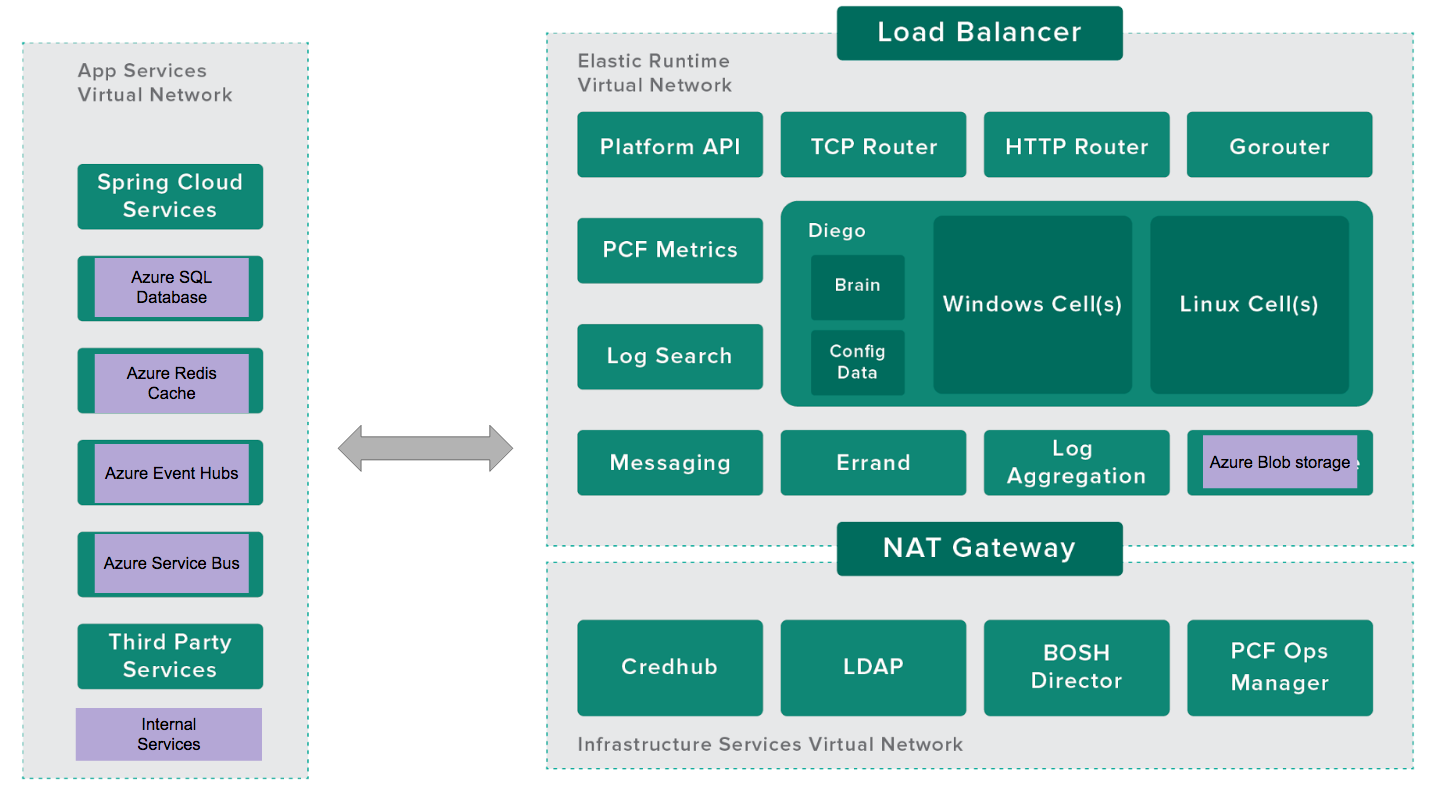
## What’s Included

### Pivotal Cloud Foundry: Platform & Services

Pivotal Cloud Foundry (PCF) is a cloud-native platform for deploying and operating applications. PCF can run on-premises, and atop public cloud providers like Microsoft Azure. This gives enterprises a hybrid and multi-cloud platform.  
  
PCF is a uniform way for you to launch, and quickly iterate, on applications in any popular language. The platform manages many implementation details for you. With PCF, you no longer have to think about how to deploy, scale, and expose an app. You can instead focus on adding business value with custom code. PCF enables developers to speed up application development and reduce time to market.

The diagram in Figure 1 highlights important PCF components. This architecture organizes elements according to their network affinity.

* **Infrastructure Services Virtual Network**, used by Pivotal Ops Manager to control the underlying infrastructure
* **Elastic Runtime Virtual Network**, used by PCF’s Elastic Runtime, container orchestrator, and other related services
* **App Services Virtual Network**, used by PCF managed services like MySQL, RabbitMQ and Spring Cloud® Services for Pivotal Cloud Foundry

*Figure 1 - An overview of selected Pivotal Cloud Foundry components.*

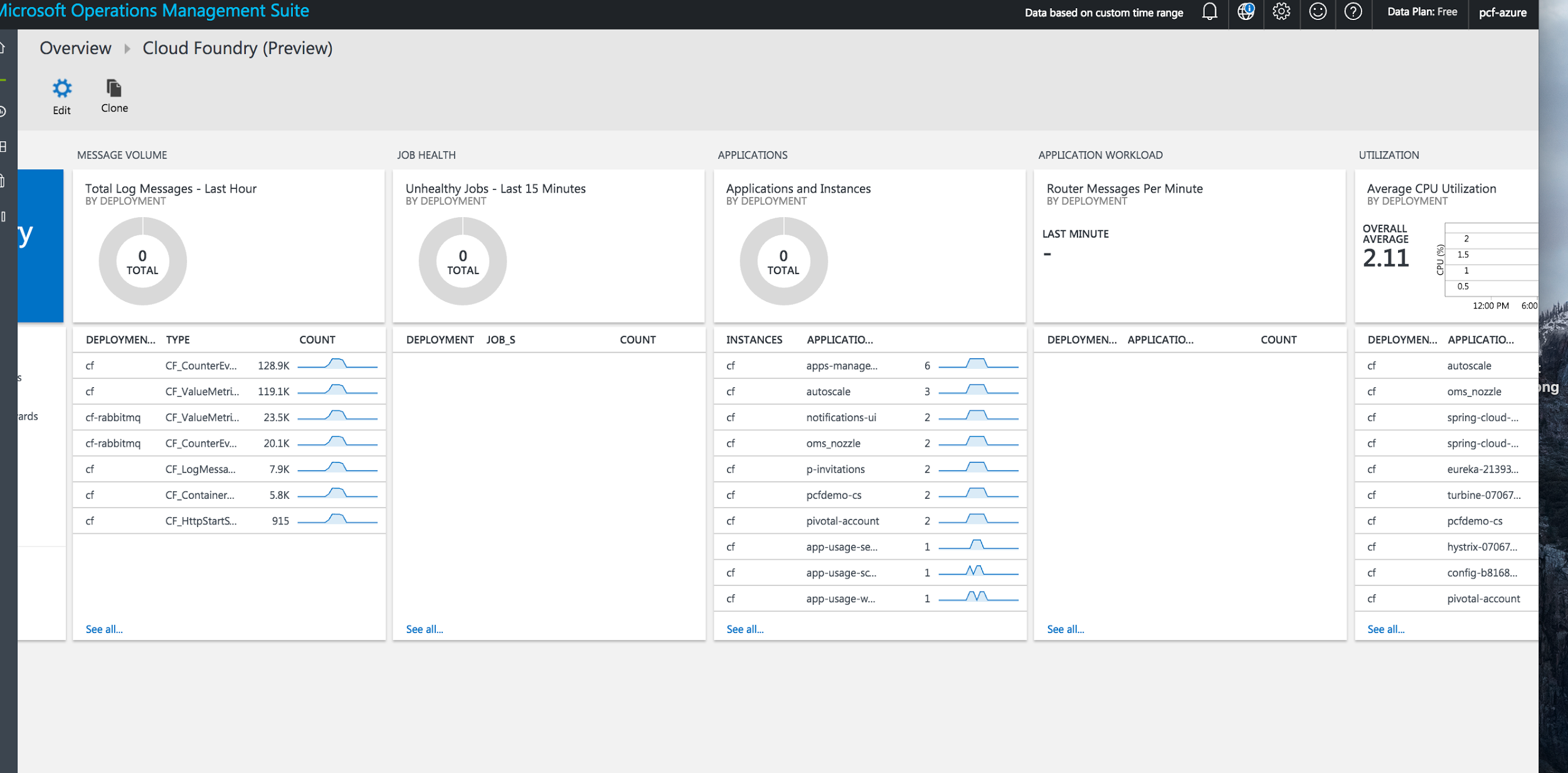
Let’s review each item in Figure 1.

#### Infrastructure Services Virtual Network

* **CredHub.** CredHub is a forthcoming extension to Cloud Foundry. This feature centralizes and secures credential generation, storage, lifecycle management, and access. CredHub can mitigate the risk of leaked credentials, a common culprit in data breaches.
* **LDAP.** Lightweight Directory Access Protocol is a protocol used for single sign-on methods. It connects an identity management tool (such as [Azure Active Directory](https://azure.microsoft.com/en-us/services/active-directory/)) and 3rd party systems. Pivotal Cloud Foundry supports LDAP and SSO via its [UAA service](http://docs.pivotal.io/pivotalcf/1-9/concepts/architecture/uaa.html).
* **PCF Ops Manager.** Operations Manager (“[Ops Manager](http://docs.pivotal.io/pivotalcf/1-9/customizing/index.html)”) is a web application used to deploy and manage PCF. Ops Manager communicates with the BOSH Director to deploy Elastic Runtime components and other services.
* **BOSH Director.** [BOSH Director](http://docs.pivotal.io/pivotalcf/1-9/customizing/vsphere-config.html) performs a highly automated PCF deployment based on user-provided configuration details.
* **NAT Gateway.** A network address translation (NAT) gateway enables services in a private subnet to connect to other PCF components. The NAT Gateway also prevents components *outside* the Infrastructure Services VLAN from initiating a connection with these services. [Read more about configuring firewalls for PCF](http://docs.pivotal.io/pivotalcf/1-9/customizing/config_firewall.html).

#### Elastic RunTime Virtual Network

* **Load Balancer.** Production PCF environments use a highly-available [load balancer.](http://docs.pivotal.io/pivotalcf/1-9/customizing/custom-load-balancer.html) The load balancer route traffic to PCF Router IPs and supports SSL termination with wildcard DNS location. It adds appropriate x-forwarded-for and x-forwarded-proto HTTP headers to incoming requests. WebSockets can be supported as needed. Global deployments should use the [Azure Traffic Manager](https://azure.microsoft.com/en-us/services/traffic-manager/), routing to [Azure Load Balancers](https://azure.microsoft.com/en-us/services/load-balancer/) in different regions.
* **TCP Router.** The [TCP Router](http://docs.pivotal.io/pivotalcf/1-9/adminguide/enabling-tcp-routing.html) works with applications that serve requests on non-HTTP TCP protocols. TCP Routing terminates the TLS as close to your apps as possible. In this scenario, packets are not decrypted before reaching the application level. This configuration helps compliance with certain regulations.
* **HTTP Router.** The [HTTP Router](http://docs.pivotal.io/pivotalcf/1-9/concepts/http-routing.html) manages HTTP traffic into and out of the PCF deployment.
* **Gorouter.** The [Gorouter](http://docs.pivotal.io/pivotalcf/1-9/concepts/architecture/router.html) routes traffic coming into PCF to the appropriate component. For example, the Gorouter is used when an operator addresses the Platform API. It is also used when application user accesses an app running on a Diego Cell.
* **Platform API.** The Platform API (or [Cloud Controller](http://docs.pivotal.io/pivotalcf/1-9/concepts/architecture/cloud-controller.html)) provides REST API endpoints for clients to access the system. It maintains a database with tables for orgs, spaces, services, user roles, and more.
* **PCF Metrics.** [PCF Metrics](http://docs.pivotal.io/pcf-metrics/1-2/index.html) stores logs, metrics data, and event data from applications running on PCF. The module renders this data visually. This treatment helps operators and developers better understand the application health and performance.
* **Log Search.** Operators can use [PCF Log Search](https://docs.pivotal.io/logsearch/index.html) to analyze logs from different system components. Some customers prefer Azure's native capabilities in this area. Options include the [Microsoft Operations Management Suite](https://docs.microsoft.com/en-us/azure/operations-management-suite/operations-management-suite-overview) (OMS) and [Monitor](https://azure.microsoft.com/en-us/blog/announcing-the-public-preview-of-azure-monitor/) services. Microsoft has adapted Cloud Foundry's Firehose and nozzle features, shown below. Users can analyze logs and metrics using this starter OMS visualization widget.



*Figure X - Cloud Foundry metrics and logging, shown in the Microsoft OMS.*

* **Diego.** [Diego is the container orchestrator](http://docs.pivotal.io/pivotalcf/1-9/concepts/diego/diego-architecture.html) for Pivotal Cloud Foundry.
  + **Brain.** The [Diego Brain](http://docs.pivotal.io/pivotalcf/1-9/concepts/diego/diego-architecture.html#brain-components) distributes one-off tasks and long-running processes (LRPs) to Diego Cells. These processes are dynamically managed, increasing fault-tolerance and long-term consistency.
  + **Linux Cell.** Each Linux-based application VM has a [Linux Cell](http://docs.pivotal.io/pivotalcf/1-9/concepts/diego/diego-architecture.html) that executes local application start and stop actions. The Cell manages the containers within a VM, and reports app status and other data to the Log Aggregator.
  + **Windows Cell.** [Windows Cells](http://docs.pivotal.io/pivotalcf/1-9/windows/index.html) perform the same functions as Linux Cells, but for tasks and processes that require Windows and .NET components.
  + **Config Data.** [This database](http://docs.pivotal.io/pivotalcf/1-9/concepts/diego/diego-architecture.html#database-vms) maintains a real-time representation of the state of the Diego cluster. Data stored includes metadata about all desired LRPs, running LRP instances, and in-flight tasks. This also includes a consistent key-value data store to Diego.
* **Messaging.** [Messaging](https://docs.pivotal.io/pivotalcf/1-9/concepts/architecture/messaging-nats.html) in PCF is done via NATS, a lightweight publish-subscribe and distributed queueing messaging system.
* **Errand.** These instances are dedicated to [running one-off errands](https://docs.pivotal.io/tiledev/tile-errands.html) (for example, adding and removing Tiles).
* **Log Aggregator.** The Log Aggregator (or “[Loggregator](https://docs.pivotal.io/pivotalcf/1-9/loggregator/architecture.html)”) aggregates and streams logs and metrics. It collects and processes data from all user apps and Elastic Runtime components.
* **Azure Blob Storage.** PCF uses [Blob Storage](http://docs.pivotal.io/pivotalcf/1-9/concepts/high-availability.html#blobstore) to host buildpacks, droplets, packages and resources. This can be an object storage service or internal file system. Azure Blob storage is recommended for PCF deployments on Azure. [This is supported by default in the recommended BOSH manifest files](https://azure.microsoft.com/en-us/blog/cloud-foundry-integrating-with-azure-blob-storage-and-managed-disks/).

#### Application Services Virtual Network

Customers use Pivotal Cloud Foundry's core components to deploy and operate their apps. Developers and operators tend to extend their modern apps with other services.  
  
These add-ons connect via an Application Services Virtual Network. Here are a few popular services that Azure customers often integrate into their apps.

* **Spring Cloud Services.** [Spring Cloud Services for Pivotal Cloud Foundry](https://docs.google.com/document/d/1HkjX7DyY5szFiuwqmYmCHz-vUbXN3cQoRjFDYP8kDr0/edit#heading=h.rys80p658l9p) (PCF) packages server-side components of Spring Cloud projects, including Spring Cloud Netflix and Spring Cloud Config, and makes them available as services in the PCF Marketplace.
* **Azure SQL Database.** [Azure SQL Database](https://azure.microsoft.com/en-us/services/sql-database/) is a managed cloud database for app developers. The service makes building and maintaining applications easier and more productive. SQL Database includes built-in intelligence that learns app patterns and adapts to maximize performance, reliability, and data protection.
* **Azure Redis Cache.** [Azure Redis Cache](https://azure.microsoft.com/en-us/services/cache/) is based on the popular open-source Redis cache. It gives you access to a secure, dedicated Redis cache, managed by Microsoft and accessible from any application within Azure. It is available in three tiers: basic, standard, or premium.
* **Azure Event Hubs.** [Azure Event Hubs](https://azure.microsoft.com/en-us/services/event-hubs/) is a hyper-scale telemetry ingestion service that collects, transforms, and stores millions of events. As a distributed streaming platform, it offers low latency and configurable time retention enabling you to ingress massive amounts of telemetry into the cloud and read the data from multiple applications using publish-subscribe semantics.
* **Azure Service Bus.** [Azure Service Bus](https://azure.microsoft.com/en-us/services/service-bus/) offers a highly-reliable cloud messaging service between applications and services, even when one or more is offline. Available in every Azure region, this fully-managed service eliminates the burdens of server management and licensing. Asynchronous operations enable flexible, brokered messaging between client and server, along with structured first-in-first-out (FIFO) messaging and publish/subscribe capabilities—perfect for tasks such as order processing.
* **Third-Party Services.** Many different software packages can be integrated with Pivotal Cloud Foundry. Application performance management (APM), API Gateways, and NoSQL databases are some of the most popular categories.
* **Internal Services.** Customers may connect their own internal services using the service  
  broker interface.
* **Application Gateway**. [MICROSOFT CALL TO ACTION: CONTENT MISSING]

#### Other Topics

* **Apps Manager.** [Apps Manager](http://docs.pivotal.io/pivotalcf/1-9/console/index.html), a web-based tool, manages roles and permissions in Pivotal Cloud Foundry.
* **Buildpacks**. [Buildpacks](https://docs.pivotal.io/pivotalcf/1-9/buildpacks/index.html) provide framework and runtime support for your applications. Buildpacks examine user-provided artifacts to determine dependencies. It also manages the configuration needed to communicate with bound services. When you push an application, PCF detects the needed buildpack. It then installs the buildpack bits where the application will be staged.
* **Service Brokers.** Applications depend on services from databases or third-party SaaS providers. A [service broker](https://docs.pivotal.io/pivotalcf/1-9/services/overview.html) arranges for this connection. The service broker provides the service instance, then the app may communicate with that instance.

### How the Platform Supports Compliance & Security

A common security mindset: “going slower reduces risk.” Pivotal and its customers believe that the *opposite* is true. The faster systems change, the harder they are to penetrate. That’s the core idea of cloud-native security, and the “secure by default” features in PCF. These features help companies meet common compliance and security requirements.

* **Authentication.** The [User Account and Authentication](http://docs.pivotal.io/pivotalcf/1-9/concepts/architecture/uaa.html) (UAA) is the identity management service for PCF. It is an OAuth2 provider, issuing tokens for client applications to use when they act on behalf of PCF users. UAA works with the login server to authenticate users with their PCF credentials. It performs [single sign-on (SSO) duties](https://docs.pivotal.io/p-identity/1-9/) using those credentials (or others). UAA has endpoints for managing user accounts, and other functions like registering OAuth2 clients. On Azure, customers tend to use their enterprise Azure Active Directory (AAD). [PCF offers easy integration with AAD for platform access.](https://docs.pivotal.io/pivotalcf/1-9/opsguide/auth-sso.html#configure-pcf-for-saml) In fact, it's the same as on-premise Active Directory or LDAP integration. You need to provide the necessary XML metadata file, and it's done!
* **BOSH.** [BOSH](https://bosh.cloudfoundry.org/docs/about.html) deploys Pivotal Cloud Foundry and related services. It unifies the management of cloud software across its lifecycle. BOSH can provision and deploy software over hundreds of VMs. It also performs monitoring, failure recovery, and software updates with zero-to-minimal downtime. BOSH supports many “immutable infrastructure” concepts. It enables two important elements of [cloud-native security](https://pivotal.io/cloud-native-security):
  + **Repair.** Repair vulnerable software as soon as updates are available.
  + **Repave.** Repave servers and applications from a known good state, and do this often. Malware thrives on vulnerable software and static, unchanging systems. Change the state of your environment frequently, and end malware-friendly conditions. Use BOSH to deploy your environment several times a day - all with no downtime and minimal manual effort.
* **CVEs, Patches, & Updates for a Broad Range of Elements.** Pivotal regularly patches PCF components, the underlying operating system, middleware, and third-party dependencies. When our security team identifies a "high" or "critical" CVE, they respond with a fix within 48 hours. These updates can applied using Ops Manager and other tooling, often without downtime. Refer to <https://pivotal.io/security> for additional information.
* **Encryption of Data at Rest.** The native encryption features of [Azure storage](https://docs.microsoft.com/en-us/azure/storage/storage-service-encryption) in the deployment achieve this requirement.
* **Encryption of Data in Transit (IPSec Add-on).** The [IPsec Add-on for PCF](http://docs.pivotal.io/addon-ipsec/index.html) encrypts IP data in transit in a PCF deployment. This module provides internal system protection if a malicious actor breaches your firewall.
* **Role-Based Access Controls.** PCF supports enterprise access controls with [Orgs, Spaces, Roles, and Permissions](http://docs.pivotal.io/pivotalcf/1-9/concepts/roles.html). This feature works in concert to ensure developers and operators have the right level of access. The Apps Manager is a web-based tool to help administer these roles and permissions.
  + **Org.** An org is a development account used by an individual or a team. Collaborators access an org with user accounts. Collaborators share the org's resource quota plan, applications, services availability, and custom domains.
  + **User Accounts.** A user account represents an individual in a PCF installation. A user may have different roles in different spaces within an org.
  + **Spaces.** Every application and service is part of a space. Each org contains at least one space. A space provides a shared location for application development, deployment, and maintenance. Each space role applies only to a particular space.
  + **Roles and Permissions.** A user can have one or more roles. These roles defines the user’s permissions in the org and within specific spaces in that org.
* **SSL/TLS.** [SSL/TLS certificates](http://docs.pivotal.io/pivotalcf/1-9/adminguide/securing-traffic.html) can secure HTTP traffic into your Elastic Runtime deployment. To secure non-HTTP traffic, terminate TLS at your load balancer or at the application with [TCP Routing](http://docs.pivotal.io/pivotalcf/1-9/adminguide/enabling-tcp-routing.html).

### Spring Cloud Services & Steeltoe

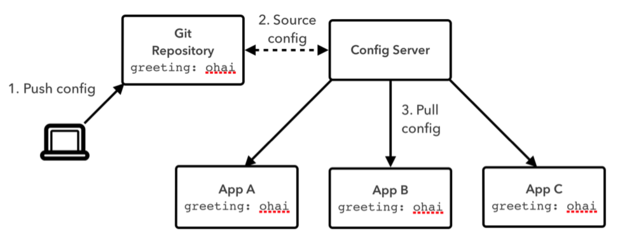
#### scs.pngOverview

[Spring Cloud Services (SCS) for PCF](http://docs.pivotal.io/spring-cloud-services/1-3/index.html) includes components of [Spring Cloud projects](http://cloud.spring.io/) (like [Spring Cloud Netflix](https://cloud.spring.io/spring-cloud-netflix/) and [Spring Cloud Config](https://cloud.spring.io/spring-cloud-config/)). SCS are available as services in the PCF Marketplace. With SCS, you do not have to manage or maintain these components, Pivotal does that for you.

Use SCS to create a [Config Server](#_fut9ugqu8nvd), [Service Registry](#_l45fgxogbi0r), or [Circuit Breaker Dashboard](#_4gv1otlx4psh) service instance on-demand. From there, you can bind to it and consume it. This again frees you to focus on the value added by your own microservices.

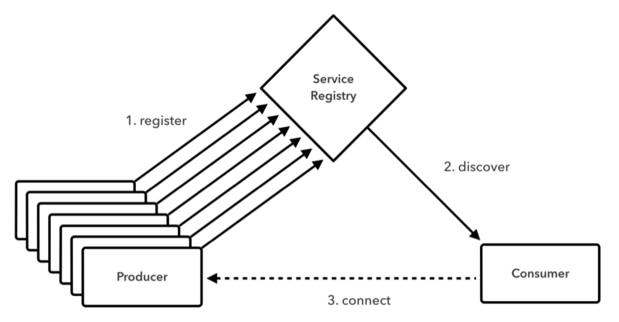
#### Config_Server.png Config Server for Pivotal Cloud Foundry

[Config Server for PCF](http://docs.pivotal.io/spring-cloud-services/1-3/config-server/) is an externalized application configuration service. This delivers a central place to manage an application’s external properties across all environments.   
  
Config Server will manage the configuration for an app as it advances through the deployment pipeline (dev, test, prod). Developers can be sure that an app has everything it needs to run during this process. Config Server supports labeled versions of environment-specific configurations.Users can manage configuration content with many tools, including Git. We also plan future support for Vault (as well as Git and Vault in composite fashion).

  
*Figure 2 - Config Server for Pivotal Cloud Foundry.*

#### Service_Registry.pngService Registry for Pivotal Cloud Foundry

[Service Registry for PCF](http://docs.pivotal.io/spring-cloud-services/1-3/service-registry/) provides an implementation of the [Service Discovery pattern, based on Netflix Eureka](http://docs.pivotal.io/spring-cloud-services/1-3/service-registry/resources.html). This is one of the key tenets of a microservice-based architecture.  
  
Manual configuration of each client or service is difficult. It often proves brittle in production. Instead, use Service Registry to dynamically discover and call registered services.

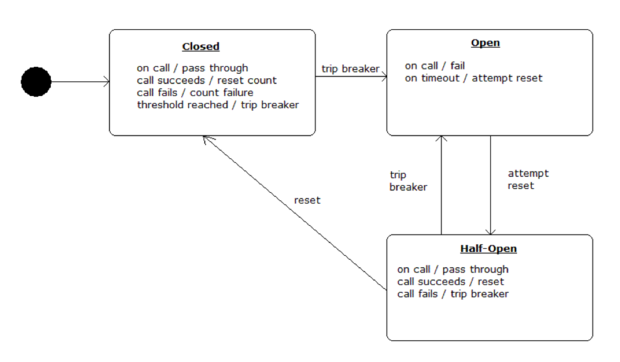


*Figure 3 - Service Registry for Pivotal Cloud Foundry.*

First, a client registers with the Service Registry. The client also provides metadata about itself, like its host and port. Once registered, the Registry expects a regular heartbeat message from each service instance. If the heartbeat message is not received consistently, the Service Registry removes the instance from its registry.

#### Circuit_Breaker.pngCircuit Breaker Dashboard for Pivotal Cloud Foundry

The [Hystrix](https://github.com/Netflix/Hystrix) library (part of [Spring Cloud Netflix](https://cloud.spring.io/spring-cloud-netflix/)) provides an implementation of the [Circuit Breaker pattern](http://docs.pivotal.io/spring-cloud-services/1-3/circuit-breaker/resources.html). The [Circuit Breaker Dashboard for PCF](http://docs.pivotal.io/spring-cloud-services/1-3/circuit-breaker/) visualizes the metrics of the circuit breakers inside an app. Cloud-native architectures are often composed of many layers of distributed services. End-user requests may comprise multiple calls to these services. If a lower-level service fails, that failure can cascade up to the end user and spread to other dependent services. Heavy traffic to a failing service can also make it difficult to repair. Hystrix circuit breakers can prevent failures from cascading. They can also provide fallback behavior until a failing service is restored to normal.



*Figure 4 - Circuit Breaker for Pivotal Cloud Foundry.*

When applied to a service, a circuit breaker watches for failing calls to the service. If failures reach a certain threshold, it “opens” the circuit. The circuit breaker automatically redirects calls to the specified fallback mechanism. This gives the failing service time to recover. The Circuit Breaker Dashboard provides operational visibility into the behavior of all of the circuit breakers present in a fleet of cloud-native services.



#### Microservices for .NET with Steeltoe

Spring Cloud Services bring common microservices patterns to Java developers. [Steeltoe](http://steeltoe.io/) does the same for .NET developers. Steeltoe helps .NET client apps integrate with Spring Cloud Services. Steeltoe also includes connector libraries for Cloud Foundry. This handles the parsing of environment variables (VCAP\_SERVICES) for you. As a result, extending apps with backing services like Azure SQL DB is that much easier.

Steeltoe includes three modules:

**Service Discovery**  
How do you make the interactions between your microservices reliable and failure tolerant? For starters, you need a service registry—basically a phone book for your microservices—so service consumers know exactly where to find healthy service instances. Steeltoe includes a .NET client for Netflix Eureka so your microservices can register themselves and discover other registered services.

**Config Server**  
“Strict separation of config from code” has become a cloud mandate, but that begs the question, where do you put it? And once you’ve externalized your config from your app, how do you track who changed what, when? Steeltoe leverages Spring Cloud Config Server so you can store your app’s config in a centralized, version-controlled git repo and then load it at runtime.

**Cloud Connectors**  
Steeltoe automatically wires up common backing services, because no microservice is an island. And because it was built by Pivotal, Steeltoe integrates elegantly with Cloud Foundry.

## 

## 

## Reference Architecture: Logical Infrastructure

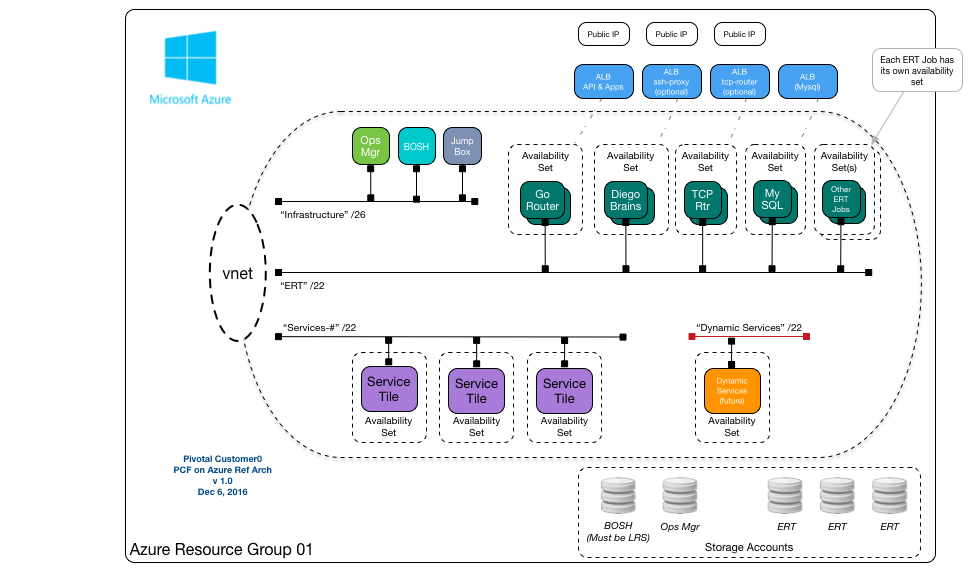
This Reference Architecture simulate a base deployment of Pivotal’s products.

### Logical Infrastructure: Microsoft Azure in the Public Cloud

The architecture in Figure 5 describes a secure, PCF Foundation hosted in Microsoft Azure.

The diagram shows core networking, created via an Azure virtual network. It has the following subnets:

* Infrastructure
* ERT (Elastic Runtime)
* Service tiles
* Dynamic or On Demand Service tiles



*Figure 5 - A logical reference architecture for Pivotal Cloud Foundry atop Microsoft Azure, featuring a single resource group.*

#### Availability Sets in Microsoft Azure

Some cloud providers provide VM availability guarantees through the use of explicit availability zones. In Azure, you can achieve equivalent availability guarantees through the use of availability sets. Availability sets automatically spread VMs across the Azure infrastructure to eliminate single points of failure, which are referred to as fault domains.

In this reference architecture BOSH will create multiple Availability Sets for your PCF deployment.

[**MICROSOFT CALL TO ACTION:** WE NEED TO ELABORATE ON THIS, I.E. BASED ON SCOTIABANK LEARNINGS]

#### IaaS Architecture

Here are the core Azure architectural constructs required to deploy PCF. Pivotal recommends that customers automate their deployment with the [Azure Resource Marketplace template](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/pivotal.pivotal-cloud-foundry). This reduces the likelihood of human error and makes your deployments more consistent.

Figure 5 shows a single PCF deployment in a single Azure Resource Group. Availability Sets within Azure expose infrastructure services to PCF. Use Availability Sets to deliver implicit HA for all virtual machines within a region.

The ‘base’ reference approach creates a single Resource Group. The Resource Group is then populated with the required Azure constructs, such as virtual networks, network interfaces, network security groups, public IP addresses, load balancers, and and storage accounts. From there, Pivotal Operations Manager deploys PCF.

* **One service principal account bound to an Azure Active Directory (AAD) application.** This enables BOSH to interact with the API. [Pivotal.io includes useful documentation on this step](http://docs.pivotal.io/pivotalcf/1-8/customizing/azure-prepare-env.html).
* **One or two Resource Groups per PCF installation.** PCF deployments on Azure need at least one resource group. Two resource groups are an option when more control over access to Azure networking objects is desired. Here, the 'Network' Resource Group manages Azure virtual networks. The second 'PCF' Resource Group is for objects deployed with BOSH.
* **Many Availability Sets, created by BOSH for each deployment job type.** Availability sets allocate BOSH jobs across 1 or more fault or upgrade domains. This allows jobs to complete, even if Azure failures occur. Each BOSH job in a PCF release gets their own availability set. This creates many instances, so that jobs can proceed in case of a single Azure Fault Domain failure. [See here for more information](https://docs.microsoft.com/en-us/azure/virtual-machines/virtual-machines-windows-manage-availability) on Azure Availability Sets.
* **One virtual network (vnet) with a large range of address space that is sub-divided.** Let's look at a common division.
  + Example: 10.xxx.yyy.0/20
    - One network for Infrastructure 10.xxx.yyy.0/26
    - One network for Elastic Runtime 10.xxx.yyy.0/22
    - One for Services-# 10.xxx.yyy.0/22
    - One for Dynamic Service-# 10.xxx.yyy.0/22
  + Note that a subnet is a logical component, bound to a single virtual network. It must exist in the the same Resource Group. In Multi Resource Group deployments, 'Network' and 'PCF resource groups should share the same region. This allows BOSH jobs in one resource group to attach networks to another.
* **One Network Security Group (NSG).** NSG manages firewall rules that apply to network interfaces. Ops Manager for Azure currently limits PCF deployments to 1 security group.
* **Four Azure Load Balancers (ALBs).** Load balancers are used as follows:
  + One for Public app access, enabling an API (or Control Path) and Apps (or Data Path)
  + One for Internal use, for example MySQL
  + One for an Optional TCP Routers, if selected
  + One for an Optional SSH Proxy
* **Five Standard Storage Accounts to match deployment needs.** Storage accounts are used to store the VM disks. Multiple storage accounts are suggested for two reasons. First, IOPS is capped at around 20,000, per account. Second, this avoids a single point of failure in the storage tier. Please note that customers pay for the *consumption* of storage, not the *number* of Storage Accounts.

The five accounts should be allocated as follows.

* + One for Ops Manager
  + One for BOSH
  + Three for Elastic Runtime and other tile deployments

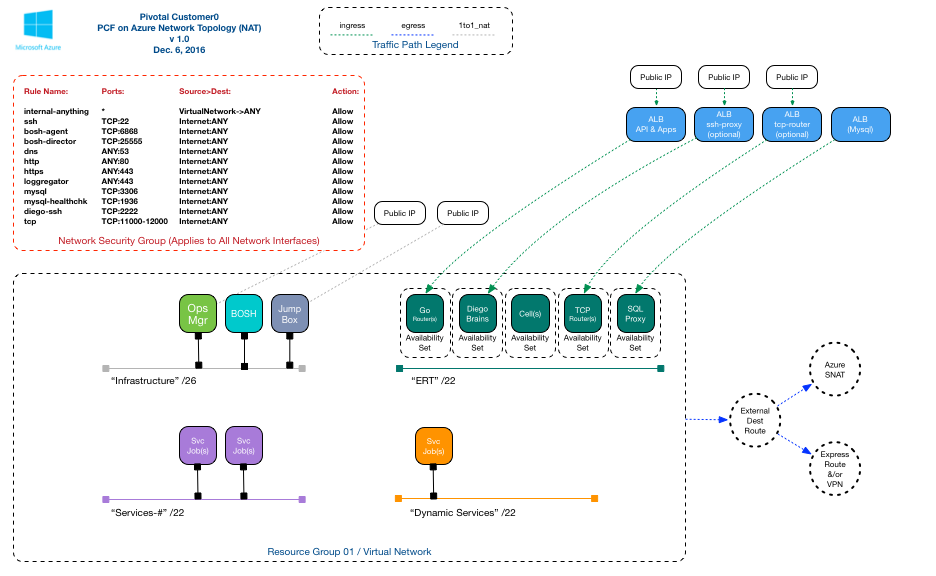
We recommend Premium storage for Elastic Runtime and tile deployments.

Support for Azure’s Managed Disks feature is planned for Pivotal Cloud Foundry.

* **One jump box on the infrastructure network to provide CLI tools.** A jump box inside your PCF deployment is a handy utility. That’s why a jump box is part of the [PCF Azure Resource Manager template](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/pivotal.pivotal-cloud-foundry). This template includes the jump box with the recommended CLIs, so this is done for you. Manual deployments will need to create this jump box, then install recommended CLIs manually. The CLIs are listed in the [Pivotal Customer0 repo](https://github.com/c0-ops/landingpage/blob/master/azure/refarch-main.md).
* **One to five public IPs, assigned as follows.** Note that public IPs are not needed if deploying with a VPN or Express Route Solution.
  + One VIP for Azure Load Balancer for CF domains (sys. and apps.)
  + One to SSH into jump box
  + One optional for VIP for Azure Load Balancer to TCP Routers
  + One optional HTTPS Ops Manager
  + One optional SSH Proxy to Diego Brains

#### Network Topology

Figure 6 shows the Azure network elements configured in practice.



*Figure 6 - A common network topology for Pivotal Cloud Foundry atop Microsoft Azure.*

#### Further Guidance: Microsoft Azure

Additional guidance for this reference architecture is in the [Pivotal Customer0 Github repo](https://github.com/c0-ops/landingpage/blob/master/azure/refarch-main.md). The repo also offers configurations for multi-resource group environments.

### Hybrid Considerations

The most popular hybrid configuration with enterprises? Using the public cloud as an extension of an on-premises datacenter. Microsoft support this scenario with [Azure Express Route](https://azure.microsoft.com/en-us/services/expressroute/).

Here, apps run on-prem, and in multi-tenant, elastic infrastructure. Administrators configure firewall rules in the public cloud. This restricts access, such that only certain IP ranges (those of a corporate network) can access the app.

Here are four areas to keep in mind for this scenario.

#### Data Replication

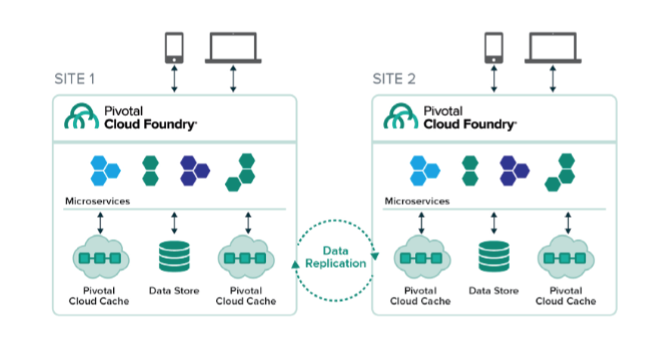
Data should be the foremost consideration in designing hybrid architectures. After all, most regulations and compliance standards apply to the treatment of data!

Traditional scenarios 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.

But, 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.

Pivotal Cloud Foundry customers achieve this with Pivotal Cloud Cache (PCC), a high-performance, highly available caching layer. 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 compresses and encrypts the 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 to ensure 100% data redundancy across locations.



*Figure 7 - Data replication across two sites, featuring Pivotal Cloud Foundry and Pivotal Cloud Cache.*

#### Routing Site Selection

If you’re running an app in the public cloud and in your datacenter, how do you want to route the traffic? Pivotal Cloud Foundry customers have chosen one of two options.

* **“Go through my data center every time.”** In this scenario, traffic flows through your existing global traffic manager. Administrators create routing policies based on custom rules. These policies steer requests to the most suitable site, either on-premises or the public cloud. Then, admins build firewall rules that only allow traffic from your data center to the public cloud. These rules block all other traffic into the company's public cloud instances. This hybrid setup requires a high-speed, dedicated connection between your data center(s) and the public cloud site. It is important to note that data is typically served from the on-premises site. Other application elements enjoy greater elasticity.
* **“Balance traffic across public cloud nodes and my datacenter.”** For this option, use DNS routing or a global load balancing service from a 3rd party. Pivotal Cloud Foundry easily supports this scenario. Deploy two PCF foundations, then configure each to work with the desired policies.

#### Active-Active or Active-Passive?

These terms mean different things to different audiences. For this whitepaper, let’s define active-active as real-time HA. Active-passive is a configuration where some lag in consistency or availability is tolerable.

One of the main obstacles to achieving active-active is latency between data centers. Distributed transactions cause latency, and should be avoided whenever possible. Access data locally; resolve requests in a single DC. Data partitioning can also mitigate latency. But, this comes with its own set of issues (consistency, who to trust for conflict resolution, and so on).

Refer to [Considerations for High Availability Across Multiple Sites](#_7ss6w6svsztw) for more on HA patterns at the app level.

Active-passive configurations are relatively easy. The passive site gets an update every X seconds or Y events, as described in the [Data Replication](#_xtxuxe8m91hh) section). If the active site goes down, you simply re-route traffic to the passive location.

Another option for replication: [SQL DB Geo-replication.](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-geo-replication-overview) This option replicates database transactions within a region and across Azure regions.

Customers may connect their private data centers to Azure data centers with [Azure Express Route](https://azure.microsoft.com/en-us/services/expressroute/).

Support for Azure Stack is planned.

#### Deployment Orchestration

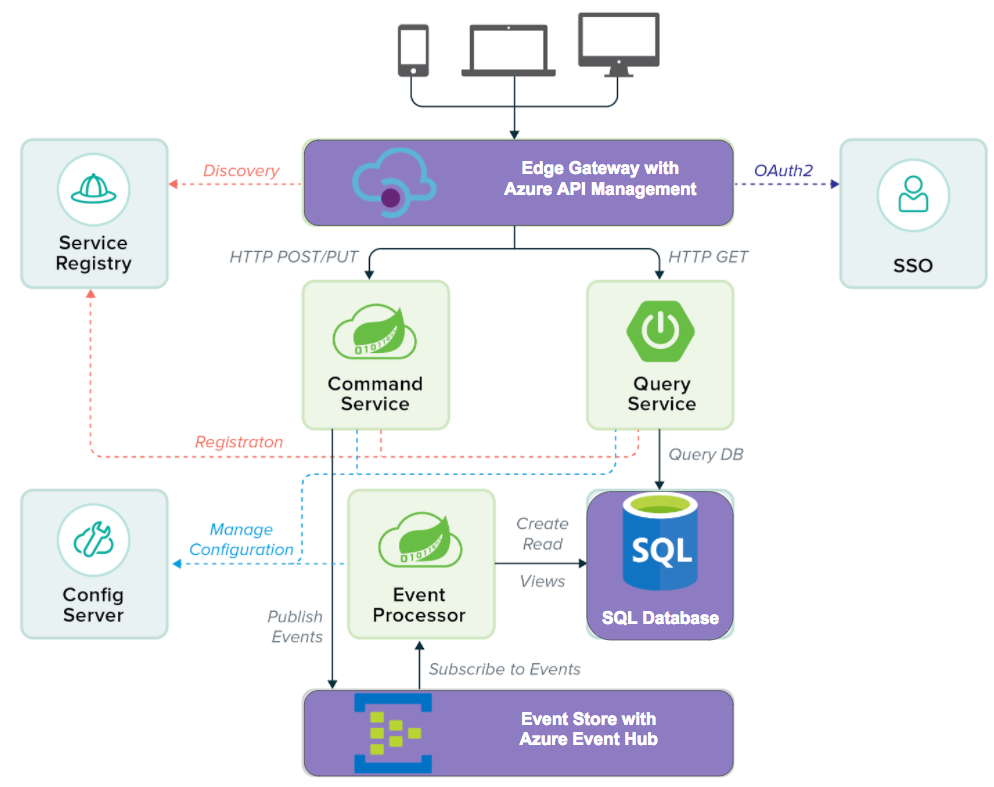
Continuous deployment is one [three crucial tenants for the digital era](http://www.slideshare.net/Pivotal/the-five-stages-of-cloud-native). Verify your CI/CD tooling and processes as you deploy asynchronous apps across different targets.Systems such as [Concourse](http://concourse.ci/) and [Spinnaker](http://www.spinnaker.io/) are particularly well-suited for this use case. [Visual Studio integrations](https://marketplace.visualstudio.com/items?itemName=ms-vsts.cloud-foundry-build-extension) also provide deployment automation to PCF instances.

There is one consideration to highlight: **keep track of database schema changes.** Replication errors and event mismatching will occur if schema versions are not aligned. Advance database versions together, and be able to roll them back together easily.

## Reference Architecture: Applications & Microservices Running on Pivotal Cloud Foundry

The architecture in Figure 8 features common cloud-native attributes.

* A modern approach to real-time data processing
* A highly distributed, scale-out pattern
* The *Command Query Responsibility Segregation (CQRS)* approach. This divides the system in two distinct parts. CQRS separates the components used for writing (executing commands) from those for querying. As such, the **Command Service** and **Query Service** are independently scalable services. They are decoupled, and can be operated separately.
* An event store for processing a high-volume of transactions



*Figure 8 - An event-driven, cloud-native application running Spring Cloud Services and Pivotal Cloud Foundry.*

### Reference Architecture Components

* **Edge Gateway, implemented with** [**Azure API Management**](https://azure.microsoft.com/en-us/services/api-management/)**.** This service facilitates communication between the API publisher and the services that consume the APIs. Other capabilities include dynamic routing, monitoring, resiliency, security, and more. In this architecture, it executes specified dynamic routing rules, and accommodates diverse inbound clients. A deeper integration is available by using a route service to bind an app to an API management proxy.
* **Service Registry, implemented as Spring Cloud Services Service Registry, tracks the services.** The Command Service and the Query Service both register with this module; the Edge Gateway uses the Service Registry to discover service locations and where requests should be routed.
* **SSO (Single Sign-On) for PCF.** The Single Sign-On service is an all-in-one solution for securing access to the application and its services. It improves security and productivity since users do not have to log in to individual components.
* **Command Service, written with Spring Cloud Data Flow, orders the system do something or change its state.** This represents part of the business logic of the application.
* **Query Service, a Spring Boot app, searches the relational database and returns relevant results.** In contrast to the Command Service, this does not change the state of the system. This service is likely to experience variable load, and can be scaled accordingly.
* **Config Server, implemented as Spring Cloud Services Config Server, stores all environmental variables.** No configuration is stored in any of the services. The Config Server stores this information and provides it on-demand. This is a cloud-native best practice, in keeping with “12 factor app” principles.
* **Event Processor, written in Spring Cloud Data Flow, works with the Command Service.** This is the other part of the application’s business logic.
* **MySQL for PCF serves as the “system of truth.”** It returns results requested by the Query Service.
* **Event Store, based on** [**Azure Event Hubs**](https://azure.microsoft.com/en-us/services/event-hubs/)**, is the high-volume event processing system.** This pattern is often referred to as “Event Sourcing”. It is a superior option for this scenario, since event streams in the past may need to be “replayed.” In contrast, systems like RabbitMQ immediately delete messages upon successful receipt. Azure Event Hubs natively supports C# .NET clients and Java clients.

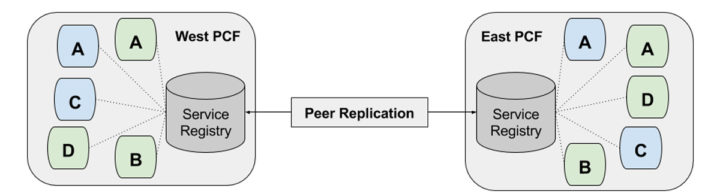
## Considerations for High Availability Across Multiple Sites

The [Hybrid Considerations](#_7109nr9ayk83) reviews some elements of high availability. Messaging middleware and the use of event-driven architecture address HA the application level.  
  
Distributed architectures thrive on asynchronous, message-driven communication between application services. This stands in contrast to traditional, synchronous systems that need massive bandwidth for traffic between clusters.

The event store in Figure 7 can be configured to send events to multiple sites. Now, let’s examine how to discover and consume services in different locations.

### Peer Replication with Service Registry

Consider a recent real-world use case from a financial services organization shown in Figure 9. This bank has a large number of services deployed across orgs and spaces (part of the permissions model reviewed in the [security section)](#_fc04ovzrauq). These services need to be discovered and used across the bank. Administrators want these services consumed by apps regardless of their PCF organization and space. With Service Registry peer replication, applications can be managed within the authorization model of a space, and services can be made accessible in a controlled way across organizations.

  
*Figure 9 - Multi-site peer replication for service registration.*

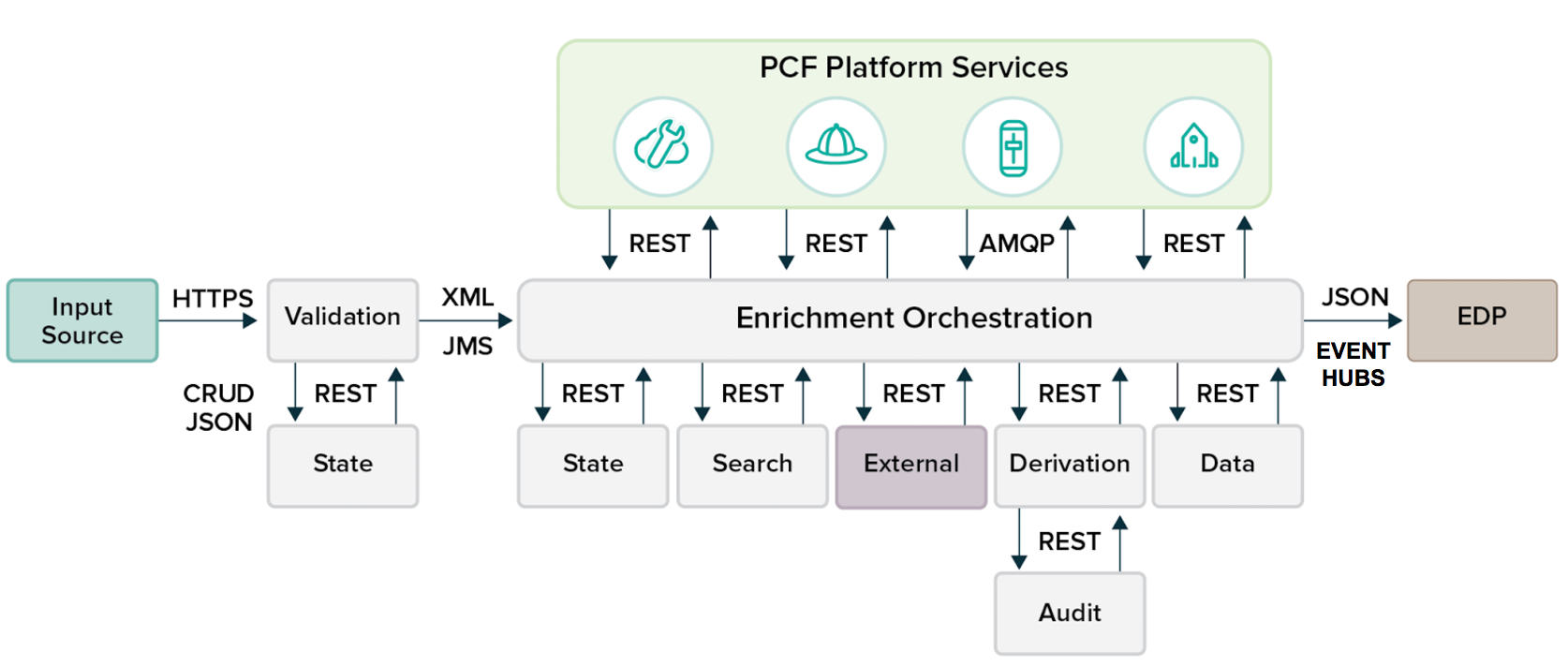
Service Registry peer replication works across PCF installations. It also works across organizations within a single PCF installation. The diagram above shows a simple Service Registry peer replication configuration across two sites. If we look at the **West PCF** data center, the local Service Registry 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.  
  
Once peer replication is enabled, development and operations teams enjoy:

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

More details on this scenario, including code samples, are [available on the Pivotal blog](https://content.pivotal.io/blog/new-in-spring-cloud-services-1-2-multi-site-service-discovery).

## Reference Architecture: Sample Customer App

Now, let's consider a Sample Application written by a PCF customer (Figure 11). We can see how the platform enables the developers to focus on their custom code. PCF and Spring Cloud Services take care of ongoing management and operation.



*Figure 11 - An production banking application architecture from a Pivotal customer.*

### Sample App Components

The customer app features these custom services:

* Validation & State
* Enrichment Orchestration
* State
* Search
* External
* Derivation
* Audit
* Data
* (EDP) Event Data Processing

### PCF Platform Services

Meanwhile, PCF the underlying management of the infrastructure and runtime dependencies. The platform also handles these essential elements:

* **Config Server.** [Config Server](#_fut9ugqu8nvd) manages the Sample App’s external configuration properties.
* **Service Registry.** [Service Registry](#_l45fgxogbi0r) performs Service Discovery for the Sample App’s microservices. It dynamically discovers and calls registered services for the Sample App.
* **Circuit Breaker Dashboard.** The Sample App includes multiple layers of distributed services. The Hystrix library (part of Spring Cloud Netflix) provides an implementation of the Circuit Breaker pattern. It can provide fallback behavior until a failing service is restored to normal operation. The [Circuit Breaker Dashboard for PCF](https://docs.google.com/document/d/1HkjX7DyY5szFiuwqmYmCHz-vUbXN3cQoRjFDYP8kDr0/edit#heading=h.4gv1otlx4psh) visualizes the metrics of the circuit breakers inside an app.
* **App Autoscaler.** The [App Autoscaler](http://docs.pivotal.io/pivotalcf/1-9/appsman-services/autoscaler/using-autoscaler.html) automatically adjusts capacity for this Sample App. It does so according to custom triggers and thresholds. The service adds and subtracts instances based on CPU Usage, HTTP Latency, and HTTP Throughput.

## Next Steps

We trust these materials have shown you how to best deploy Pivotal Cloud Foundry on Microsoft Azure, and adopt Spring Cloud Services. Now, take the next step!

Try out PCF on Azure from the [Azure Marketplace](https://azuremarketplace.microsoft.com/en-us/marketplace/apps/pivotal.pivotal-cloud-foundry). This creates a full PCF deployment, including the Azure Service Broker. Add Pivotal Spring Cloud Services and try out the powerful combination of Pivotal, Spring, and Azure.

Need more help? We offer architectural guidance and consulting for companies like yours.

At Pivotal, we help organizations assess and execute application replatforming—even operating the resulting applications on-platform for customers. Businesses of all sizes (from large enterprises to startups) across industries trust our software, services, and experience to lead them on their journey to the cloud—before, during, and after deployment.

* Work with Pivotal to identify target applications for replatforming that match the level effort and benefit tradeoff needed for the business.
* Take advantage of the Pivotal Labs approach, which uses Agile methodologies to gain results iteratively in 10-week sprints versus months of portfolio analysis before delivering value.
* Quickly find the minimum viable number of the twelve factors that must be addressed before applications can run in the cloud.
* Team with Pivotal to identify seams in the monolithic approach that can be extracted, migrating incrementally and keeping value flowing to the business as apps that need the full benefit of cloud-native design are replatformed.
* Deploy and manage your applications with Pivotal Cloud Foundry—our cloud-native platform that accelerates time to value.