WHITE PAPER

Multi-Site Pivotal Cloud Foundry: Deployment Topology Patterns & Practices

By Michael Krumpe, Jared Ruckle, Guillermo Tantachuco, Adam Zwickey, and Chris DeLashmutt

Pivotal

Table of Contents

Introduction ... 3

Learn From Your Peers ... 3

Your Applications Need Reliable Microservices ... 4

Multi-Site, Multi-Foundation Microservice Availability ... 4

Multi-Site Service Discovery: An Eventual, Inevitable Requirement ... 5

Client-Side Load Balancing with Ribbon ... 7

Let's Talk About Data ... 8

Caching and Data Grid ... 9

Three Pivotal Cloud Foundry Deployment Topologies ... 10

Deployment topology # 1: Active/Active ... 11

PCF Foundation availability ... 11

Deployment topology # 2: Disaster Recovery (D/R) ... 14

Deployment topology #3: Public cloud as extension of the datacenter ... 15

Handling Bursts of Traffic: Autoscaling in Secondary Sites ... 16

Conclusion ... 17

Appendix A: Learn from Your Peers - How the Best Companies Deploy Multi-Site Pivotal Cloud Foundry ... 18

Allstate ... 18

Comcast ... 18

T-Mobile ... 19

The Home Depot ... 19

Appendix B: BOSH Backup and Restore ... 19

Burning Down the House: How to Deal with Disaster Recovery in Cloud Foundry ... 20

How It Works: First, Create The Backup Artifact. Then, Put It Back. ... 20

Introduction

Application resiliency, high availability, speed of responsiveness, business impact, service levels, disaster recovery, recovery point objectives. These are the things that drive C-level conversations, and keep IT leaders up at night. Excelling in these areas is crucial to the success of an enterprise today.

Adopting a cloud-native platform and microservices raise the expectations that business units have for an IT organization. As development velocity increases, cloud-native software development principles, horizontal scaling, and elastic computing put a new twist on the old theme of "availability". It's inevitable, then, that multi-site availability becomes a standard requirement, no longer a luxury.

So how do enterprises do it? And how can you do the same?

Learn From Your Peers

In this paper, we will review how Pivotal Cloud Foundry (PCF) customers achieve different levels of multi-site availability.

Enterprises have a mix of environments and systems across various infrastructure and data center types. As hybrid IT becomes the new normal, production environments blend on-prem data centers, off-site colo facilities, public and private clouds, siloed legacy systems, and the beloved mainframes.

Under this backdrop, various deployment methodologies are used, depending upon the type of availability and scalability required. This paper will cover the major scenarios - and corresponding best practices - customers have implemented to support the availability of Pivotal Cloud Foundry. Each pattern could be used singularly, or in conjunction with other methods. For example, while two datacenters may be run as Active/Active. One site may also be backed up and used as a warm standby location to a third data center.

To maintain an always-on environment, an enterprise must consider people, processes and technology. In this document, we will only focus on technology. It is important to stress that Pivotal's primary subject matter expertise in this context is not on IaaS or network infrastructure design, but on Pivotal Cloud Foundry's behavior and performance in various IaaS environments.

Your Applications Need Reliable Microservices

Horizontal scaling patterns and distributed systems gave rise to microservices; they are the architecture of choice for high-velocity development teams. Let's begin our journey at the microservices layer and explore how Pivotal Cloud Foundry supports reliability and resiliency.

Microservices adhere to 12-factor principles, and are stateless by nature. When an enterprise designs a microservice pattern, components become smaller. Ensuring availability often requires multiple copies of each service, distributed over HA infrastructure with dynamic routing. Further, as an app scales, individual instances of the app, hosted within numerous containers across various platform clusters, still rely on data in some form or fashion.

Traditional approaches to active-active patterns fall short with microservices. You need to think differently.

Multi-Site, Multi-Foundation Microservice Availability

Let's examine an enterprise that handles health checks against systems, applications, and failover at the microservice level, rather than the Pivotal Cloud Foundry (PCF) foundation level.

The following diagrams show their approach. Figure 1 describes PCF foundations across sites.

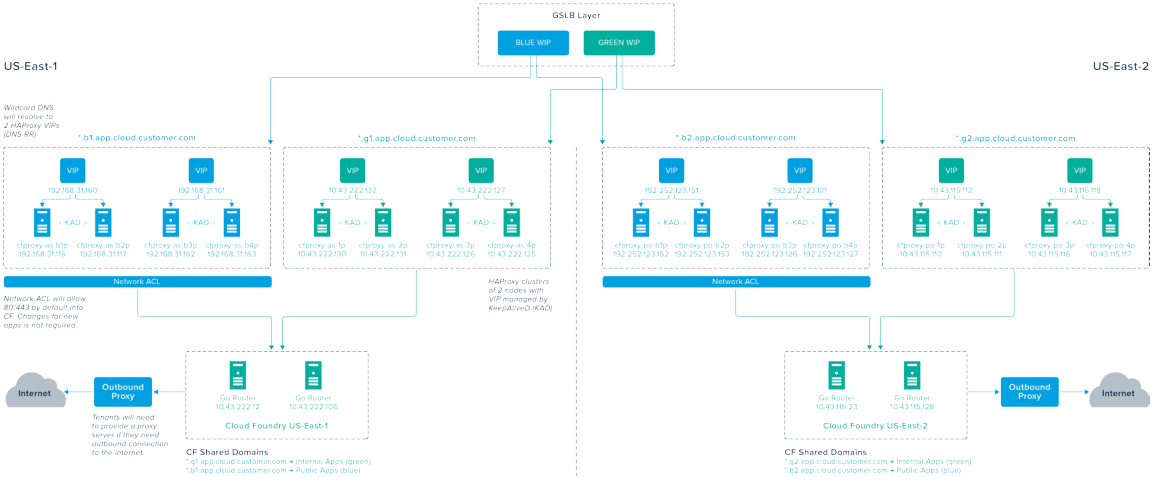

Figure 1 - A reference architecture with PCF foundations across sites, an active/active deployment. Global server load balancers (GSLB) route traffic across two data centers. The "blue" and "green" foundations span locations, boosting availability. Network access control lists (ACLs) act as a firewall for controlling traffic in and out of Cloud Foundry.

Now consider Figure 2, with microservices deployed across all PCF foundations.

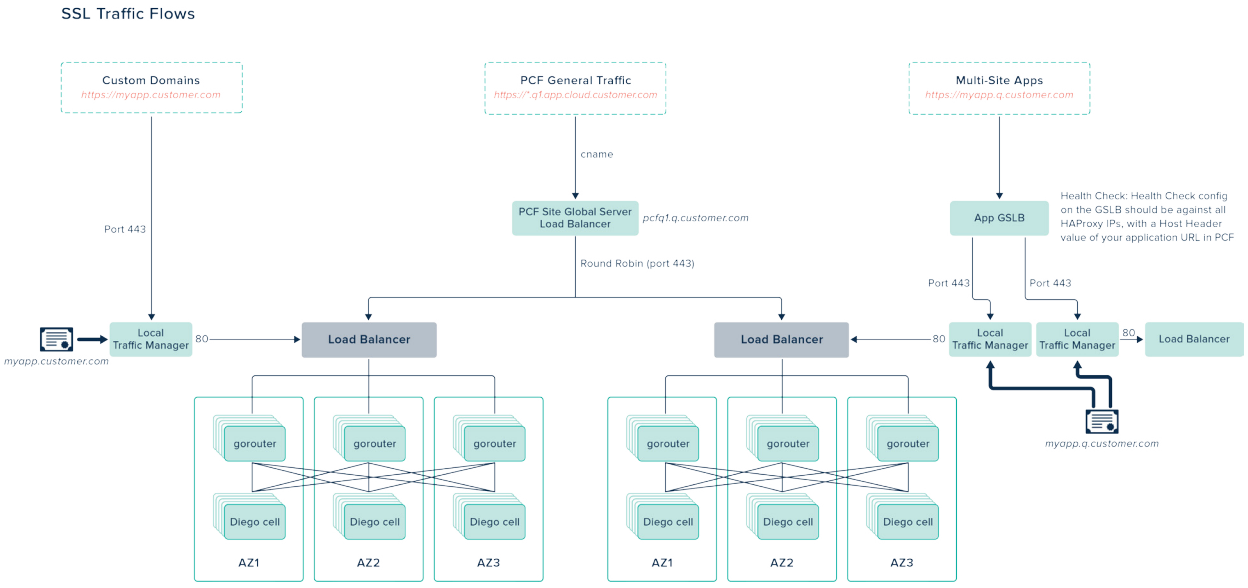

Figure 2 - Failover at the microservice level. Network engineers ensure availability by configuring a local URL for the microservice and SSL termination. This is done in the local traffic manager at each site. From there, admins define a global URL for the service (as well as a health check policy) in site to site global server load balancers. This inspects the state of apps running within the PCF foundations and reacts accordingly.

Let's review behavior for both Figure 1 and Figure 2. If the health check is successful, the site's local traffic manager will then balance traffic across PCF's HAProxy pool. However, if the health check for that service is not successful, even though the PCF foundation may be operational, the global load balancer will direct traffic to another app instance available in a different site.

Multi-Site Service Discovery: An Eventual, Inevitable Requirement

One key tenet of a microservice-based architecture: the Service Discovery pattern. Microservices architecture should use a Service Registry to dynamically discover and call registered services. For this purpose, Pivotal offers the Service Registry for Pivotal Cloud Foundry.

Over time, multi-site service discovery becomes an essential pattern for adopters of microservices. Pivotal customers use Spring Cloud Services for this requirement. Operators configure peer replication of Service Registry service instances.

Now, when an application registers with a Service Registry instance, that same application is available for lookup by consumers in a PCF installation in another data center, or across organizations within a single PCF installation. Here's a conceptual look at this feature in Figure 3.

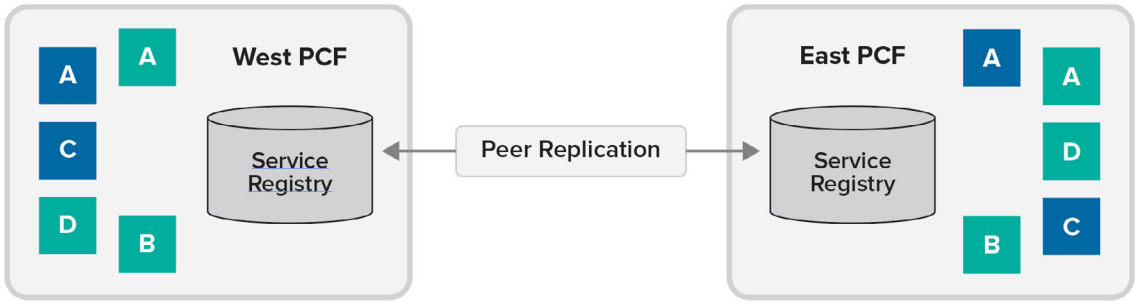

Figure 3 - Multi-site peer replication for service registration.

Service Registry peer replication works across PCF installations. It also works across organizations within a single PCF installation. Figure 3 above shows a simple Service Registry peer replication configuration across two sites. If we look at the "West PCF" data center, the local Service Registry registers the A and C services in light blue. They are also made available for lookup in the "East PCF" data center. The same is true of services in East PCF. The Services Registry there registers the services in light green locally, then makes them available for lookup in West PCF.

In practice, the feature works just like it's described above. That's because the commercial service registry feature in Spring Cloud Services has its roots in the Eureka, a NetflixOSS project. Figure 4 shows this configuration as part of a larger Pivotal Cloud Foundry environment.

More details are available in a recent blog and in documentation.

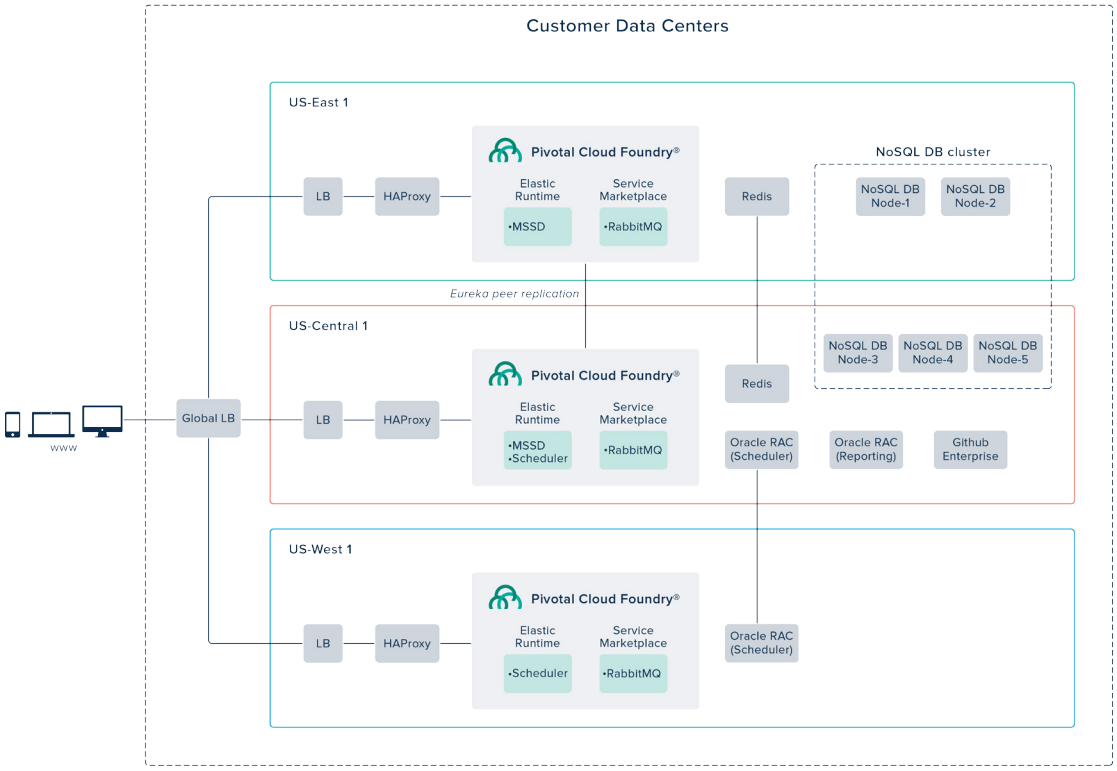

Figure 4 - Multi-site service registry in an active/active deployment. The "Eureka peer replication" makes services discoverable in across two sites. This is shown as MSSD (multi-site service discovery).

Once peer replication is enabled for service discovery, development and operations teams enjoy:

- Increased availability for their microservices, even in the face of partial failures
- Load balancing across network boundaries
- Successful discovery of services across distinct PCF installations or organizations

Client-Side Load Balancing with Ribbon

For multi-site deployments, customers may choose to deploy a client-side load balancer. This way, developers have control over the behavior of HTTP and TCP clients. Pivotal customers use the Netflix Ribbon project in Spring for this purpose.

Ribbon reduces load on a centralized load balancer. It uses client-side load balancing to make intelligent decisions about where to send a request. Ribbon integrates with Service Registry to ensure that clients have an accurate representation of the landscape.

Figure 5 examines how Ribbon is often deployed in a multi-site PCF environment.

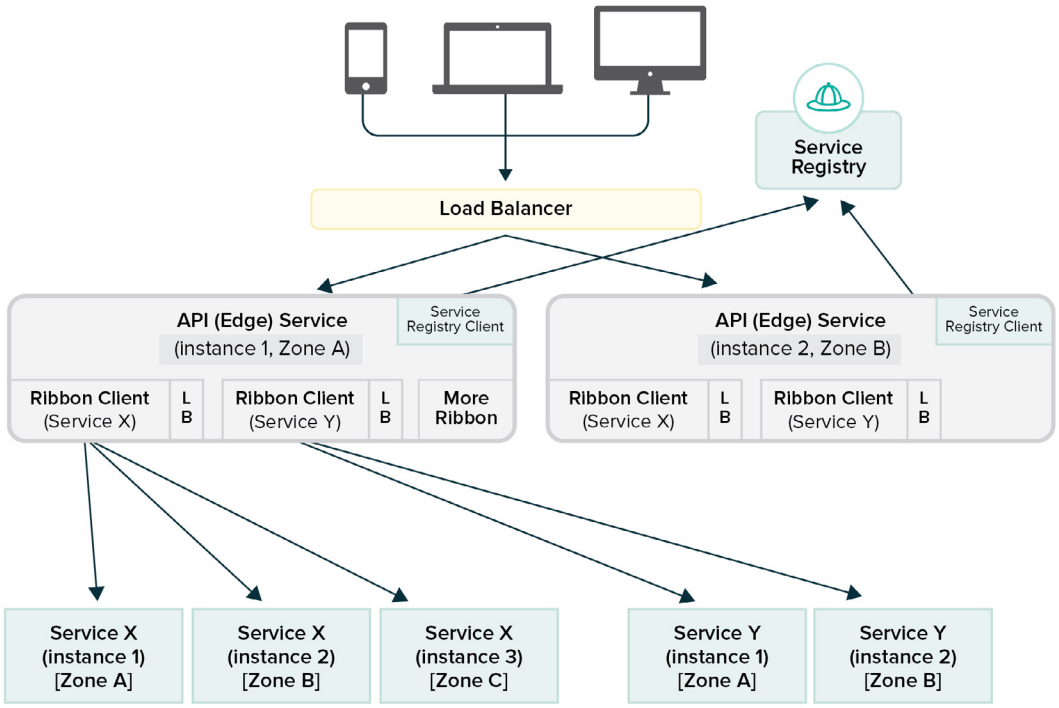

Figure 5 - Deployment topology of Ribbon with a multi-site PCF environment.

The load balancer handles public traffic. Internal mid-tier requests are handled via the Ribbon framework. The Service Registry registers the application's services. As a best practice, Ribbon clients should be created and configured for each target service. Ribbon's Client offers a good set of configuration options (connection timeouts, retries, retry algorithm choices, etc.).

Let's Talk About Data

Microservice application architecture patterns are well-known and can be followed with precision. Data, however, is not such a clean area to address in multi-site scenarios. Thankfully, there are methods and tools that can help. But do not expect the DBA to make these decisions for you! You must consider how data is handled at the application layer as well.

Traditional architectures feature a primary site and a secondary site. The system sends database logs from the primary data center to the secondary location in regular intervals. This action replays data streams to update data tables on both ends. However, most Pivotal customers desire something different and more powerful: bi-directional data replication. Here, the system replicates data in near real-time, between sites. Both sites serve traffic and process requests. This approach boosts reliability and resiliency of the application.

There are some important considerations to keep in mind when deciding how to replicate data:

1. **Data consistency** - There may be trade-offs in performance and real-time consistency. Distance, latency, and load are factors that affect the system at any given time. The asynchronous nature of data updates makes this a constant struggle.
2. **Replication direction** - One-way versus bi-directional data replication choices can have an outsized impact on the system. A substandard implementation introduces needless complexity. It can be dangerous to the integrity of the data. Conflicts between the data sets need to be handled in a methodical way to sanitize bad records.
3. **Choose the right backing store for the job** - Utilize the right type of data services for the right layers of the application. For example, a caching layer that can handle high-speed updates across multiple sites helps availability and performance. But it also requires additional rigor to normalize that data down into a relational database.

For data availability, application owners (typically a product owner or engineering manager) are responsible for selecting an external data grid or data store that provides the right type of replication. Let's review popular approaches: caching and data grids.

Caching and Data Grid

To replicate data across the WAN, apps may use one of the following patterns.

Microservice Caching

Pivotal Cloud Cache (PCC) is a service-bindable cache that features the same in-memory architecture as Pivotal GemFire. PCC is squarely focused on making caches of data available across instances of a microservice or layers of an application.

The product helps customers achieve bi-directional data replication across sites, as shown in Figure 6. As changes occur, PCC creates events in a replication queue, then sends them to the other site. Events can be replicated in user-specified intervals ("every 5 seconds"), or after a defined count ("every 1000 events"). PCC is also able to compress and encrypt events before sending. Upon receipt at the other site, it decompresses and decrypts the data.

Either cluster can perform read/write operations. PCC also handles de-duplication and conflict resolution. This is an important feature that ensures 100% data redundancy across locations.

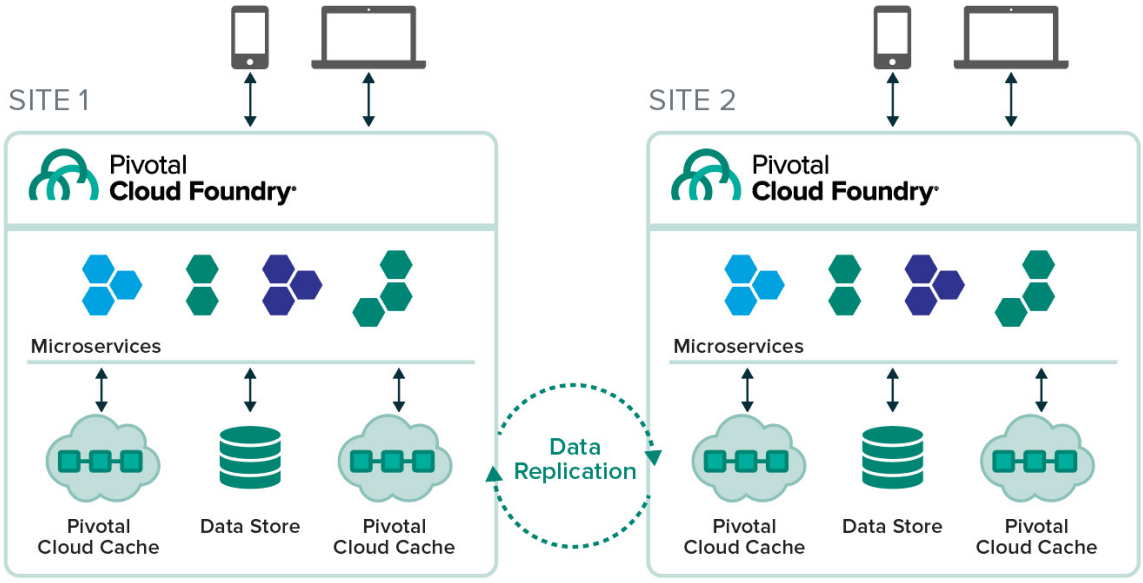

Figure 6 - Data replication across two sites, featuring Pivotal Cloud Foundry and Pivotal Cloud Cache.

More detail on microservices caching is in this whitepaper: In-Memory Data Caching for Microservices Architectures.

In-Memory Datagrids
Pivotal GemFire, an in-memory data grid, can be used as a cache and for fast WAN replication of transactional data. It's often backed by a "system of record" RDBS like Pivotal Greenplum.

Three Pivotal Cloud Foundry Deployment Topologies

Most enterprise systems have multiple points of potential failure. In the event of a disaster, many factors - technical debt, siloed teams, concentrated "tribal knowledge" - conspire against rapid recovery. Even a few seconds of downtime can mean lost revenue, dissatisfied customers and missed opportunities to win in a competitive market. Pivotal Cloud Foundry offers you several options to meet your business goals.

The four (4) levels of HA within Elastic Runtime (ERT) work exceptionally well inside a single foundation. However, in order to achieve high availability across multiple foundations and multiple sites, we need to do more. Customers gravitate to three common patterns.

- Active/Active
- Disaster Recovery
- Public cloud as an extension of the datacenter

Deployment topology # 1: Active/Active

In this option, an enterprise runs the same apps across multiple sites at all times.

Operators usually install a PCF/ERT foundation on each site. They keep all foundations identical by deploying the same apps to all sites. Updates and new features are continuously delivered to both sites as well.

Geographically distributing an app introduces the need for a consistent view of system state. This is a very difficult undertaking. So how do customers do it?

Many opt to deploy distributed data stores in different data centers, with redundant replicas for disaster recovery. Administrators thoughtfully choose the location of data centers, to avoid network latency across nodes.

Once an enterprise has PCF foundations and apps running on multiple sites, they monitor application availability across different levels of the architecture.

PCF Foundation availability

The deployment depicted in Figure 7 illustrates how a global service load balancer (GSLB) balances traffic evenly across PCF foundations. This is the familiar round-robin policy.

As in Figure 2, app owners are responsible for choosing an external data store that provides data replication capabilities. Here, apps use an external Oracle database with “read” replicas in both data centers. However, writes are made to the primary database in a single data center.

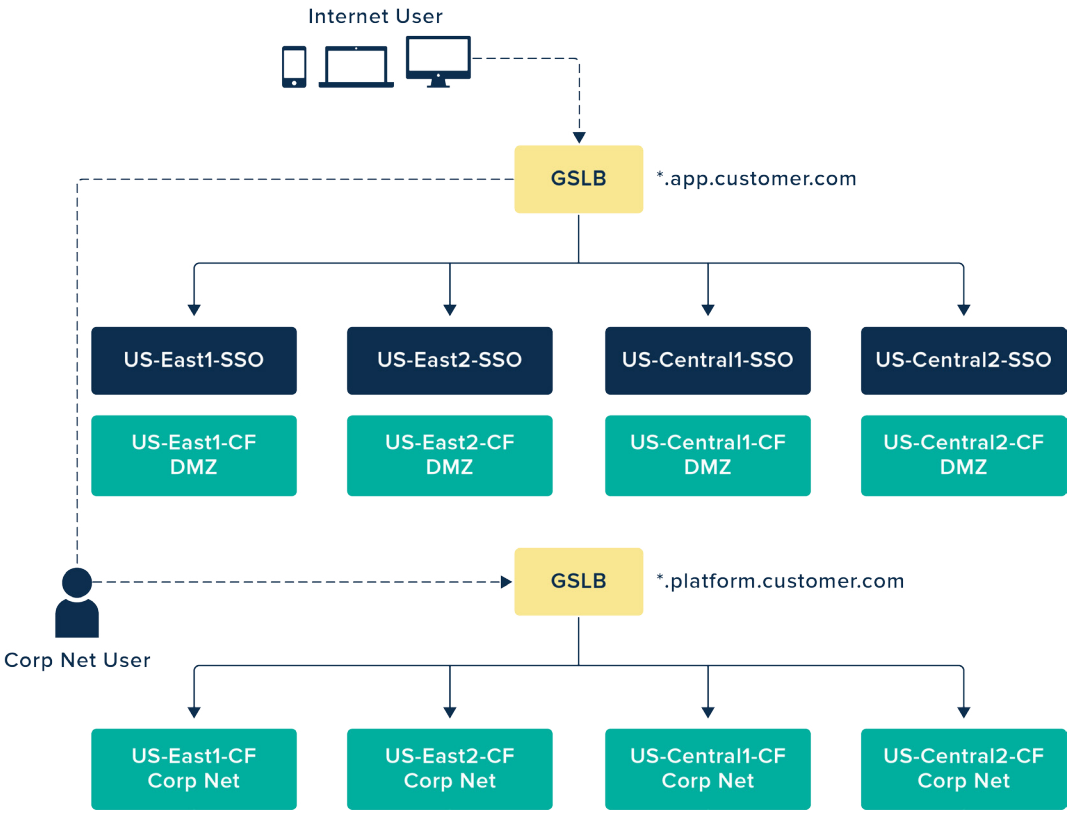

Figure 7 - Round-robin distribution of traffic in an PCF acte/ctive deployment. Single Sign-On (SSO) is used to verify the identity of application users.

What happens when a failure occurs in the architecture shown in Figure 7? Let's consider the case where a PCF foundation becomes unresponsive.

Customers rely on one of the following networking elements to manually (or automatically) direct user traffic to another site running PCF:

- Private data centers use a global traffic manager (GTM) to send traffic to a given data center via a local traffic management (LTM). Traffic is directed to a site based on routing policies such as: round-robin, weight-based, latency-based, geolocation, and session affinity (cookie-based or client IP). Figure 8 shows routing based on geolocation.
- Public cloud consumers will use that provider's global load balancer service
- DNS-based traffic management
- 3rd-party services such as Akamai, Dyn, or Cloudflare

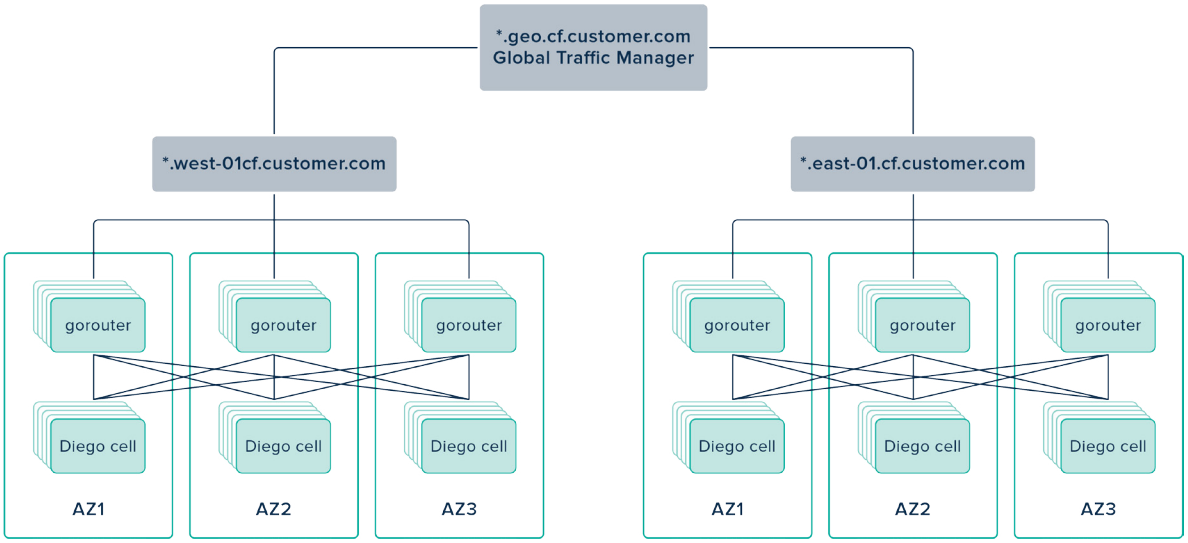

Figure 8 - An active/active deployment where a global traffic manager uses a geolocation policy to direct traffic. Users are routed to the PCF foundation physically closest to them.

On the application configuration side, Blue-Green deployments keep app configuration identical in each foundation, with no downtime. This is shown in Figure 9, using Continuous Integration (CI) and Continuous Delivery (CD) tools.

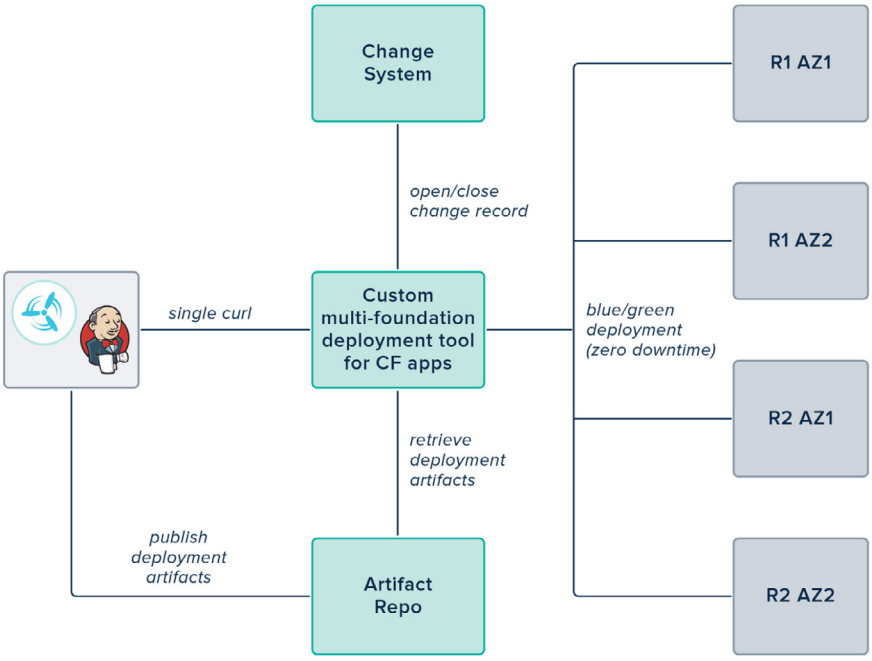

Figure 9 - This CI/CD process pushes applications to multiple foundations. Here, the foundations are in multiple regions (R1, R2) and multiple availability zones (AZ1, AZ2).

Deployment topology # 2: Disaster Recovery (D/R)

Instead of having a true active/active deployment across all layers, some Pivotal customers prefer to install a PCF/ERT foundation on a backup site. The backup site resides on-premises, in a co-location facility, or the public cloud.

Look at this option as more of a restoration pattern. It can be used in conjunction with other methods, and the DR site would be a stand-by location used to initiate a restoration. One thing to keep in mind: the DR site would not be available instantly in the event of a failure at the primary site. It may take hours to recover from a true disaster, even with various levels of sync and readiness in place.

The backup site includes an operational foundation, with only the most critical apps ready to accept traffic should a failure occur in the primary data center. This is shown in Figure 10.

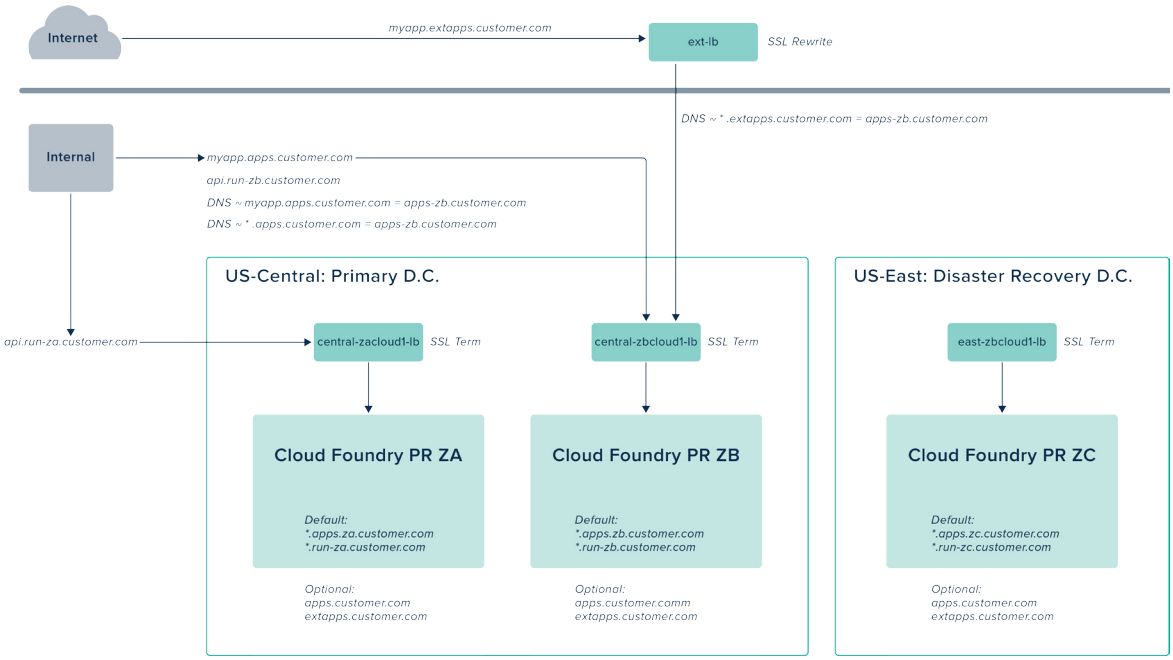

Figure 10 - An active/passive D/R deployment. Note the multiple foundations in the primary data center. The recovery site is brought online as needed.

Deployment topology #3: Public cloud as extension of the datacenter

Another popular hybrid configuration with Pivotal customers: using the public cloud as an extension of an on-premises datacenter. By utilizing best-in-breed public cloud services, apps can run on-prem and in multi-tenant, elastic infrastructure. This hybrid setup requires a high-speed, dedicated connection between your data center(s) and the public cloud site. PCF can deliver a consistent platform experience across multiple IaaS through the BOSH cloud provider interface (CPI). The CPI is an abstraction layer that manage and orchestrate the infrastructure underpinning PCF.

We recommend elevating the external IP management to global IP providers (such Verizon, AT&T, or BT). This way, you can consume global load balancing and border gateway protocol (BGP) services that extend the span of IP space, reaching down into the physical data center of the public cloud provider. Administrators can then configure firewall rules in the public cloud as LAN-to-LAN extensions and endpoints.

Policies can be set at the global provider IP layer to route traffic to the various sites either through an A/B fashion for failover, or throttled based on geographic location of traffic requests. Rules can be complex, however the application architecture and supporting services should be responsive, regardless of routing needs. PCF as a platform is compliant in either scenario.

Figure 11 shows an sample deployment with PCF running in an on-premises system and in the public cloud, Google Cloud Platform in this case.

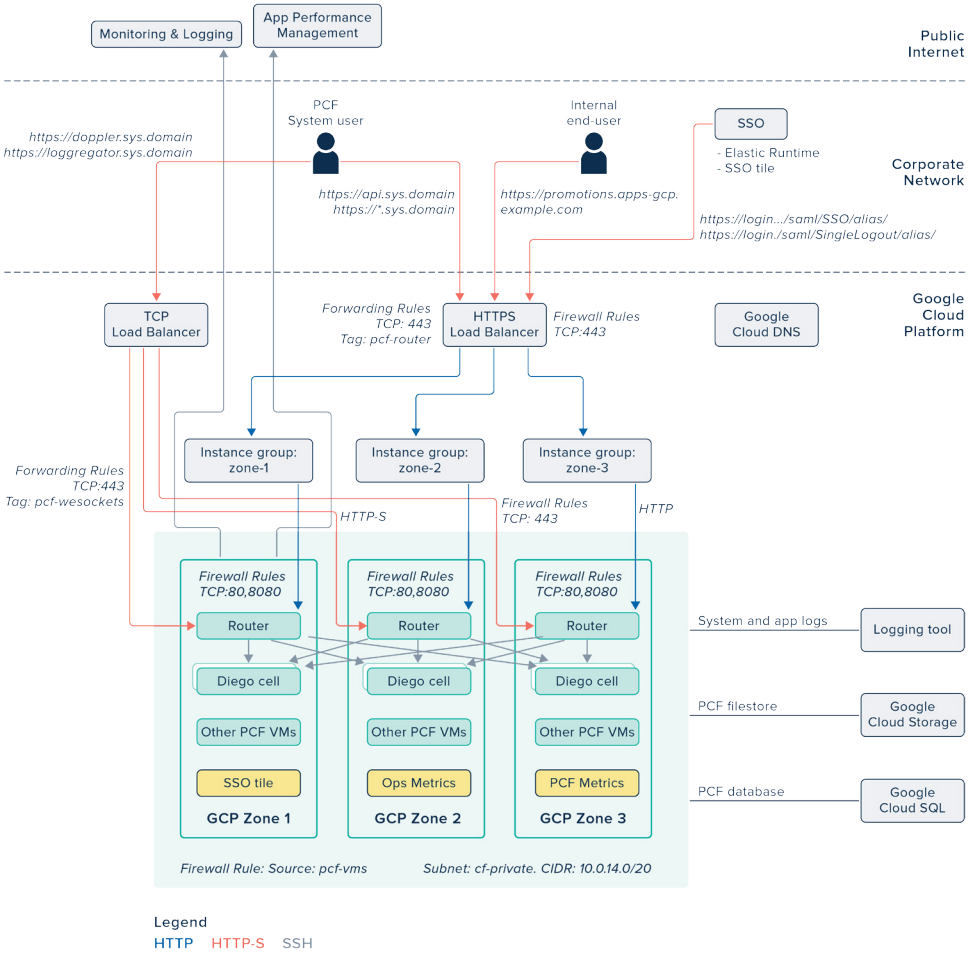

Figure 11 - A reference architecture for a hybrid on-premises and public cloud ERT deployment.

Handling Bursts of Traffic: Autoscaling in Secondary Sites

There's another flavor of this topology, where operators use the public cloud to offload spikes in short-term traffic.

Here, the public cloud is used for short periods of time, usually days or weeks. This short window often corresponds to a major business event (like a product launch, marketing campaign, or surge in seasonal traffic). The PCF App Autoscaler can handle scheduled scaling of applications, but not of the platform itself. Here, BOSH is used to scale the platform in conjunction with application instances. However, this can get costly in a public cloud environment. Administrators should use automated pipeline patterns with tools like Concourse for repeatability.

Depending on your organization's appetite for OpEx versus CapEx, it may be a more economical to use the public cloud for temporary spikes in traffic. Investments in private data center resources can be unattractive, if unused for a large percentage of the time.

Conclusion

Multi-site deployments of Pivotal Cloud Foundry are an eventuality for most enterprises. Early adopters of cloud-native platforms have pioneered reference architectures at the microservices, application, data, and platform levels. These designs should be strongly considered by new adopters of PCF. There may be a place for each deployment topology within a single enterprise.

Each approach offers unique benefits, summarized below. Carefully vet each topology according to your unique requirements and hybrid IT environment.

Deployment Topology	Benefit	Other Considerations
Active/Active	• Greatest availability; superior protection against unexpected failure • Flexibility and control to optimize traffic any number of ways • The best option for advanced microservices architectures	• Highest CapEx and OpEx • Best suited for your most critical apps • Ensure compatibility with CI/CD processes • Complex application load balancing patterns • Most complex data management scenario • Address related dependencies and integration points to fully support this configuration
Disaster Recovery (D/R)	• Offers a measure of fail-over, without requiring a full-featured secondary site • Offers a DR recovery plan for essential apps • Practical starting point for enterprises just getting started with Pivotal Cloud Foundry	• Longer recovery time in the event of a failure - usually hours, rather than minutes. • Still requires a secondary site, infrastructure, and foundation to operate in a “passive” state • Minimal protection for non-essential apps • Requires a review of RTO/RPO SLOs for business critical apps running on the platform.
Public cloud as extension of the datacenter	• Higher utilization of on-prem assets • Flexibility to elastic capacity on-demand as needed • Compliant with most enterprise InfoSec guidelines	• Initially complex to setup orchestration • Keeping platform versions in sync require ongoing management

Recommended Reading

New in Spring Cloud Services 1.2: Multi-Site Service Discovery

Secure Hybrid Banking Reference Architectures for Cloud-Native Applications

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

Running Microservices on Pivotal Cloud Foundry

Beyond the Twelve Factor App

In-Memory Data Caching for Microservices Architectures

Unlocking Operational Efficiency with BOSH

Appendix A: Learn from Your Peers - How the Best Companies Deploy Multi-Site Pivotal Cloud Foundry

Allstate

How does an 84 year old insurance company transform business and IT groups into a cloud-centric model?

A cloud platform like Pivotal Cloud Foundry certainly helps, but the organization also needs to embrace cultural and process change. This overall change across technologies and processes allows the company to do a better job creating excellent customer experiences and protect revenue.

“The growing variety of programming languages and frameworks supported by Pivotal Cloud Foundry offers developers at Allstate the choice they require. The platform takes care of the details of application container scheduling, cluster management, and scaling, greatly reducing the operational burden to reliably run Cloud-Native applications.“

Matt Curry
Director of Platform Engineering, Allstate Insurance

Learn More

Comcast

Deploying Pivotal Cloud Foundry within an enterprise as large and multifaceted as Comcast presents a number of challenges. The platform must scale while being capable of serving a number of application needs, spread across multiple teams. During a talk at the Cloud Foundry Summit in May, Comcast’s Neville George, Sam Guerrero, Tim Leong, and Sergey Matochkin discussed the decisions the Cloud Architecture team made when they introduced Pivotal Cloud Foundry to the operations team.

"We're constantly working to change and transform our experience, and add new features and services to the Comcast customer experience. We have a great bunch of vibrant developers who are constantly working to improve that experience. They, along with Pivotal Cloud Foundry, are at the heart of the Comcast customer experience transformation."

Neville George
Principal Engineer at Comcast

Learn More

T-Mobile

T-Mobile wanted Cloud Foundry, and they wanted it quickly. The target was an application receiving 12 million daily calls running on the platform in three months. With no IaaS and complicated politics, were they able to meet their deadline? Watch this video case study to learn how to overcome unexpected problems (technical and otherwise) that you may encounter in a large enterprise and how to address them.

Learn More

The Home Depot

From one team and some hardware in a closet to becoming the platform of choice for hundreds of developers across multiple data centers - what has The Home Depot's journey with Pivotal Cloud Foundry looked like in our first year? How did they get their development community to quickly adopt the platform? What are some things they did wrong and would like to help others avoid? What are some things they did right and would encourage? What were the technical, organizational, and people challenges along the way? Watch this video to learn about organizational change and driving platform adoption.

Learn More

Appendix B: BOSH Backup and Restore

There's no substitute for HA across sites. But it's nice to have options. To this end, the Cloud Foundry ecosystem has formulated a way to make backup and restore operations for Cloud Foundry much simpler. That service, BOSH Backup and Restore, recently became generally available.

A recent blog post from a Pivotal product manager describes how this feature can improve uptime, availability, and everything else that comes with those high expectations we opined on in the Introduction. Key excerpts are below.

The largest companies in the world run their most important apps on Cloud Foundry. Their stories were on display at Cloud Foundry Summit last month.

Operators have a range of approaches for ensuring they can recover Cloud Foundry, apps, and data in case of a disaster. The approaches fall into two categories: backing up the raw data or automating recreation of the data, and both have associated issues and complexity.

We set out to change that. Our recommended solution for the community: BOSH Backup and Restore.

Burning Down the House: How to Deal with Disaster Recovery in Cloud Foundry

What is BOSH Backup & Restore (BBR)?

> "BBR is a framework for backing up and restoring BOSH deployments and BOSH directors. It orchestrates triggering the backup or restore process on the deployment or director, and transfers the backup artifact to and from the deployment or director."

Our ideal solution needed to answer two questions. First, how do you take a distributed backup that's consistent? And second, how do you avoid modifying your backup scripts every time Cloud Foundry changes?

BBR does that by defining a contract between the backup orchestrator and the component to be backed up. The orchestrator calls scripts on the components to be backed up and restored, and the components are responsible for generating the backup and restoring the backup.

The orchestrator is the BBR binary, and the component is a BOSH job.

To enable consistency, the component (a BOSH release) can implement locking - a pre-backup-lock script and a post-backup-unlock script. Backups are triggered per BOSH deployment, with each pre-backup-lock script on all the jobs in the deployment getting called before each backup script is called.

The authors of the component write and maintain the backup and restore scripts for that component, and the scripts are packaged with the component. As a result, scripts can stay in sync with the component, avoiding compatibility issues. And the scripts can be smart, only backing up / restoring required data and, if necessary, performing processing like encryption or credential generation.

There's a lot of BOSH here. Isn't this supposed to be for Cloud Foundry? Well, yes! Consider that:

- All components in Cloud Foundry are BOSH deployments
- The BOSH director is a BOSH release
- For an operator, a BOSH deployment is the logical unit of backup

This our rationale for BOSH Backup and Restore. The key to making this all work: the responsibility for writing and maintaining backup and restore scripts sits with the BOSH release author.

We gave a talk at CF Summit 2017 on BOSH Backup and Restore. Check it out if you'd like to hear more about the service, and how we got to this point.

How It Works: First, Create The Backup Artifact. Then, Put It Back.

To understand how BBR works, let's look at the steps that happen once an operator initiates a backup. The BBR binary is run from a jumpbox that has access to PCF deployments. The operator triggers a backup for a BOSH deployment (or director) using the cli. The BBR binary then looks at the jobs in the deployment (or director) for lock / backup / unlock scripts. The binary then triggers those scripts in the prescribed order. The backup artifacts are transferred to the jumpbox. The operator proceeds to transfer the artifacts to external storage.

A restore is the inverse of this process - the backup artifact must be copied into the jumpbox where the BBR binary is located. Then the restore process is triggered by the operator using the cli, specifying the deployment or director to restore, as well as the path to the backup artifact. The BBR binary identifies which jobs implement the restore script, copies the matching backup artifact into the job, and triggers the restore script.