WHITE PAPER

Multi-Site Pivotal Cloud Foundry Reference Architecture

A Technical Comparison of PCF Foundation Configurations: Active-Active, Active-Passive, and “Stretched”

By Ryan Pei, Jared Ruckle, Sudhindra Rao, and Luciano Silva

Pivotal

Table of Contents

Pivotal

Introduction

In this paper, we will review three reference architectures with Pivotal Cloud Foundry (PCF). Our goal: to help you achieve the right level of multi-site availability for your applications, data, and the platform itself.

This is a hot topic for enterprises that want to modernize their software delivery practices. A previous whitepaper introduced some of these architectures ("Multi-site Pivotal Cloud Foundry: Deployment Topology Patterns and Practices"). To build on this excellent work, we will take a closer look at three popular configurations:

- **Active-active** is two fully functional PCF platforms deployed and primed with applications to serve traffic in case of failure.
- An **active-passive** configuration, by contrast, features one platform deployment that acts as a "standby" site.
- A **"stretched" deployment** has a single PCF deployment that spans across two data centers.

All three configurations are used in production by enterprises like yours. And all three are perfectly wonderful choices, depending on your requirements. This paper attempts to help you select the most suitable option, according to your uptime and availability needs.

We'll study each of these deployment topologies—and their characteristics—in detail.

These reference architectures can be applied to PCF deployments that span multiple enterprise data centers, multiple public cloud sites, or some combination of the two. The choice is yours!

Rethink How You Achieve High Availability

At this early stage, you may be leaning towards an "active-active" configuration for PCF. And what's not to like? This option features two foundations, each in its own datacenter. This has to be the best option to achieve the highest level of availability possible, right? Not necessarily.

To be fair, it's easy to default to the "active-active" option. After all, traditional IT solutions typically encourage "two of a kind" models as the standard way to achieve high availability. At Pivotal, we ask all our customers if simply meeting this definition of high availability is what they truly want to accomplish, regardless of cost and complexity. There is an alternative way to select your reference architecture!

Instead, we recommend using metrics and data wherever possible. (The paper will address how to go about this.) It's best to track actual vs. objective uptimes (which we refer to as SLIs and SLOs, or Service Level Indicators and Service Level Objectives). You should also look at familiar metrics like RTO (Recovery Time Objective) to measure how quickly you can recover, as well as RPO (Recovery Point Objective) benchmarks to assess how quickly you can restore the system's overall state.

We'll also talk about error budgets. Error budgets help you estimate, and subsequently track, your platform's reliability metrics. Each architecture comes with an example error budget spreadsheet (in the Appendix). Use these tools for your own deployment.

In the world of distributed systems, you can't assume that redundancy alone will help you meet your uptime goals. These metrics will modernize how you think about operations.

"Do I have to have this all figured out now?"

At this point, you might be thinking: "This is a lot of information to take in. Am I really supposed to make the right choice for my business right now? Thankfully, the answer is no.?" But you do need to think about how to optimize performance based on what you learn so you can make better decisions over time as you gain experience.

Once you're up and running with the platform, you'll learn more about your actual availability requirements. From there, you can measure how the platform performs. And you can better fine-tune the design of the platform. This way, you can meet uptime expectations while also optimizing for IT costs. Your initial choice isn't your final decision (we'll explain later). But it is important to consider your requirements under the strengths and weaknesses of each option.

Deployment Topologies

How do we compare the three multi-site PCF platform architectures? All of them are widely used by enterprises in production. Pivotal has performed some testing and validation on them as well. This experience tells us to evaluate with these key factors in mind:

Application Factors

These affect your users and your application developers. What is the end user experience for your apps? Can users access it? What is the latency like? For your developers, can they successfully perform a cf push operation? And, what is the per-app cost for running on the platform?

Platform Factors

What is the operational experience with this architecture? How are backups and patching affected? What about major upgrades? How is failover handled? What is the ongoing operational cost?

Data Factors

What kind replication do my of apps need? Does this architecture support that?

These factors are a useful starting point. Your actual experience with the platform will depend on many other elements, like your IaaS resources and your network.

It is therefore important to actually track and measure uptime and other SLIs for the platform once deployed. This way, you can change the architecture later if you find that other components affect the platform.

One quick term to know before we dig in: “foundations.” A PCF foundation refers to a single Ops Manager deployment and its associated tiles.

Active-Passive Deployment: The Right Choice for Most Apps

The **active-passive** architecture has two PCF foundations deployed in separate data centers. One foundation is "active," serving as the primary foundation for app traffic and platform services. The other is "passive," and is only put into active use if the other foundation is unable to serve requests for a given amount of time.

This architecture is a good choice—especially if you can implement a sufficiently fast-failover process. This way, you can minimize the costs and complications of day-to-day operation. **In fact, we recommend active-passive as the default option for most enterprises' first production environment that requires high availability.** Let's explore why.

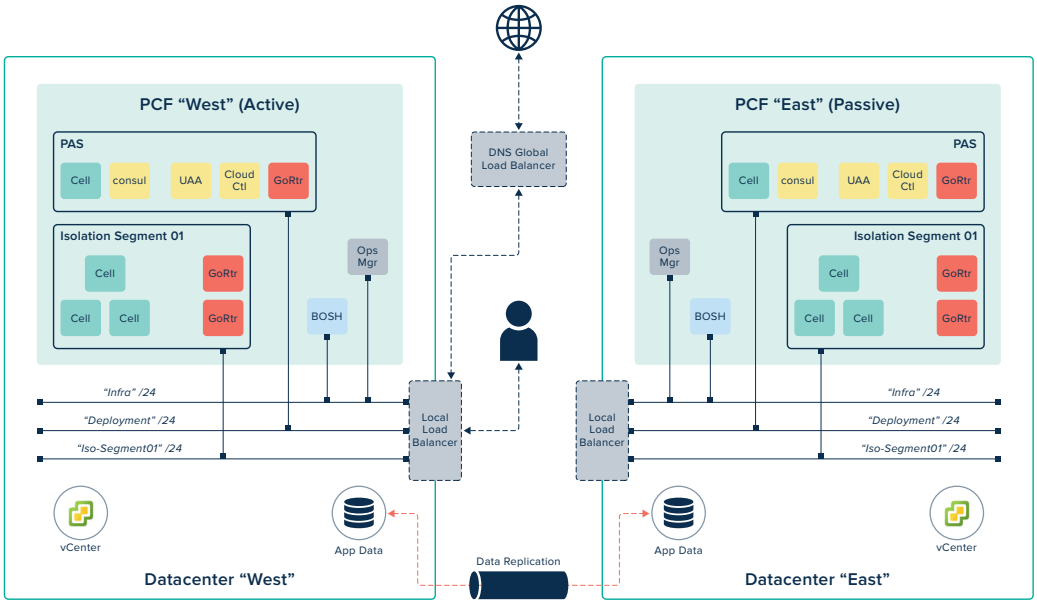

Figure 1: A common active-passive reference architecture.

In Figure 1, the focus of this architecture is a single PCF foundation that is permanently "active." A second foundation, the "passive" one, can be brought online as needed. Another option to mull over: make both foundations active, but have the secondary site operate at a much smaller scale. You can always scale up in times of distress or increased traffic.

In fact, a passive foundation can be made "cooler" or "warmer," to suit your preferred balance of cost and uptime. A "cooler" passive foundation (i.e. a PCF foundation deployed at bare-minimum scale) will be more cost-effective, but will take longer to bring online when needed than a "warmer" passive foundation. The latter will cost a bit more, but will initialize faster. The warmer the passive foundation, the closer it starts to resemble an active-active architecture.

We've seen many customers deploy an active-passive configuration to start. From there, they learn and adjust the "temperature" of the passive foundation. Read more about this process in the Summary.

Application Factors for Active-Passive

- **Expected Application Uptime:** Greater than 99%. Your actual app uptime will, of course, vary depending on several factors like:
  - Storage/power/network outages of the active datacenter.
  - The availability of any backing services used by your apps.
  - Configuration changes or updates to PCF with unplanned consequences.
  - Platform RTO/RPO (here "platform" refers to both PCF and the underlying infrastructure.)
- **cf push uptime ranking:** Greater than 99%. Application developers should be able to deploy apps and expect cf cli availability to manage the process, with occasional interruptions in service. (Most of these can be planned ahead of time, like PCF upgrades).
- **Cost of Compute:** Approaches half what it would be for the active-active architecture. But it can cost more than a single foundation (depends how "warm" or "cool" the passive foundation is.)
- Your application versions will need to be in sync across PCF foundations. (This is true of the active-active configuration as well.)

Platform Factors for Active-Passive

- **Patching and Upgrades.** The ideal PCF upgrade procedure: schedule planned CF API downtime windows to coincide with major/minor upgrades. Be mindful of patch upgrades, as well. These patches can sometimes include configuration changes that are necessary for a successful CVE update.
- **Backups.** Ideal PCF backup procedure is to schedule planned CF API downtime windows to coincide with scheduled backups. Use BOSH Backup & Restore for this scenario.
- **Failover.** This depends on the passive foundations' warm/cool state. If the active foundation goes down, and the passive PCF foundation has capacity, you can deploy more apps there. Start with more critical workloads first. Otherwise, you must deploy or scale up the passive foundation before you can host these apps.
- **Overall Operational Cost:** With active-passive, you will be effectively managing two foundations instead of one, until you have attained a sufficient level of platform automation. (We recommend automating as much as possible, no matter which architecture you choose!) Two sites will need to be kept in sync to ensure reliable operation.

Data Services Factors for Active-Passive

This architecture will require data replication between the two foundations. Your ability to replicate data will depend on the type of data store you're using and the supported replication capabilities of that service. The following PCF data services can be setup to replicate data across foundations:

- **Pivotal Cloud Cache.** Uses WAN replication across sites.
- **RabbitMQ.** You can setup federation or shovel-based replication across sites.
- **MySQL V2.** Cross platform replication is expected to become available for this relational database sometime in 2019.
- **Partner Services.** Several third-party data services also work for cross-site replication use-cases, such as Minio and Redis Labs.

You can also use non-Pivotal data services to handle replication.

Note: Your tolerance for latency will depend on the app. As such, all the recommended maximum inter-data center latencies are subject to each app's use case.

This means you can potentially use data services that do not support cross-site replication. And you can sidestep replication latency challenges. How? By restoring these data services from backups instead. How often you take these backups will of course affect your RPO.

Prerequisites for Active-Passive

- **DNS Global Load Balancer.** This needs to be configured to point to the two PCF foundations, each with their own local load balancer. Global DNS is configured to point to the two local load balancers' VIPs.
- **Replication.** Platform and app data replication across two foundations, as mentioned above.
- **Identical Release Versions.** PCF foundations across data centers should have the same PCF release versions, configurations and capacity, to the extent possible.

Active-Active Deployment: When Every Minute Counts

Active-active architectures feature two PCF foundations deployed in separate data centers. Each PCF foundation is configured and scaled identically. Furthermore, each foundation hosts at least one copy of a given workload. This architecture allows for the highest availability of your apps, and for PCF itself. It is also the most costly and complicated architecture of the three. Why? Cost comes from additional infrastructure consumption and software licensing. Additional complexity stems from a higher burden on your operational teams.

It's important to note that your application versions and data will need to be in sync across PCF foundations. Make sure your CI/CD tools and processes support this option. (We recommend Concourse.)

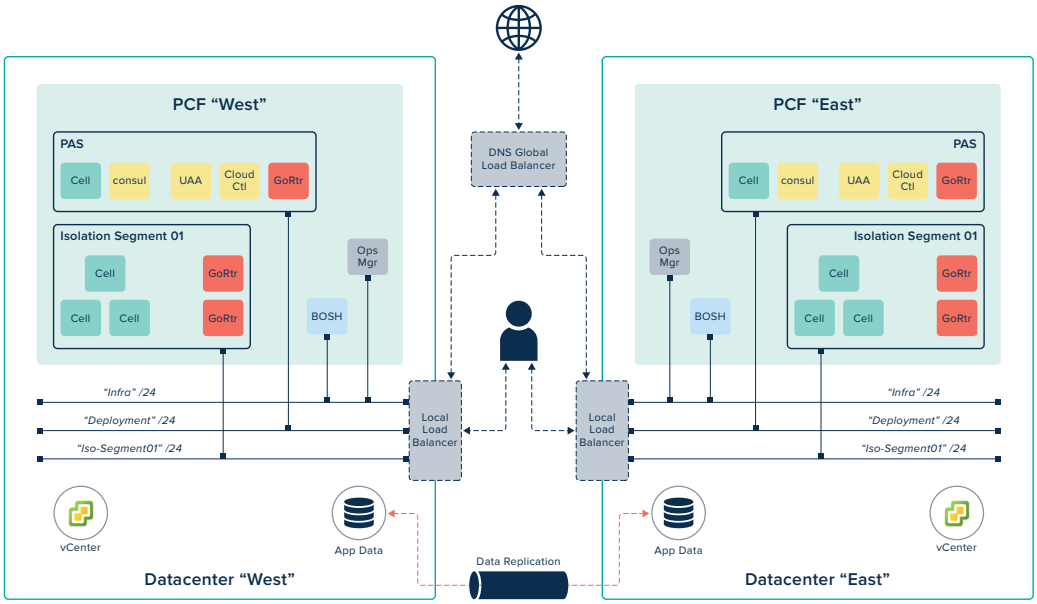

Figure 2: An example active-active reference architecture.

Application Factors for Active-Active

- **Expected Application Uptime:** Greater than 99.9%. Your actual app uptime will vary depending on several factors like:
  - Any disruption that affects **both** foundations, such as a misconfiguration of PCF.
  - Or if, for instance, an app workload is thought to be deployed on both foundations, but is actually mistakenly only on one.
  - The availability of any backing services used by your apps.
- **Expected cf push Uptime:** Greater than 99.9%. Once again, this will vary based on incidents or changes that happen to affect **both** foundations.
- **Cost of Compute:** Approaches **double** what it would be, on a per app basis, when compared to the active-passive architecture, or a single PCF foundation.

Platform Factors for Active-Active

- **Backups.** The foundations are not completely identical for active-active, so you will need to backup each foundation separately. It's recommended to backup each foundation at different times, to eliminate any downtime impact (one foundation will be able to accept requests while the other is being backed up). We recommend BOSH Backup & Restore for this purpose.
- **Patching and Upgrades.** PCF upgrades should also not impact uptime, if performed in a blue/green manner.
- **Failover.** If a data center or PCF foundation goes down, then the DNS Global Load Balancer can be configured to route requests to the secondary site's local load balancer. This process involves pointing to DNS entries for routing to the second site.
- **Overall Operational Cost:** Like with the active-passive architecture, with active-active you will be effectively managing two foundations instead of one, until you have attained a sufficient level of platform automation. (Again, we recommend automating as much as possible, no matter which architecture you choose!) Two sites will need to be kept in sync as much as possible, as we discuss in the next section.

Data Factors for Active-Active

For the most part, the same guidance applies to active-active as for active-passive; please refer to the active-passive section above.

The major difference? Foundations in the active-active architecture need to be kept in sync with a greater real time requirement than the active-passive foundations. Remember, the passive foundation is not always serving users. You can afford to be "out-of-sync" for a longer duration with active-passive. But when both foundations are active and serving traffic, there's a greater chance of out-of-sync data affecting users.

- **Data replication is most complicated for active-active.** Here's why:
  - **You have to sync platform data in real time.** This is all the metadata for application pushes, users, and permissions. You can either replicate this in the backend, using cache/database replication. Or you can do it in the frontend, using the CLI and plugins that replicate user interactions.
  - **You have to sync application user data in real time.** It's usually best to use replication at the caching layer. Products like Pivotal Cloud Cache are purpose-built for this function. Otherwise, replicate at the database layer. This way, you don't have to rely on application developers to deploy their applications with front-end replication built-in.
  - The CAP Theorem limits how consistent and available your data will be across sites at any given time. **Data may not be 100% consistent across the two sites, at any given time.** However, it is possible to deliver an excellent user experience.

Prerequisites for Active-Active

Prerequisites for active-active are similar to that of active-passive; please refer to the “Prerequisites” in the active-passive section.

The one difference: the level of real-time consistency you need to maintain between the two foundations. With active-active, you can’t afford as much “drift” in the data, software versions, and update differences between the two foundations as you could otherwise tolerate with active-passive.

Given the cost and complexity, some customers opt to use active-active configurations for their most critical workloads. For example, payment processing and customer-facing services could be deployed as active-active across both PCF foundations. Meanwhile, non-critical apps could run in only one foundation. Each app owner can decide what level of uptime their app requires, and if the cost trade-off is justified in their scenario.

Refer to the Summary for the scenarios where Active-Active is the recommended option.

Stretched PCF Deployment: Ideal for Smaller Ops Teams with A Single Foundation

A stretched architecture is a single PCF deployment where the underlying infrastructure—such as VMware vSphere clusters—are configured across data center sites. (Hence the term "stretched.") Other infrastructure resources (network, storage) are shared between each site.

When considering this option, you need to study latency. Will the distance between the two sites cause issues? After all, data is flowing inside the platform constantly. You should know that certain PCF components rely on quorum clustering (the MySQL backend for the Pivotal Application Service). In the stretched configuration, this data store may be more likely to suffer a more-than-quorum loss of its nodes if an entire data center fails. (One fix: if available, use a third data center to include in this stretched deployment.)

Does this approach sound familiar? It should—it's how public cloud providers offer multiple availability zones (AZs)s within a geographic region. An example deployment is shown below in Figure 3.

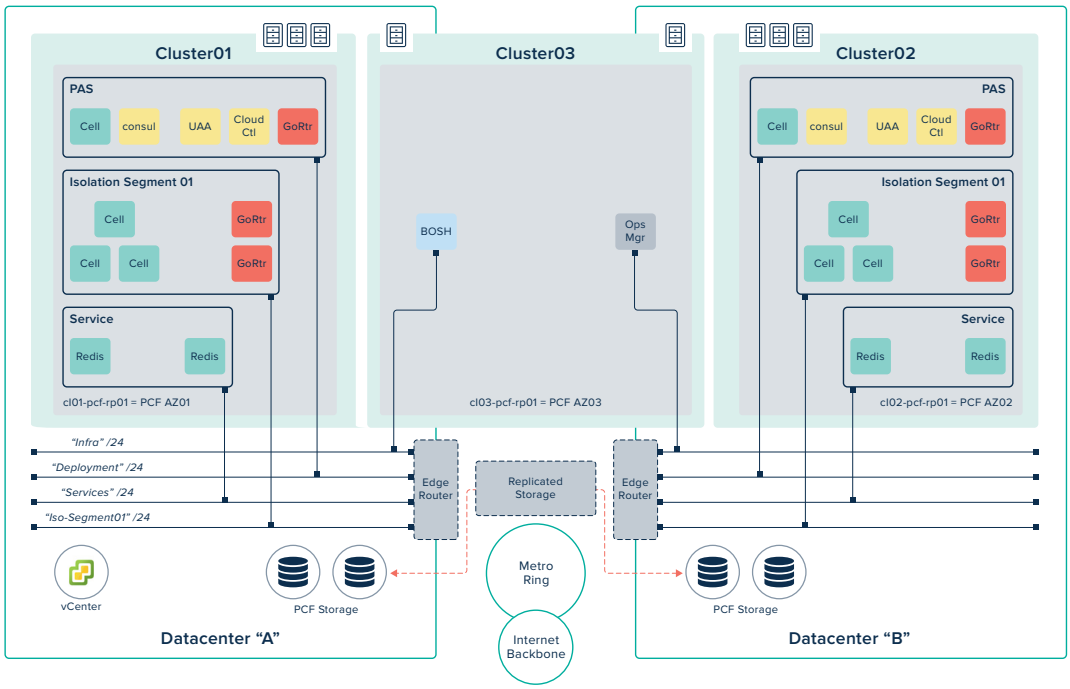

Figure 3: An example "stretched" reference architecture..

In the stretched deployment, each data center hosts one cluster. The PCF foundation has one AZ mapping to each cluster. This ensures that, in the event of a data center outage, at least one AZ remains.

The stretched cluster (Cluster03 in Figure 3) is optional. The advantage this provides: the VM resurrection capabilities of vSphere can automatically resurrect the BOSH Director or Ops Manager in the event of a data center failure. How? By moving these jobs to the other data center. The stretched cluster can host VMs on the "healthy" site should the other site fail. This has not been as extensively validated though, which is why we call this configuration optional for now.

Application Factors for Stretched

- **Expected Application Uptime:** Greater than 99%. Once again, your actual app uptime will vary, depending on several factors like:
  - **Shared** storage/power/network availability for **both** data centers.
  - The availability of any backing services used by your apps.
  - Configuration changes or updates to PCF with unplanned consequences.
  - PCF-specific RTO/RPO—because there is just one PCF foundation in this architecture.
- **cf push uptime ranking**: Greater than 99%. Application developers should be able to deploy apps and expect cf cli availability to manage the process, with occasional interruptions in service, most of which can be planned for (e.g. PCF upgrades).
- **Cost of Compute:** The stretched option is unlikely to be as costly as in an active-active scenario. But it still approaches **double** what a single foundation on a single data center would cost. Remember, you are presumably replicating live workloads across two data centers.

Platform Factors for Stretched

- **Patching and Upgrades.** The ideal PCF upgrade procedure: schedule planned CF API downtime windows to coincide with major/minor upgrades. Be mindful of patch upgrades, too, since these patches can sometimes include configuration changes which are necessary for a critical CVE patch.
- **Backups.** Ideal PCF backup procedure: schedule planned CF API downtime windows to coincide with scheduled backups. Use BOSH Backup & Restore!
- **Failover.** As there is only one foundation, there is no foundation-level failover. In this case, PCF will handle data center failover automatically. If one datacenter goes down, then each PCF foundation with an AZ in that datacenter would lose that AZ. Diego will automatically resurrect app instances from the failed AZ to Diego Cells on the other AZs, wherever capacity is available. Note that BOSH, the deployment tool for PCF, does NOT resurrect those VMs on the remaining data center(s). It is recommended to have at least one AZ's worth of Diego Cell capacity available across the other AZs to be able to absorb the loss of capacity from a failed AZ.
  - Another circumstance worth noting is if a datacenter with half or more of the nodes of a quorum clustering component of PCF, such as the MySQL backend of PAS, goes down, then you will lose quorum and the platform data may need to be restored from a backup to prevent data corruption. To avoid this, you may also consider designating a "preferred site" in vSphere.

Data Factors for Stretched

There will be no data replication required between the data centers. That's because this architecture is based on a single PCF foundation and uses the the two sites' inter-connectivity to share data.

Prerequisites

- A stretched architecture is possible with all supported versions of PCF.
- If you are using PCF 2.1 or earlier, you will need a stretched vCenter. That is, you will need a single vCenter stretched across two sites.
- If you are using PCF 2.2 or higher, you can enable this architecture with a multi-vCenter configuration for PCF. This feature exists for OpenStack in Ops Manager 2.3; future support for other IaaS targets is planned as well.
- Load balancer with local load balancer capability, and access to both DCs.
- Shared/replicated storage between data centers.

Best Practices & Recommendations

Now you have a solid understanding of each option and their respective “pros” and “cons.” Let’s finish up the paper with some recommendations and best practices.

Application Uptime

When you need greater than 99.9% availability...these critical apps should be spread across two PCF foundations, deployed in active-active or active-passive architectures.

When you need greater than 99% availability...having only one foundation hosting these apps may be enough. If you still desire a multi-datacenter configuration (or if you have a data center with a history of unpredictable performance), you can use a stretched PCF architecture across multiple data centers.

Availability of Data

Some PCF data services support cross-site replication, but not all. There are non-Pivotal options as well.

Cost and Complexity

Maintaining two PCF foundations for **active-active** comes with the highest cost and complexity. This can be the best option for a subset of your applications, however.

When implementing **active-active** architectures, you must automate platform changes across two PCF foundations to keep them in sync. Automation is highly desirable in all three settings.

Latency

For **stretched architectures**, data center proximity affects network latency for PCF’s inter-component traffic. This is a critical factor in determining the suitability of this option.

If two data centers are close to each other—and if network latency is low—then a single, stretched PCF foundation stretched across two sites is ideal. What kind of latency is “too high”? This number will largely depend on your traffic throughput, especially with respect to logging and data replication.

If two data centers are further apart—and you have high latency—then a multi-foundation deployment is your best option. (Latency numbers are not benchmarked and are out of scope for this paper.)

Platform Availability for App Developers

For **active-active** and **active-passive** architectures, the app developer must cf-push and manage their apps on both sites. This gives the application development team the freedom to use the right configuration for the scenario. Use active-active when you need to, and use active-passive for the rest of your portfolio. Automating your chosen workflow is encouraged!

Platform Operations for Platform Teams

For **active-active** and **active-passive** architectures, your platform team has to manage two different PCF foundations. Make sure each site is as consistent as possible. Currently, there is automation for PCF software upgrades and updates (via PCF Concourse Pipelines). You should also automate the replication of important metadata (users, orgs, spaces, quotas, security groups, etc. A tool called cf-mgmt can help.)

For **active-active** and **active-passive** architectures, the DNS Global Load Balancer will need routing rulesets that make sense for your business. Session stickiness, closest proximity to the user, and throughput load balancing are all factors to consider.

For **active-active** and **active-passive** architectures, health checks are complicated by having two foundations. Gorouters (which direct app traffic on a per-foundation basis) have no way of indicating the health of apps on the other foundation. Ensure that your DNS Global Load Balancer has logical health-checks that can reason about platform vs. app-instance failures.

For PCF upgrades, we recommend a "rolling upgrade" for **single foundations** (e.g. dev -> test -> stage -> prod) and blue-green upgrades for multi-foundation (e.g. upgrade PCF-east first, then PCF-west.)

Security

Deploying the IPsec Add-on can slow performance for multi-datacenter PCF deployments. This is particularly true for **stretched PCF deployments**, where there is a high dependency on network latency. Transport layer security (TLS) is used throughout the platform, mitigating this concern. Check with your InfoSec team for the precise nature of your encryption-in-transit requirements.

For **active-active** and **active-passive** architectures, configure both foundations to utilize a single user repository for user authentication and authorization. Further, user data needs to be synced between the two sites to ensure proper user access.

For **active-active** and **active-passive** architectures, you will need to configure all security add-ons, like ClamAV, for multiple foundations.

For certificate rotation on **active-active** and **active-passive** architectures, ensure that certificates are properly rotated on both foundations.

I've learned and experimented. How do I adjust my architecture based on my experience?

As mentioned earlier in the paper, you aren't confined by your initial decision. PCF customers use the guiding principle of "treat your platform as a product". Just as your business wants to constantly improve its products and services, you should iteratively improve the platform. You want PCF to continuously improves the value it delivers your internal customers: app developers. That's why we embrace SLIs and SLOs. These are metrics that help you determine if change is needed. For instance, if you discover you are "overachieving" your availability requirements, you can make plans to reduce your platform's redundancy characteristics. If you are underachieving, you can increase redundancy.

How does this change happen? It's more straightforward than you might think!

Changing from Active-Active

You may want to go from an **active-active** to an **active-passive** architecture if you find you are not using the failover foundation as much as you anticipated. You can simply remove PCF from this second site, and reallocate that capacity to other needs. Or, if you don't want to go that far, you can simply scale down the foundation.

Other considerations for switching away from an active-active architecture:

- If you observe that you don't need a fully active secondary foundation, then you can make it passive by simply switching off real-time replication capabilities where desired.
- Do you need **even higher availability**? You can deploy a third active foundation (active-active-active). Realize of course that this will be considerably more expensive to maintain. Consider other options first. For instance, if it appears that downtime incidents are often due to underlying hardware issues, consider either upgrading the hardware, or adding more capacity for your existing foundations.

Changing from Active-Passive

You may want to go from **active-passive** to **active-active** if you find that the passive foundation is needed more often than you expected, or your RTO is not being met for key workloads. You can make the passive foundation "warmer" (more active) by scaling it up closer to the size of the active foundation, and even enable automatic failover with the DNS Global Load Balancer and automatic, real-time replication of app data. This change effectively results in an **active-active** architecture. But, don't forget to consider the added cost and technical requirements to support active-active, as discussed previously.

Changing a Stretched Architecture

To change a **stretched** architecture, you can change the availability zone (as defined in Ops Manager) that maps to the vSphere cluster on the data center you no longer want to use. Instead, map it to another cluster on the same data center as the other AZ(s).

If you want to convert the **stretched** architecture to an **active-passive** or **active-active** architecture, then you can remap the AZs as mentioned above. This action changes a foundation from "stretched" to "unstretched." From there, you can proceed to deploy another foundation in a second data center. You will then need to replicate all data and apps from the existing foundation to the new one. Finally, you can setup your networking according to the requirements of the active-passive/active-active architectures.

Summary

Now you know about the most common PCF multi-site deployment topologies: active-active, active-passive, and stretched architectures. We've described their respective technical characteristics, benefits, cost, and complexity.

We trust this material will serve as a handy reference guide for you during the design of a PCF multi-site deployment.

Of course, the design is just one iteration in your platform-as-a-product design. Also remember that platform teams must be constantly optimize the platform by refining SLA requirements, SLI/SLOs, and continuously learning from experiences and outcomes.

Appendix

Recommended Reading

- **Active-Active Spreadsheet.** A more quantifiable assessment of uptime can be estimated using a Customer Reliability Engineering worksheet for the Active-Active PCF architecture.
  - This particular worksheet estimates the uptime for an application running on an active-active PCF deployment on two "typical" enterprise datacenters.
  - This tool assesses the platform's ability to meet target uptime requirements (e.g. 99.9% availability) based on the estimated downtime each year due to risk (e.g. datacenter power failure)
- **Active-Passive Spreadsheet.** A more quantifiable assessment of uptime can be estimated using a Customer Reliability Engineering worksheet for the Active-Passive PCF architecture.
  - Note that this worksheet calculates the expected uptime for a non-critical application (i.e. not live on both sites constantly) for an active-passive PCF deployment on two "typical" enterprise data centers.
  - This tool assesses the platform's ability to meet target uptime requirements based on the estimated downtime each year due to risk (e.g. data center power failure)
- Blog Post: No, You Can't Cheat the CAP Theorem. But Pivotal Cloud Cache on PCF Comes Close. Here's How.
- Blog Post: Announcing Pivotal Cloud Cache V1.3
- Whitepaper: A Developer Primer for Pivotal Application Service: Get Ready to "cf push"
- Blog Post: SLIs and Error Budgets: What These Terms Mean and How They Apply to Your Platform Monitoring Strategy
- Blog Post: Thinking in Error Budgets: How Pivotal's Cloud Ops Team Used Service Level Objectives and Other Modern SRE Practices to Improve Outcomes

Disaster Recovery for PCF

The recovery time objective (RTO) to restore a single PCF foundation is the sum of the time required to run these four steps:

1. Recreate PCF platform (via a BOSH deployment).
2. Restore its data. This includes re-populating the foundation with apps, orgs, spaces, and quotas.
3. Re-run the tile errands for any on-demand service tiles' errands (if you have these tiles in-use).
4. Restart the apps that use the service bindings in step 3.

The recovery point objective (RPO) for restoring a single PCF foundation is the time of the last BBR backup snapshot. The best practice here is to regularly run a scheduled backup of PCF with BBR backup pipelines in Concourse. Manual processes are not recommended!

For the complete set of instructions on how you to restore a PCF foundation, refer to this documentation.

SLIs/SLOs and Error Budgets

Customers often ask us these questions:

- Why do I need to understand the concept of Service Level Indicators (SLIs), Service Level Objectives (SLOs), and error budgets?
- And how does this differ from Recovery Time Objective and Recovery Point Objective?

SLIs, SLOs, and error budgets are great tools for tracking whether your platform actually meets your business' continuity-of-service requirements. When your CIO mandates that your services cannot be down for more than X amount of time each year, it is important that you actually be able to measure this metric!

SLIs and SLOs are measures of availability over a period of time, such as the uptime of a particular service or app over a month or a year. RTO and RPO are measures for a single incident that affects the availability of your service/app.

For example, let's say your RTO for a network failure is two hours. So, when the network fails, it will take two hours to restore the network and restore your service for which the RTO applies. If this type of incident happens three times a year, then you can say that your SLO is impacted by this incident at least by six hours each year (two hours per incident, three times a year). So with this incident alone, you are below the 99.99% SLO mark.

Here's an example of some key SLIs that you can track with PCF Healthwatch to measure the effectiveness of your platform in achieving SLOs that are typically important to any IT enterprise:

SLI	SLI Description	Probe
App Runtime	An existing Canary App with multiple instances responds to a HTTP GET request with HttpStatus== 200 and the expected body text within 10 seconds.	Healthwatch – CanaryApp up/down
Control Plane	A user is able to cf push a new Spring application to the platform and have that application respond to a HTTP GET request within 2 minutes	Healthwatch – CLI Command Health