Legacy Code:* "legacy code is simply code without tests." Most code will need to be changed regularly, and when you change code, you need to run tests—to verify not only that the code works, but that your new code didn't negatively affect existing behavior. If you have good test coverage and a good continuous integration and delivery processes in place, changing code is not that big of a deal and you probably won't think of your code as legacy. Without adequate, automated testing, however, things are going to go poorly.

Thus, one of the first steps with legacy code is to come up with a testing strategy. The challenge, as Feathers points out, is going to be testing your code without having to change your code (too much). After all, to quote from Feathers again:

> *The Legacy Code Dilemma*
> *"When we change code, we should have tests in place. To put tests in place, we often have to change code."*

Feathers' book is 456 pages of strategies for dealing with this paradox that I won't summarize here. What I want to emphasize is that, until you have sufficient test coverage, you're going to be hampered. In other words, this is one of those pesky prerequisites for being a successful Cloud Native enterprise.

AUTOMATING LEGACY, HIGH VALUE ONLY

One thing that may be possible is automating as much of your build process as possible. As numerous industry studies show, the use of continuous integration is not wide-spread. Even if you can't wrap tests around everything, try to automate your build by adding in as many "smoke tests" as possible throughout. This will give you the scaffolding to put test harnesses in place, but also start to follow some 12 factor principles like separating configuration and putting everything (beyond just code!) into version control.

When it comes to picking what to automate and test, *Continuous Delivery*'s Jez Humble & David Farley suggest picking the highest value parts of your system. Ideally, you can automate building all of the system and spend extra time testing these more high value parts of the system. Prioritizing and focusing on deeper testing like this is a short-cut from doing everything. However, it is more pragmatic and at least gets you a "skeleton to protect legacy functions" within your old code.

WHEN YOU HAVE TO CHANGE CODE

If you are lucky enough to have the option to change code (or cursed, depending on your situation), there are many options and tools available to move your legacy applications into a Cloud Native platform like Pivotal Cloud Foundry. Thankfully, we have been documenting these concerns a lot recently with more to come so I'll just summarize here.

For Architecture, Favor The Strangler Pattern

The "strangler pattern" (named for vines that slowly take over a tree, not some brutish EOL'ing technique) provides guidance on how to slowly modernize a legacy code base. You carefully select the well separated sub-components of the system and create new services around them, instituting the policy that new code must only use these new service interfaces. Over time, the effect is that fewer and fewer of the legacy services are used directly until one day, you can swap over to just new code (having replaced the legacy implementation behind the new service interfaces) and convert them over to the new approach. In other words, converting them to a microservice may be slow and deliberate,

but that steady pace generally makes it a safe approach. This pattern is covered numerous places, including Matt Stines' book on migrating to Cloud Native applications, which contains several other approaches for modernizing legacy architectures.

One variant of this looks to convert your ESB and SOAP driven SOA services over to microservices, slowly but surely. In this instance, instead of looking at Pivotal Cloud Foundry as an application enabler, you can also look at it as a services "hub" (to use an SOA friendly term that will make Cloud Natives cringe). Indeed, if you stare at the architecture of Pivotal Cloud Foundry itself, you'll realize that the whole thing is actually a service oriented architecture itself. Recursion is fun!

Some Things Are Happy To Move: Java, Static Content, Simple Apps

Thus far, moving legacy applications to Pivotal Cloud Foundry can seem like a chore. In fact, there are many applications that can be easily fork-lifted to Pivotal Cloud Foundry without much suffering. Java applications that use standard libraries and don't have dependencies on local file systems or vendor specific application server libraries can be easily moved, as Josh Long has covered recently. Unspooling Java applications from proprietary application servers can also be done with a good, systematic approach.

When it comes to static content, Pivotal Cloud Foundry can provide a surprisingly easy and well governed approach to basic content management as well. I've spoken with several large organizations who are hobbling along with modernizing ancient CMS systems used by their content groups to quickly update their web and mobile applications. In these instances, looking at that static content as an application that is cf push'ed into Pivotal Cloud Foundry—benefiting from all the release management and cloud management capabilities—can be one path for tackling your legacy applications.

For more ideas on how to modernize legacy code, check out Josh Kruck's post on the topic and his paper with Abby Kearns on the topic (which I'm told will have a major updating soon). Jared Gordon also has a series on about working with legacy code which collects lessons we've learned as we've been helping our customers on their Cloud Native journey.

LEGACY PROCESS AND GOVERNANCE CHANGES

Because Conway's Law has such large sway over software and the structure of the organization that supports that software, it's hard to tinker with the organization much unless you tinker with the software as well (and vice-versa)!

Legacy processes often involve much review as the software passes through phase-gates. Metaphorically, and often literally, this is thought of as "going to the change review board," a governing body that approves advancement to the next stage. It can be metaphoric—as there may not actually be a sitting board that approves things but a latent "council of elders" who approves all changes in IT. For older, slow moving IT that's laden with risk and the chance of failure, this may have seemed like a good idea at the time. It does, however, slow down the small batches and effectively quash the more frequent delivery mentality that a Cloud Native approach favors. Indeed, it can cause the dreaded "waterscrumfall."

Heavy oversight like this may be inescapable, mostly due to regulatory concerns. In such cases, we've been finding that working under the rules of the change review board can lead to some positive effects, namely:

- **The one about babies and bathwater.** Working with the review boards may allow you to identify the valuable parts of the legacy process. These are parts you'll want to retain! Hopefully, as you look towards transforming your organization, you can work towards automating these beneficial processes. For example, a result of the change review board may be updating the PPM tools that allow your organization to properly do portfolio management, as discussed above. Perhaps the process can be automated as part of your continuous deployment process instead of done manually.

- **With trust, comes much permission to change.** For a rational organization (hopefully you're in one!), if you can show the change review board that a series of quick, small changes actually reduces risk and increases quality, they may start to trust the new way, accepting quick change as the new routine. As you transform more broadly, small wins you've had with greenfield and legacy systems will have built up organizational trust in your new approach. This may allow you to lighten the heavy hand of the change review board over time.

- **The process fifth columnist.** Sometimes, the best way to change an onerous process is to fully engage with it, making sure to demonstrate why it's onerous. If the change review board becomes an obvious bottleneck, it may be helpful to demonstrate this as a way to start transforming it. It also demonstrates how much the legacy process gets in the way of rapid improvement and becomes a barrier to scaling value delivery. Remember, the business probably wants smaller, more agile, more regular batches of change rather than a great big long one.

Navigating these changes can seem like extra work and be really annoying to deal with. However, if you're taking the bigger view—that your organization's process is just as important as the actual product, you'll realize that you won't get all the benefits of a Cloud Native approach if you don't change most, if not all of, of the organization creating the production.

USING YOUR SUFFERING FOR A BETTER TOMORROW

Dealing with legacy software is a hassle and often painful. As any student of legacy code will tell you, the paradoxical thing about legacy software is that because it's been so valuable to the business over the years, developers have added more and more features to it. This adds complexity and, often because tests are not updated and staff have churned, architecture is unknown, and the code-base altogether poorly understood, the code becomes a risky mess to touch. If you're experiencing that, use that pain as a motivation

to start doing things in a more healthy, future-proofing fashion—insist on writing all those tests, really automate all stages of building and deploying the application, and otherwise be disciplined.

The benefits of legacy code (it's what runs the business!) and the suffering that code causes (we spend all our time just keeping this thing up because it's so fragile!) are good reminders for why you should always be improving your process. Constant submission to short-term optimization yields to long term pain and business stagnation—bang head here. The next time something feels weird or painful on the Cloud Native journey, use that line of thought to remind your team that it could be worse, and probably is.

PART 4

Enterprise Transformation

This final section discusses what an IT department, that has transformed into a Cloud Native enterprise, looks like and how it got there. It also fills in some of the missing items from the greenfield and legacy discussions above.

BEYOND TACTICS: TRANSFORMING TO A MORE RESILIENT CULTURE

The "transformation" Cloud Native journey is largely about how you establish the new behavior and habits of a software defined business. While many of the tactics and technologies I covered in the greenfield and legacy parts carry over here, the transformation journey is focused on building a new operating model for IT. Of course, this means changes to responsibilities, processes, and technology. Yet, there is an even more critical foundation—creating and cultivating a beneficial culture. These can seem like "soft" concerns, but just as zombie movies are never really about zombies and instead about human culture, new technologies are rarely about the technology themselves but new ways of using technology. Of course, zombies, and technologies, are more than just MacGuffins and play a major role themselves, but focusing on the human, behavioral, and cultural element is the transformative part.

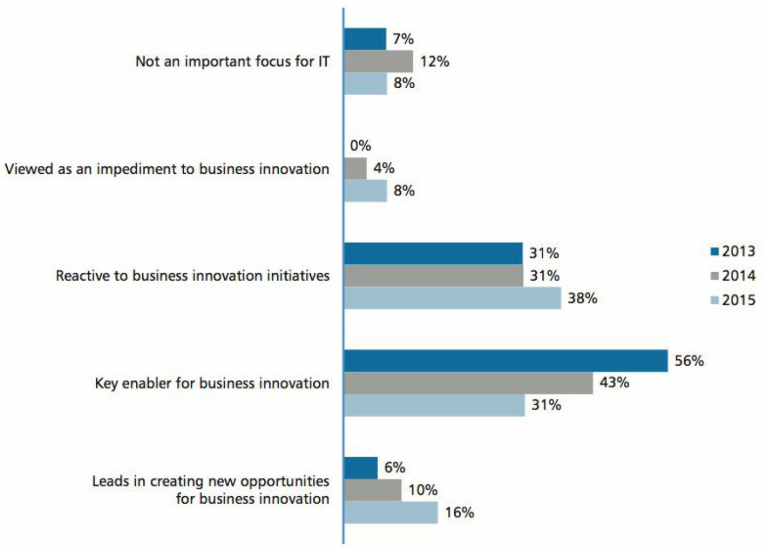
In our case, the MacGuffin, as covered in the introduction to this series, is the ability to reliably and safely deploy software weekly, if not multiple times a day. Importantly, this addresses a larger goal—you get the ability to start investing more into digital customer experiences or new, digital business models by more perfectly designing and crafting software. The cultural change starts with these goal in mind—leaping at the opportunity to improve your software and more closely aligning your applications with business strategy. To make this change sustainable, you'll need to change how the IT department is organized and run. The new roles, responsibilities, practices and processes you'll put in place, intermixed with new group behaviors, will become your new "culture."

This is hardest, most tedious part of the Cloud Native journey. So let's jolt ourselves out of any laurel-bedded reposes we're enjoying by revisit the motivations for going through the suffering of change by looking at an assessment of how "IT culture" is currently doing with respect to proving value to their organizations.

WHAT YOU'RE DOING ISN'T WORKING

Companies definitely feel the need to improve the role IT plays in the overall business, as multiple studies have been showing. One of the more illustrative studies comes from the Cutter Consortium and compares the value of IT to business innovation over the past three years:

What is your IT Organization's role in business innovation?



Source. Cutter Benchmark Review, May 2015, n="80 organizations."

The drop in IT being a "key enabler for business innovation," from 54% in 2013 to 31% in 2015, is shocking. It points to a real need—for leaders in IT to focus on transforming their organization into a strategic business partner.

Thankfully, there are proven ways to address this need. Let's take a look at some of the ways IT starts transforming and hopefully reverses this downward trend in usefulness.

YOUR MEAT IS ROTTING: MEATWARE IS THE PROBLEM

As I talk with companies who are looking to transform how they do IT, I realize more and more that the problem is not so much technology, or even how that technology is used. The problem is their organization's unhelpful processes, behavior, and culture. You can think of these as "thought technologies," but I like to call them "meatware." Much of what it takes—to transform IT organizations into centers for innovation—is about changing the IT culture from a command and control, big planning up-front, organized by function (or "silo'd") mind-set. Most IT departments operate in this traditional way, creating teams aligned by function and capability then assigning them to projects as needed. When the problems of IT are well understood, and the IT team's job is to keep all the processes humming and perform routine maintenance, this kind of approach can be good. However, when your

task is exploring new processes and software products—as a Cloud Native enterprise, reinventing and evolving how its business operates—this traditional, keep the trains running on time mindset is harmful.

There are many ways to look at these two modes of operating: bimodal IT, the three horizons, Agile vs. Six Sigma, etc. Out of all the great strategy framings, I like the explore/exploit model. When businesses start off, they are exploring new innovations in products and services as well as how they create and deliver those wares to customers. This process requires experimenting, learning, and risk taking until the team finally finds the right mix of customer need, product, and go-to-market model, ultimately achieving competitive advantage and exploiting a growing, profitable, business model.

Once a company finds that model, they usually switch to the exploit mode of operating. This mode knows exactly what the business is, understands the customer need, has a product or service that meets that need, and understands how to effectively get that offering into a customer's hands in exchange for money. With organizations like governments and nonprofits, the end goal is satisfying "the mission," fundraising, or providing a service to citizens. In the exploit mode, companies look to optimize how their processes run and "execute to plan" because the plan is working!

Too often, IT departments who need to be operating in an explore mind-set have a culture of operating in an exploit mindset—they fail to change their culture over to the needed explore mode. It's little wonder then that Gartner is predicting 90% failure rate in DevOps adoption if organizations don't address cultural issues. The first step, then, is to become cognizant of which mode you should be operating in. This knowledge—like all good strategy—needs to be spread "down" the ranks to individuals and become a key enabler of transforming your IT department. Here is how one Pivotal customer explains the transformation:

> CoreLogic brought software engineers to Pivotal Labs to learn Pivotal's extreme agile software development methodology. The methodology focuses on test-driven development, pair programming, short development cycles, and continuous verification and integration of code to improve software quality and flexibility and reduce cost. As a result, the culture of CoreLogic product management and technology groups has fundamentally changed to become more nimble and collaborative.

"For us, it's been much more than a technological transformation moving to a (Cloud Native platform). It's a new way to develop products.It's the most exciting thing we've done in the last 12 months."

—Richard Leurig, Senior Vice President, Innovation Development Center, CoreLogic

STRATEGY & MOTIVATION

Instead of legacy meatware, Cloud Native enterprises are typically comprised of self-motivated and directed teams. This reduces the amount of time it takes to make decisions, deploy code, and see if the results helped "move the needle." More than just focusing on speed of decision making and execution, building up these *intrinsically* motivated teams helps spark creativity, fueling innovative thinking. Instead of being motivated by metrics like "number of tickets closed" or "number of stories/points implemented," these teams are motivated by ideas like "increased customer satisfaction with faster forms processing" or "helped people track and improve their health by moving the application to where the people are when they're being healthy."

In my experience, one of the first steps in shifting how your people think—from the Pavlovian KPI bell—is to clearly explain your strategy and corporate principles. Having worked in strategy roles in the past, I've learned first hand how poorly articulated goals, constraints, and strategy can lead to equally poor results. While there are many beautiful (and not so beautiful!) presentations that extoil a company's top motivations and finely worded strategies, most employees are left wondering how they can help day-to-day.

Whether you're a leader or an "individual contributor," you need to know the actionable details of the overall strategy and goals. Knowing your company's strategy, and how it will implement that strategy, is not only necessary for breeding self-motivated and self-managed people, but it also comes in handy when you're looking to apply agile and lean principles to your overall continuous delivery pipeline. Tactically, this means taking the time to create detailed "maps" of how your company executes its strategy. For example, you might do value-stream mapping, alignment maps, or something like value chain mapping. This is a case where I, again, find that companies have much less than they think they do—often, organizations have piles of diagrams and documents, but, very rarely have something that illustrates everything—from having an idea for a feature to getting it in customer's hands. A Cloud Native enterprise will always seek to be learning from and improving that end-to-end process. o, it's required to map your entire business process out and have everyone understand it. Then, we all know how we deliver value.

The process of value-stream mapping, for example, can be valuable for simply finding waste (so much so that the authors of *Learning to See* cite only focusing on waste removal as a lean anti-pattern). As one anecdote goes—after working for several weeks to finally get everything up on the big whiteboard, one of the executives looked up at all the steps and time wasted in their process to get software out the door and said "there's a whole lot of stupid up there." The goal of these exercises is not only removing "stupid," but also to focus on improving the processes.

MANAGEMENT CREATES THE GAME...SO, HOPEFULLY THEY SHOW UP

Coupled with discovery and experimentation of strategy and approach, there is a necessary shift in the role of management. In an exploit mode of operating, management is often responsible for making decisions—approving changes, putting top-down plans in place,

and endlessly reviewing "status." In an explore mode, management needs to shift to being more hands on, more like a personal trainer or coach that encourages exploration before quick decisions and immediate action.

Having grown up like many computer nerds, I played role playing games like Dungeons and Dragons extensively in youth. In that game, there are "players" who go on the adventure and a "dungeon master" who orchestrates the adventure, including narrating the adventure, controlling the monsters, doling out rewards, and enforcing (or not) the rules of the game. Without players to star in the game and a dungeon master to run the game, there is no game. Similarly, the role of management in a Cloud Native enterprise is to create and orchestrate the "game" that the organization is playing. This is much different than sitting through endless meetings reviewing statuses and rewarding or punishing those responsible.

First, management needs to "create the game" by establishing the strategy, goals, and general operating constraints (you could say "principles," but I prefer the Meat Loaf approach of specifying the small set of things you won't do, assuming that anything else is possible). As put in *Leading the Transformation*:

> Management needs to establish strategic objectives that make sense and that can be used to drive plans and track progress at the enterprise level. These should include key deliverables for the business and process changes for improving the effectiveness of the organization.

In addition to this game setting, the hands-on approach comes down to "roaming the halls" and participating in the process—especially at first—as teams are learning to become self-reliant, self-motivated, and self-managed. Here, leaders can lead by example and start to move out from behind the bustles of PowerPoint.[5]

Speaking of being more hands on, let's look at a buffet of management tactics to start considering and deploying—starting with how to manage your IT portfolio.

CRAFTING THE ORGANIZATION: PRODUCTS RUNNING ON A PLATFORM

The traditional way we think about structuring the IT department is different than how Cloud Native IT departments structure themselves. To massively simplify it, traditional IT departments are oriented around working on projects, whereas technology companies are oriented around working on products.

Traditional thinking tends to focus on service delivery, responding to requests as they come in. Granted, the request may be a large project, but the idea is to have the requester more or less know what they want. The IT department delivers on that and then keeps the service up and running. This is great for the exploit mode of thinking where requesters

5 For more on how management's role and tactics change in during this transformation, check out Siobhan McFeeney's recent blog post on the topic.

know what they want and don't change requirements around too much. However, there are drawbacks to the exploit mind-set. First, it can lead to over-thinking each problem, often called "analysis paralysis" or "solutionizing." In essence, you can end up doing both too much planning and then over-service (doing too much!) to solve an otherwise simple problem. The bigger problem with this approach is that it's not always appropriate for the type of problem being solved. When the nature of problem is largely unknown, a project mind-set often results in poor solutions.

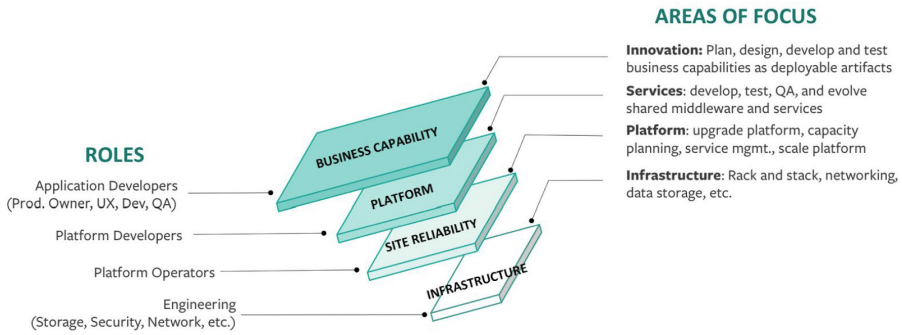
An *innovative, product-centric* approach is what's needed in the explore mode—where people don't know what they want and are often (at first) completely wrong about what they want. Teams in this setting should be more closely attached to the software being written rather than parachuted in as needed. The team's understanding of the problem being solved, approaches tried in the past, and the overall "tribal knowledge" will be invaluable in figuring out the right product to build. These integrated teams are not only important for product management continuity but also for ensuring that the product is resilient in production. As outlined back in the greenfield section above, these teams needs to be kept small—somewhere between 6 and 12 people—and you want them to have all the skills needed for the full life-cycle of the product, including development, testing, design, and operations.

When it comes to operations, this is where DevOps plays an enabling role in Cloud Native enterprises. If the team that wrote the application code is also responsible for keeping it up in production, that team will make sure they write resilient code. The team will seek to add operations staff directly to the team rather than leaving that as "someone else's problem." Setting up an organization like this requires, not only developers and operators comingled on teams, but creating an organization that supports the cloud platform. This introduces two new roles in IT, the platform operators and the platform developers.

Platform operators are responsible for keeping the cloud platform up and running along with updating the software and infrastructure. Early on, they may be responsible for consulting with the application teams as the organization learns Cloud Native development and operations practices.

Platform developers work on the evolution of the cloud platform itself. They focus on adding in new services like databases, integrations with partners and third parties, and otherwise adding in industry and business specific "business logic" to the cloud platform. These platform developers are creating a "product," one that's targeted at the company's developer teams.

The Emerging Cloud Native Organization Model



Source. based on slide from Pivotal Cloud Foundry Solution team, discussions with GSA's Diego Lapiduz

As the illustration above shows, the amount of staff in each of these layers reduces as you go "down" the stack. This is because you're focused on applications as the primary point of customer value, and the most staff are there, exploring and perfecting your software. Next down, especially at first, the platform developers will be a smaller pool but well staffed. The platform operators, in practice, tend to be a shockingly small number—from single digits in smaller organizations to 10's or 20's in very large organizations. Finally, if you're running all of this on your own— as many Pivotal customers do— you'll need staff to take care of the raw infrastructure, from the hardware all the way down to the dirt under your datacenters.

This is all based on the early days of the Cloud Native approach and may evolve, but, so far, this is what has been panning out in organizations that I've observed.

MELT THE SPECIAL SNOWFLAKES: PORTFOLIO CONSOLIDATION

To transform, you'll likely need to start consolidating your applications down as much as possible. If you're like many of the large IT shops I see, you've accumulated 1,000's of applications, many of which are one-offs and "special snowflakes." In addition to heterogeneity, the second problem with these applications is that they've generated a lot of technical debt—compromises made to get them out the door and do them cheaply, including a lack of automated test coverage. You're sitting in a house of well reinforced paper mache, essentially. To address this, most companies begin to systematically put in a program to reduce this debt. This takes many years to do, and you should be mindfully managing and measuring it to ensure that this debt doesn't consume all of your resources. This is the continued portfolio management I discussed in the legacy section above, with the addition of looking to consolidate some applications.

In addition to finding duplication in the portfolio and combining applications as it makes sense, the Cloud Native approach seeks to move as many applications as possible to a shared cloud platform, like Pivotal Cloud Foundry. This will have tremendous resource

scaling effects based on reducing variability in your portfolio, which will lower many costs while providing additional value:

1. **More automation and standardized management.** Operations manages one platform and uses similar, if not the same, tools to monitor all applications in production. Companies have long benefited from standardizing on operating systems or even building standard virtual machine images. This has still left plenty of variability for how applications were packaged, provisioned, updated, and managed in production. It's only recently, with the advent of cloud platforms, that this next layer—the application layer—has been helpfully standardized with respect to automation and management in production.

2. **Standardized application lifecycle management (ALM).** While development teams are free to choose languages, frameworks, and even the developer tools they use, each development team is working within the same "contract" specified by the cloud platform. This gives developers the freedom to choose how they code their applications, but limits the middleware and systems they can choose, reducing the overall effort to manage your portfolio of applications.

3. **Compliance and regulation speedup.** By standardizing on the same underlying platform, you can greatly reduce the amount of "paperwork" needed for governance, risk, and compliance (GRC), as well as audits. As an example, the GSA has been able to reduce the length of time required to pass strict government audits for software releases down from 9 months to just days by standardizing on a shared cloud platform.

Always be on the look-out for consolidating services and applications. At scale, this is how you'll survive. The trick is make sure you're not giving up flexibility and speed in favor of controls. In my experiences, the practice of traditional, ITIL-driven IT service management falls too far over the line of control. So, that's a habit to be wary of. Of course, the right cloud platform will balance standardization, automation, and developer flexibility.

FURTHER READING & READING FROM YOU

At the outset of this paper, I focused on greenfield and legacy journeys because there's already a tremendous amount of work on transforming IT organizations. I find much of the "transformation" work out there thrilling and equally breathless—it all sounds thrilling, but it leaves you wondering exactly what to do. There are some stand-out sources that cover actual tactics for transforming your organization in much more depth than I do here.

For example, there are two books I recommend strongly if you're interested in transforming how your organization operates. First, *Lean Enterprise* is a concise but comprehensive guide to all of the recent thinking about how to better run "normal" IT departments, much along the software defined business lines I've been discussing in this

paper. Second, as I've referenced several times here, *Leading the Transformation* is a good collection of tactics for managers to deploy—both up the chain to convince corporate overlords that transforming how IT is done is a good idea as well as down the chain to help staff in the IT department become a continuous learning organization.

On the topic of your people—staffing and training—I can't recommend Andrew Shafer's talk "there is no talent shortage" enough. To reduce it down to a related, favorite anecdote:

> Reaction from F100 CTO:
> *"But Netflix has a superstar dev team, we don't!"*
>
> @adrianco's (formerly from Netflix) response:
> *"We hired them from you."*

Finally, I'm curious to hear what you, dear reader, are finding out there. Simply knowing the problems companies are encountering as they're trying to become Cloud Native enterprises is helpful to everyone. Hearing about what has worked and hasn't worked is helpful as well. To be overly recursive, that's the broader mind-set we, in the industry and who want to improve the state of software, need to take—a continuous learning approach to the craft of the software life-cycle itself.

Part of that is sharing our experiences with each other and I'd love to hear yours: feel free to email me at cote@pivotal.io.

WHAT'S NEXT? GETTING STARTED WITH PIVOTAL

As you might be thinking, most Pivotal customers are in all three buckets. My recommendation is to start with a greenfield project to just figure it all out. Because Pivotal Cloud Foundry is truly multi-cloud, you can start in our public instance Pivotal Web Services and move to your own dedicated hosting or even on-premises if you want. That is, there's no technical reason (like having to build your own cloud) that would hold you back from starting and deploying on day one. Again, you can start with our on-demand platform today, deploy to your own single-tenant version in the public cloud tomorrow, and decide later when to set up and push it to an internal cloud platform. And when it comes to the actual Cloud Native application development, we also have something to offer in the form of Pivotal Labs where we've been working directly with companies for many years to help transform how they do software development, literally pair programming and growing skills together, not just outsourcing coding over the wall.

A three part presentation series of the above is available as well: part 1, part 2, and part 3.

ABOUT MICHAEL COTÉ

Michael Coté works at Pivotal in technical marketing. He's been an industry analyst at 451 Research and RedMonk, worked in corporate strategy and M&A at Dell in software and cloud, and was a programmer for a decade before all that. He blogs and podcasts at Cote.io and is @cote in Twitter.

ABOUT PIVOTAL

Pivotal's Cloud Native platform drives software innovation for many of the world's most admired brands. With millions of developers in communities around the world, Pivotal technology touches billions of users every day. After shaping the software development culture of Silicon Valley's most valuable companies for over a decade, today Pivotal leads a global technology movement transforming how the world builds software. More at **pivotal.io**

Pivotal's Cloud Native platform drives software innovation for many of the world's most admired brands. With millions of developers in communities around the world, Pivotal technology touches billions of users every day. After shaping the software development culture of Silicon Valley's most valuable companies for over a decade, today Pivotal leads a global technology movement transforming how the world builds software.

Pivotal 3495 Deer Creek Road Palo Alto, CA 94304 **pivotal.io**