WHITE PAPER

From Months to Hours: Accelerating Software Deployment with Pivotal Cloud Foundry

By Brian Jimerson and Jared Ruckle

Pivotal

Table of Contents

Introduction

Large enterprises are in an unchartered era of innovation and competition. Startups can use software to challenge large incumbents, from anywhere around the world, with little investment. Big companies now realize that innovation must be their engine of growth. Organizations can use software to develop and deliver new ideas quickly, and rapidly create new value streams for customers.

Today's delivery channels are simply Internet-connected devices. The real opportunity lies in the software that runs on hardware. For example, people and businesses communicate and transact via phones and tablets that run connected applications. Many goods and services, from books to music to even financial products, were once purchased at traditional brick and mortar locations. Now, they are delivered electronically. Even automobiles become autonomously driven after receiving a software update over the air.

All of this brought on a common realization: companies must adapt their business model or become obsolete. The new imperative: build custom, highly differentiated software that's delivered early and often to customers.

A change in product strategy is accompanied by a shift in culture. Large organizations have tended to be risk-averse, analyzing markets and opportunities through well-established techniques, for long periods of time. These days, leading firms now embrace a culture of experimentation. This is the best way to determine what customers want and need.

Software-defined businesses must be able to deploy software early and often. Deployments must be consistent, automated, and low risk. If traditional deployment practices are left in place, time-to-market slows. An archaic culture hinders a company's transformation efforts.

How should business leaders shift their policies, practices, and mindsets? How can they build modern applications on cloud-native platforms to achieve rapid "time to value" for new code?

This paper aims to answer these questions. We'll share the best practices and lessons learned by Pivotal through our work with hundreds of large enterprises. It is meant to be an overview of how modern applications and platforms are deployed in today's large enterprises, rather than an all encompassing guide. The majority of the paper addresses deployment considerations for application developers. However, applications only perform as well as their underlying platforms! As such, we close the paper with guidance and best practices for operators on to how to rapidly deploy patches, upgrades, and upgrades to Pivotal Cloud Foundry.

Application Deployment

The Status Quo: Risky Business

Enterprise software deployments today involve a complex web of interactions between many departments. Each task may take very little time, but the lead time between each step is often significant. Release schedules are further complicated by change review boards and the scheduling of outage windows. The end result: a release cycle of 3 to 6 months.

Now, let's take a step back. How did this happen? Why does it take so long to deliver new code?

All organizations strive to avoid risk as much as possible. Risk means uncertainty, and uncertainty can hurt an established, profitable line of business. With traditional software development practices, risk has historically been introduced by change. Consequently, change is avoided to reduce risk.

Over time, risk mitigation becomes the responsibility of a dedicated team. Their goal: avoid risk in each release, and in future releases. This reactive approach to risk avoidance built up over the last several decades. Here is the culprit, the root cause of the siloed processes that we see today.

Let's look at a simplified, representative software release process in an enterprise. Consider a development team that just finished up important new features to meet a business goal. The team submits a request for the code to be part of the release cycle. This requires the following:

1. The virtualization team needs to create new virtual machines to run the software.
2. The operating system team needs to patch and configure the virtual machines to bring them up to corporate standards.
3. The middleware team needs to install and configure the software runtime and associated libraries.
4. The database team needs to create a database and associated schema.
5. The networking team needs to create firewall rules to allow traffic to and from the software.
6. The DNS team needs to create DNS records for the software.
7. The load balancing team needs to add and configure load balancer configuration for the software.
8. The release engineering team needs to merge the changed source code into a release branch (or stream) in version control.
9. The change review board needs to review and approve the software.
10. The software release needs to be scheduled for the next release window.
11. All of the participating teams need to develop and approve a rollback plan if one of the steps fails.
12. The software release is coordinated among all of the participating teams during the release window, which is usually a weekend night.

Take another look at these steps. It's obvious why releasing software today requires such a large window. Even if each team has automated their tasks in this process, the lead time between each step prolongs the timeline. What is needed is a new process for software delivery, one that ensures consistency, reliability, and efficiency.

In short, enterprises need a new process that drastically reduces the delivery timeline.

Developer Productivity

It bears repeating: enterprise development teams are responsible for delivering value to customers and partners. Any time spent on tasks that don't directly enable these business goals is wasted. Unfortunately, much of a developer's time is spent on non-valuable tasks relating to infrastructure, boilerplate code, and ticket management. Today's deployment models are an impediment to most developers.

Use a Cloud-Native Platform to Empower Developers

Pivotal aims to drastically increase developer productivity by automating many tasks related to building, deploying, and operating software. Pivotal Cloud Foundry (PCF) enables high developer productivity through several avenues:

- It's a self-service platform that "just works" for building and running apps.
- Deep compatibility with enterprise-grade development tools and frameworks for boilerplate code and industry-standard design patterns.
- An opinionated approach to software lifecycle methods and frameworks that remove tedium.

Pivotal Cloud Foundry is a catalyst for organizations that embrace "cloud-native" thinking. This movement incorporates:

- A set of new design patterns for enabling high-quality, resilient, and scalable software (12-factor apps, microservices).
- Enterprise-grade development frameworks for development (Spring, Steeltoe, .NET Core).
- Continuous integration and delivery tooling (CI server, artifact repository, source control, automated testing and acceptance).
- Organizational structures and culture that eliminates siloed deployment models (examples include DevOps and Lean practices).
- Reliable, repeatable delivery mechanisms for software, through industry best practices and automation.

So how exactly do cloud-native teams deploy code? Glad you asked!

Application Deployment Model

The following diagram shows the recommended deployment process for development teams on Pivotal Cloud Foundry:

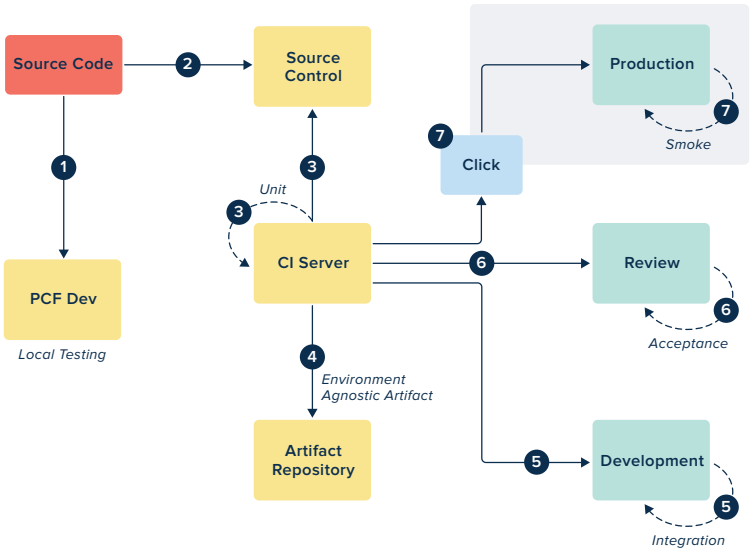

Figure 1: A cloud-native deployment process with Pivotal Cloud Foundry.

Let's examine what's happening in Figure 1:

1. A developer makes a committable change to the source code, and tests the change locally on PCF Dev to ensure it runs properly.
2. The developer commits the change to source control (Git or TFS).
3. The CI server detects the change to source control, checks out the latest copy, and runs unit tests on the changes.
4. If the unit tests pass, then the software version is built and committed to an artifact repository.
5. The built artifact is deployed to a development space. Integration tests are run automatically.
6. If integration tests pass, the built artifact is deployed to a review space, and acceptance tests are run automatically.
7. If acceptance tests pass, a release engineer or product owner can choose to deploy to production via an automated deployment. Smoke tests are typically run against the production application prior to cutting over URLs.

This is a very different process from traditional deployments. The differences worth highlighting:

1. A single built artifact is deployed to all environments, without rebuilding and repackaging. This may be a JAR or WAR file (for Java applications), or a DLL or web site directory (for .NET). Regardless of the unit of deployment, the same version of source code is **never** rebuilt or repackaged **after** it passes unit tests. The same versioned artifact is used for all environments.

2. Since a versioned artifact is never rebuilt, configuration that may change between environments isn't packaged in the artifact. Instead, configuration is provided by the platform in some fashion. For Pivotal Cloud Foundry, configuration is provided by environment variables, or an external configuration server such as Spring Cloud Config Server.
3. The entire process is automated via deployment scripts and a CI server. Therefore, automated testing needs to be in place for each step. Traditionally, functional and acceptance testing are performed manually by a Quality Assurance department. However, this is error prone, and causes long wait times in the deployment process. Organizations need discipline to automate testing and ensure proper testing context for each environment.

Continuous Integration and Continuous Delivery

It is outside the scope of this paper to cover the full gamut of continuous integration and continuous delivery (a more detailed paper is here). But it is worth discussing these two practices in the context of application and platform deployments.

Let's define these terms:

Continuous Integration (CI)—In a continuously integrated environment, developers merge changes to the main source branch often. When these changes are detected, a CI server (such as Jenkins) automatically checks out the main branch and runs automated integration tests against the main branch.

CI allows early feedback from committed code changes to ensure the updates haven't broken anything when integrating with other code changes. CI relies heavily on automated testing and tooling.

Continuous Delivery (CD)—Continuous delivery extends continuous integration practices by adding a fully automated suite of tests, including acceptance tests, as well as an automated deployment pipeline that can be executed by the click of a button (or a programmatic trigger). The goal of CD is to ensure that at any point in time, software can be deployed in a consistent, low-risk manner.

It's important to note that continuous delivery does not mean that software is deployed every time that a commit is made to the master branch; this idea is continuous deployment. Continuous delivery simply means that software can be deployed at any given time if needed or desired.

CI and CD are important aspects in software deployment. Both practices implement techniques for highly automated, low-risk, and rapid software deployment. CD methodologies can also be applied to platform deployment; we encourage platform operators to learn and understand CD practices and techniques. We address this topic later on.

Application Onboarding

Many aspects of full-stack development have been automated with tools mentioned in this paper. But there is even more automation that can be done. Consider application onboarding.

Here, a development team wants to start a new software project. It could be for testing a new idea,

learning a new technology, or some other purpose. Starting a new application project often involves many steps like:

- Creating a project or repository in source control
- Creating a new project shell with a scaffolding generator such as Yeoman or Spring Initializr
- Creating test suites in the new project
- Create one or more builds in the CI server
- Creating an Org and Space in Pivotal Cloud Foundry

Automate these steps! Many Pivotal customers do, with tools and scripts that handle each job.

It is difficult to create a "one-size-fits-all" tool for this purpose, since every enterprise's environment is different. Examples always help, so check out how Kroger accomplished this in the "Learning from Your Peers" section.

Deploying to Pivotal Cloud Foundry

So far, we've discussed how Pivotal Cloud Foundry drastically reduces the complexity and challenges associated with deploying applications within enterprises. Now, let's examine how Pivotal Cloud Foundry can provide value in all areas of software delivery.

`cf push`

Platform customers interact with Pivotal Cloud Foundry through a simple command line interface (CLI), the "cf cli". The cf cli exposes all PCF features and interactions through a set of simple commands. One of the most used commands is "cf push", which deploys and runs your application on Pivotal Cloud Foundry.

The cf push command is simple, but it does a lot for your code. That's why it's so magical!

When you cf push your application code and a few parameters, Pivotal Cloud Foundry automatically:

- Uploads your application code
- Determines and installs the correct runtime and middleware to run your application
- Sets up a route (or URL) for your application
- Creates a load balancing entry for your application
- Creates SSL termination for your application
- Creates health monitoring and logging subsystems for your application
- Starts your application in a healthy state with the desired number of instances
- Binds any backing services needed to your application by injecting the service credentials

`cf push` eliminates most of the "help ticket" silos that exist in today's enterprise IT teams. And, this action gives developers and operators a well known, consistent, and reliable mechanism for deployment.

How PCF Does Containers for You

One of the components in Pivotal Cloud Foundry that speeds deployment are buildpacks. A buildpack detects the type of application being deployed (JVM-based, .NET, Python, etc.). It then "builds" the correct runtime, middleware, and shared library stack to run the app. Buildpacks enable developers and release engineers to deploy application code that's automatically configured to run, **while at the same time** maintaining governance and control over the runtime definition!

When a cf push command is performed, the application code is uploaded to Pivotal Cloud Foundry. At this point, a "staging" process is started by the Cloud Controller in the platform. The staging process invokes a "detect" method in each of the installed buildpacks to determine which one is applicable. Then, the applicable buildpack's compile method is called. The output of the compile method: a container image called a "droplet", which includes the application code as well as any other binaries and libraries required by the application. A droplet is then used to create the containers that host the application. Figure 2 illustrates the workflow between components.

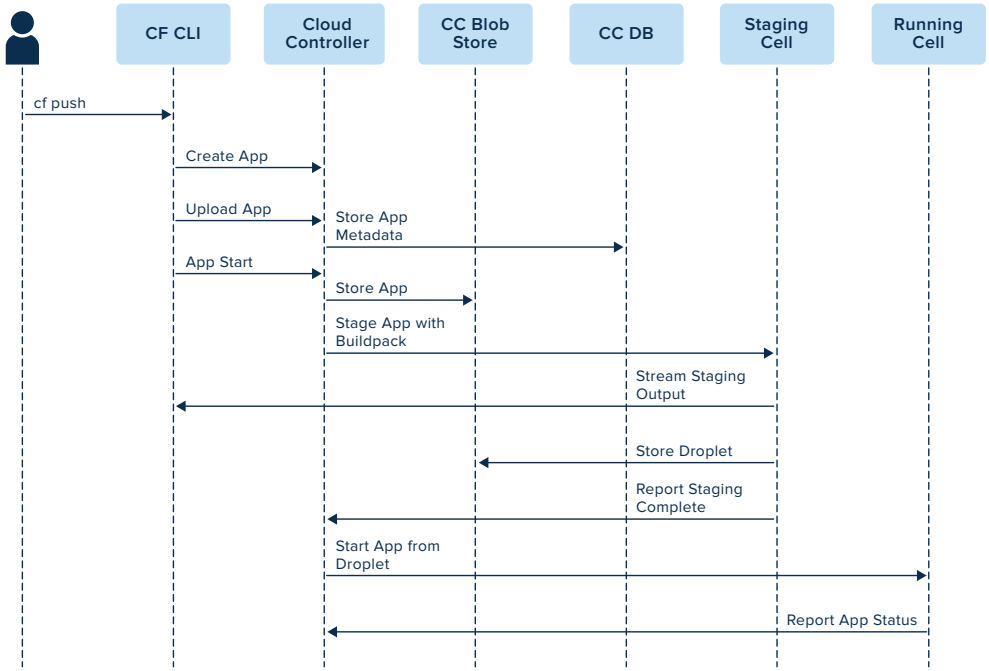

Figure 2: The automated workflow triggered by a cf push command.

Containers

Containers are gaining a lot of popularity in the software industry as a preferred method for packaging and deploying software images. Containers provide resource isolation in a multi-tenant environment, high application density on infrastructure, and portability and scalability for applications.

Pivotal Cloud Foundry uses containers to host application workloads, perform staging and to execute short-lived tasks. The component used in all these areas is a Diego Cell. Diego Cells can support both Linux-based and Windows-based workloads, providing polyglot deployment and runtimes for Java, .NET, and most other popular development frameworks.

Pivotal Cloud Foundry uses Garden for the container front-end API. Container back-ends for Linux workloads use kernel primitives (like control groups and namespaces) for resource isolation. Windows back-ends use a combination of Hosted Web Core (HWC) and Windows kernel primitives for this purpose.

Containers are a functional primitive in Pivotal Cloud Foundry, in contrast to other cloud application platforms. Containers are orchestrated as part of a larger system; developers don't interact with them directly. This is a central concept in Pivotal Cloud Foundry. Companies are most successful with gaining velocity and confidence in software delivery when containers are the output of an automated deployment pipeline. When containers are created outside of the deployment in a manual fashion, development teams tend to waste time on configuration and troubleshooting, not valuable work on writing code.

In some cases, deploying a pre-defined Docker image is convenient (such as when working with pre-packaged software or developer bootstrapping). Pivotal Cloud Foundry supports this as well, so this option should be used sparingly for reasons just mentioned.

Deployment Pipelines

Although cf push is a very simple way to deploy and run your applications, it is rarely used in practice. As discussed earlier, it is crucial to automate everything in the software lifecycle, including application deployment. Deployment pipelines are the preferred, recommended way of getting your applications to Pivotal Cloud Foundry.

Deployment pipelines consist of discrete tasks (like checkout, compile, package, and test). These tasks are tied together by a set of input and output resources. Pipelines represent a multifaceted deployment process, including error handling and rollbacks. In contrast, traditional deployment scripts simply execute a series of commands.

Successful teams adopt CI/CD along with Pivotal Cloud Foundry. They integrate their build servers with the platform; Jenkins (via plugins) and Concourse are the most popular choices.

Application deployments to Pivotal Cloud Foundry, as well as other CI/CD operations are simply tasks within a broader pipeline process. For example, a simple deployment pipeline would execute the following:

- Checkout the latest version of the main branch
- Compile and package the source code
- Deploy to Pivotal Cloud Foundry
- Run smoke tests
- Run end-to-end tests

Pivotal has created an opinionated set of deployment pipelines as part of the Spring Cloud project: Spring Cloud Pipelines. These pipelines, available under the open-source Apache License, represent Pivotal's recommended pipeline process for configuring, testing, and deploying microservice-based applications to Pivotal Cloud Foundry. The project contains both Concourse and Jenkins resources. These are useful starting points for developers deploying to Pivotal Cloud Foundry.

Deployment pipelines should be considered versionable artifacts—just like source code! After all, deployment pipelines are a crucial part of an application's lifecycle. Changes should be tracked, versioned, and rolled back if there are issues.

Backing Services

Backing services are any external resources that an application needs to consume. As per The Twelve-Factor App, all attached resources, whether local or remote, should be treated as backing services and accessed via a URL (or locator), and credentials over a network.

Pivotal Cloud Foundry enforces this architectural principle through the Service Broker API. The Service Broker API decouples applications from the services they consume through a well-defined contract consisting of 5 lifecycle phases:

1. **Catalog**—advertises services available and their capabilities
2. **Create**—creates a service instance, such as a database or message queue
3. **Bind**—exposes connection information and credentials to the application and its container environment
4. **Unbind**—removes connection information and credentials from the application environment
5. **Delete**—destroys the service instance

The Service Broker API provides a consistent, self-service experience for developers to discover, create, and connect to backing services. Backing services may be managed as part of Pivotal Cloud Foundry, or they may be external enterprise services. The Service Broker API conveniently abstracts service interactions, regardless of how (or where) they are managed.

Application Deployment and Backing Services

When deploying applications to Pivotal Cloud Foundry, applications can bind to services as part of the process. Simply define the name of services to bind to in the application manifest. NOTE: Service bindings are needed for a fully automated deployment process. Why? Applications typically require a connection to the backing service to start and operate properly. Don't overlook this step!

One thing to watch out for: the application and service instance need to exist in the same "target" Org and Space, or deployment will fail.

Customers usually ensure that this coexistence is properly configured as a task in the deployment pipeline, just prior to the deployment task. It's easy to automatically create the service instance if it's not there already.

Pivotal Cloud Foundry Marketplace

The Pivotal Cloud Foundry Marketplace advertises many of the Service Broker APIs available to developers and release engineers. It provides an interface to discover useful add-on services, their capabilities, and understand their associated plans. Developers can also create, manage, and delete service instances.

Pivotal offers several marketplace services that can be added to a Pivotal Cloud Foundry deployment. Many 3rd party integrated services are also available, across these categories:

- Relational and nonrelational data services
- Application autoscaling
- Caching
- Persistent storage services
- Single sign-on
- Distributed system coordination (service registry, configuration, circuit breaker dashboard, tracing)
- Task and batch job scheduling

For a full list of available services, visit Pivotal Network. Descriptions are available within the PCF Apps Manager interface as well. General business descriptions of add-on capabilities are described in the PCF Marketplace site.

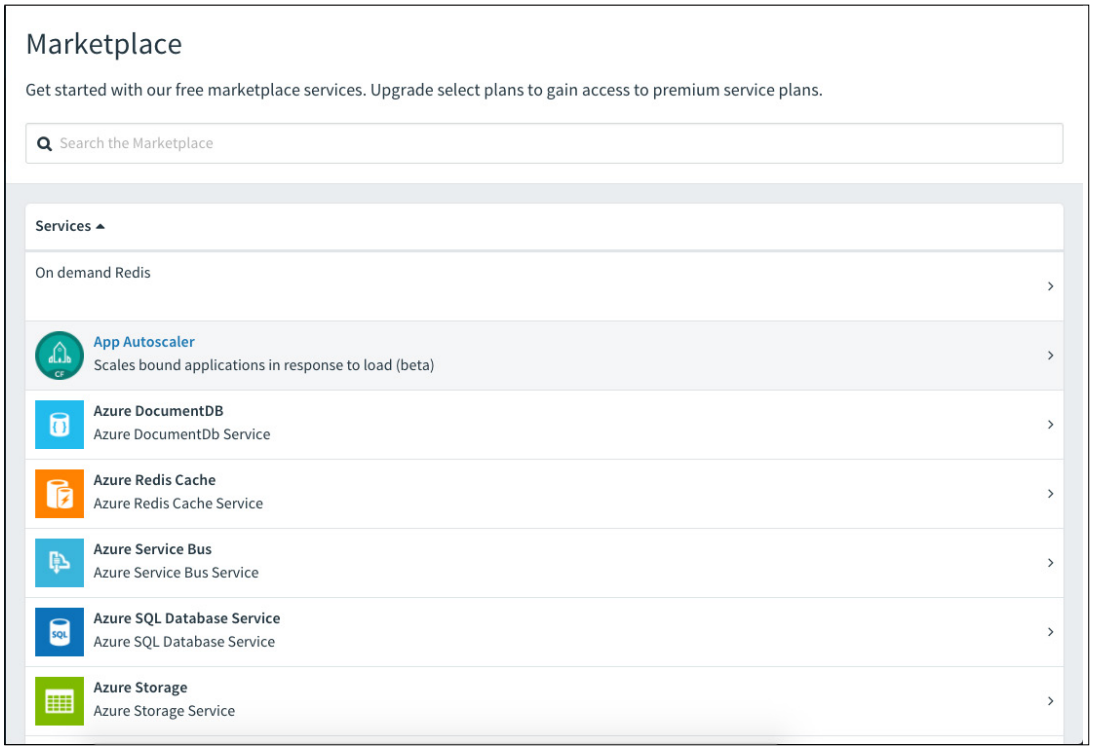

Figure 3—The Pivotal Cloud Foundry marketplace within Apps Manager. Engineers can browse and install add-on services.

Enterprise Integration

We didn't forget about your legacy systems, the tools with all your customer and product data. When you need to attach and consume these services in Pivotal Cloud Foundry, you have two options:

- Create a custom Service Broker API that interacts with the service and exposes the catalog, create, bind, unbind, and delete methods to PCF. To developers and release engineers, this brings welcome consistency. They can interact with the service exactly as they would Pivotal Cloud Foundry-managed services.
- Create a user-provided service. A user-provided service is a method of providing service connection information to the application environment, without implementing a Service Broker API. This option allows applications to bind to services with the same mechanisms. However, it does not allow users to discover and create new instances of the service.

How do you choose the best option? It comes down to how much control the developer needs over the service lifecycle. If a developer simply needs to connect to an existing enterprise resource, like a shared Oracle database or an existing enterprise web service, then a user-provided service is probably sufficient. If a developer needs to discover, create, and delete instances of the enterprise resource, then choose a custom Service Broker API. Regardless of the integration approach, the cloud-native applications bind to the enterprise resource without any changes in code.

Deploy Your Platform as Fast as Your Custom Code

We've discussed how developers can accelerate the delivery of their custom code with Pivotal Cloud Foundry. What about operators? How can they put some of these same practices into action? To become a cloud-native, you should roll platform updates frequently. Once again, let's start by examining the current state of affairs.

Legacy middleware platforms have the same characteristics as the apps that run on them: they are hard to update. They become out of date, insecure, and a detriment to the organization.

The cloud-native industry designed next-generation platforms to avoid these problems. Applying security patches and adding new platform features are relatively simple. Enterprise cloud-native platforms are secure, stable, and able to support business goals. This is only possible because it's easy to keep the platform current.

For our part at Pivotal, we release platform updates early and often. We deliver security patches weekly or even daily. Major feature releases happen every 3 months. New updates are nice—but they don't mean much if it's hard to upgrade!

Let's review how Pivotal customers deploy updates and new features in an efficient, consistent, and reliable fashion.

Unsurprisingly, automation is key to rapidly configuring and deploying PCF patches and updates. A key anti-pattern in platform operations is the notion of "snowflakes" or configuration drift with servers.

Immutable infrastructure ensures that deployed configuration is never changed; instead new infrastructure is created and deployed to replace the existing infrastructure. This way, infrastructure is always in a known good state.

Automating Pivotal Cloud Foundry deployments involves 2 broad steps:

1. Establishing a reference architecture and configuration for the platform. This is typically a collection of diagrams or drawings that represent deployment topologies, network layouts, routing and load balancing, and integration with underlying infrastructure.
2. An automated pipeline or process that represents the reference architecture. The pipeline can consistently and reliably configure and deploy the platform. Much like application deployment, this automation should deploy the platform, run automated smoke tests, and handle failures and errors along the way.

Pivotal provides reference architectures and Concourse pipelines to automate the deployment of PCF. These artifacts are based on best practices learned through hundreds of customer installations. We highly recommend that operators start with these artifacts when developing a reference architecture and a corresponding deployment pipeline. The reference architectures can be found in the Pivotal documentation site (an example is shown in Figure 4). Concourse pipelines can be downloaded from Pivotal Network; contact your Pivotal account team for access to these pipelines.

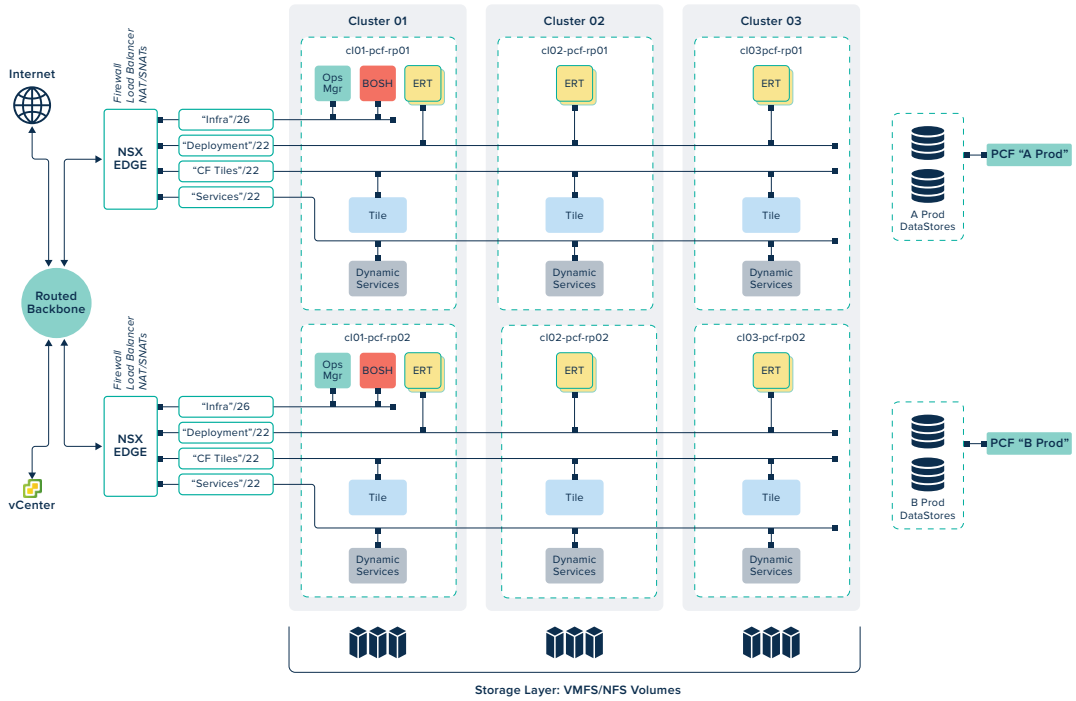

Figure 4: A recommended reference architecture for Pivotal Cloud Foundry on VMware vSphere and NSX.

Another best practice: have a sandbox Pivotal Cloud Foundry environment to test platform patching and upgrades, as well as new configurations. Most PCF customers have pre-production and production environments. This is insufficient—pre-production environments may not be production for your customers, but it is production for your development teams! Any downtime or issues for pre-production

environments should be avoided if possible, because it slows your engineering teams down. Deploying to a sandbox environment first helps avoid issues and downtime.

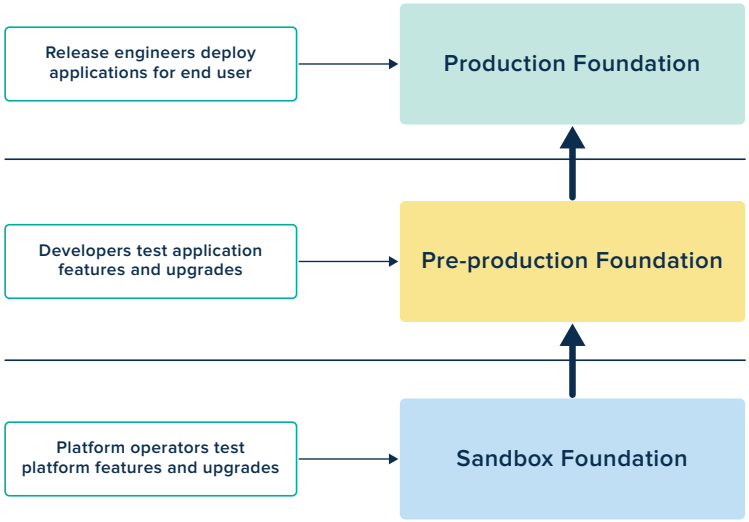

Figure 5: The functions of different PCF environments in most enterprises.

SLAs, SLOs, and the Operational Impact of Rapid Deployment

With long deployment cycles, organizations can schedule application downtime without significant impact to the business. When apps are deployed every 4, 6, or even 8 months, it's no big deal to have an app offline for a few hours on the weekend. Even if an application is unavailable for 3 hours every 6 months, it still easily meets a service level objective (SLO) of 99.9% uptime.

When applications are deployed much more frequently, say every week, the application could only be unavailable for 1 minute when being deployed to meet the same 99.9% uptime SLO. Incurring any scheduled downtime during application or platform deployment is unacceptable when deploying this frequently. So what does the IT organization do in this scenario? Zero Downtime deployments! The concept, in a nutshell:

> Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.
>
> At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.
>
> As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is not live: in this example, Green. Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.
>
> This technique can eliminate downtime due to application deployment. In addition, blue-green deployment reduces risk: if something unexpected happens with your new version on Green, you can immediately roll back to the last version by switching back to Blue.

Blue-Green deployments help you achieve zero downtime deployment and automated testing of both platforms and applications with effective strategies for reverting changes if issues do occur. Refer to Running Microservices with Pivotal Cloud Foundry for more details on this idea.

Conclusion

In order to stay competitive, enterprises need to be able to design, build, and deliver software solutions that change how companies offer valuable services, engage customers, and expose new revenue channels. Today's deployment processes are an obstacle to this objective.

Pivotal Cloud Foundry and its ecosystem of tools represent a high-impact, low-risk path to accelerating software delivery. The adoption of recommended patterns and practices helps get code into production as quickly as possible. In addition, opinionated mechanisms like the Service Broker give developers the freedom to extend their apps in an infinite number of ways, without sacrificing velocity. And finally, IT ops teams can support this transformation with a continuously updated platform that simplifies upgrades and patch management.

These strategies and tactics have proven to help large enterprises grow revenue in a constantly changing software-driven market.

Learn from Your Peers—How the Best Companies Accelerate Software Deployment with Cloud Foundry

Kroger: The start of Kroger's Journey to Cloud Foundry

Kroger, an early adopter of Pivotal Cloud Foundry, explains how operations and development teams embraced PCF and rapid deployment. They also demonstrate Kroger Internal Cloud (KIC), an automated onboarding tool that provisions all of the tools in their software development and deployment process.

The start of Kroger's Journey to Cloud Foundry

GE Embraces Its Future as A Digital Industrial Leader

Taking the reins of its digital future required GE to undergo massive transformation, both technologically and culturally. From a technology perspective, this meant developing a software platform to support the burgeoning Industrial Internet. In contrast to the consumer internet, the Industrial Internet refers to the interconnection of industrial equipment and other high value, complex machines that support mission critical market sectors like energy, transportation, and healthcare.

Agile software development was was a new concept for GE, so it worked closely with Pivotal to instill the processes and values in GE's developers.

Learn more

DISH: A Modern Platform That Allows Developers to Save Hours and Days

DISH is the very best at delivering video anywhere, anytime because innovation and entrepreneurship are part of its DNA. The company operates in the confluence of the highly competitive technology, entertainment and telecommunications industries. To be successful requires DISH to continually incorporate new technology to provide the services customers want.

Engaging with Pivotal has offered the perfect way for DISH to learn different and valuable techniques to deliver software faster. Thanks to a comprehensive, future-facing development platform and proven methodologies, DISH developers are quickly building the software that is enabling DISH to better meet the needs of customers and come to market with cutting-edge services ahead of competitors.

Learn more

Allstate: Open-Source Tools for Zero-Downtime Deployment
Allstate has created a team focused on innovation and cloud-native development called Compozed. This group of engineers built a tool to provide zero-downtime application deployments to multiple PCF Foundations. This tool, aptly named Deployadactyl, is available as an open-source project. Teams can also use Deployadactyl to rollback updates if something unexpected occurs.

Recommended Reading

eBook: Cloud Foundry: The Cloud-Native Platform, Duncan C.E. Winn

eBook: Crafting Your Cloud-Native Strategy, Michael Coté

Whitepaper: Speed Thrills: How to Harness the Power of CI/CD for Your Development Team

Whitepaper: Running Microservices on Pivotal Cloud Foundry

Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Jez Humble and David Farley