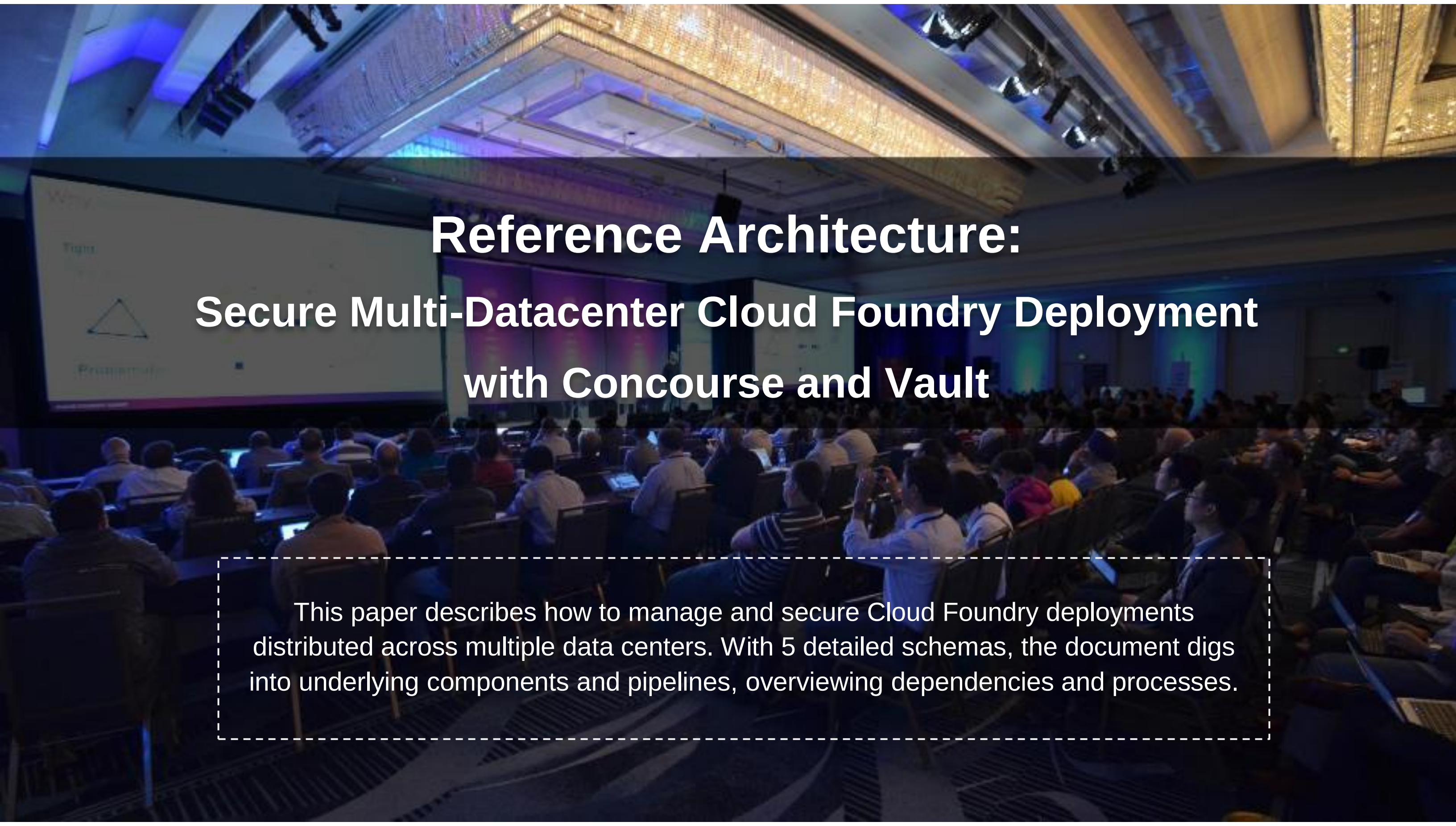

Reference Architecture: Secure Multi-Datacenter Cloud Foundry Deployment with Concourse and Vault

This paper describes how to manage and secure Cloud Foundry deployments distributed across multiple data centers. With 5 detailed schemas, the document digs into underlying components and pipelines, overviewing dependencies and processes.

Q2 2017

By Alexander Lomov, Cloud Foundry Engineer, Altoros

Table of Contents

Featured images

1. Introduction

This document describes how to securely manage Cloud Foundry deployments with Concourse, Vault, and BOSH. It provides a way to create repeatable and secure deployments with related services. Concourse and BOSH were specifically designed for deploying, managing, and updating Cloud Foundry with related services.

What is **Vault**? It is a tool for storing and managing secrets, such as API keys, passwords, certificates, etc. In addition, all configuration properties are also stored in Vault: services' IP addresses, connection properties, and so on. The tool provides a unified interface to any secret, while granting tight access control and recording a detailed audit log. Vault is also responsible for secrets encryption on a hard drive, as well as for safe delivery of secrets to final recipients.

Using Vault as a centralized storage for secrets and configuration has a number of advantages:

1) With appropriate access setup, it allows for sharing settings between deployments, reducing operator efforts, removing manual interaction from the setup stage, and reducing the area, where a human error can occur.

2) Vault is designed from the ground up as a secret management solution and brings a number of useful features out of the box: secure secret storage, dynamic secrets, data encryption, leasing and renewal, secrets revocation, password rotation, etc.

2. Alternative options

Relying on Vault for configuration management, though, is not the only option. Here is a list of other solutions for the purpose:

1) Config Server from BOSH
2) CredHub from the Cloud Foundry community

Using a hardware security module (HSM) is also an option, but it is too expensive for non-specialized usage, while the community solutions are still in the earliest production stage. Nevertheless, the architecture described below is capable to work with any of these options.

Why not using Config Server? The tool is created by the BOSH team and may seem the best solution to use in bundle with BOSH. Still, Config Server is oriented only on BOSH deployment and, now, is in the pre-beta stage. Here is a list of reasons why we are going to use Vault instead::

1) Config Server is not out yet.

2) It is a BOSH-oriented tool. There are pipelines with more configuration data than it is provided in Config Server.

3) We do not have control over properties that are stored in Config Server.
4) BOSH deployment packages (such as `cf-deployment`) will take time to transition from current variable management with `bosh-cli`.
5) Potentially, one may have customers that will be interested in storing configuration using different tools.

All the mentioned above is the reason we are using Concourse, BOSH, and Vault to enable robust and sleek deployments. Adding Vault to the architecture as a shared secret storage gives a possibility to provide deployments with access information (e.g., IP addresses, secrets, certificates, etc.) about each other. The Vault storage is also shared between deployments, which eases the creation of multi-datacenter deployments with MySQL, Cassandra, and other databases. Using a secret storage is a key requirement to building an *active-active* (single-machine HA) Cloud Foundry deployment.

3. Architecture description

The main high-level components of the system are:

1) A control plane
2) Environments
3) Products

A **control plane** (or a management plane) is a set of components used to create and manage components in different environments.

Environment is a way to split your deployments to comply with functional logic or to fit your infrastructure demands. Each environment can be represented by a distinct data center or a logical unit. Most likely, each environment will contain a BOSH instance and Cloud Foundry together with integrated services. A control plane is usually deployed in a distinct data center or a physical server to make it independent and fail-proof.

Each environment contains a set of **products**—clusters of services deployed by BOSH. Products can be deployed by using either BOSH releases or PCF Tiles. Typically, each environment will have a Cloud Foundry installation with MySQL, RabbitMQ, and Redis services.

On a diagram bellow, you will find an example of how environments can be distributed across data centers. As you can see, all the environments are managed by a control plane, which consists of BOSH, Concourse, and Vault. You will find the detailed description of the control plane in the following sections.

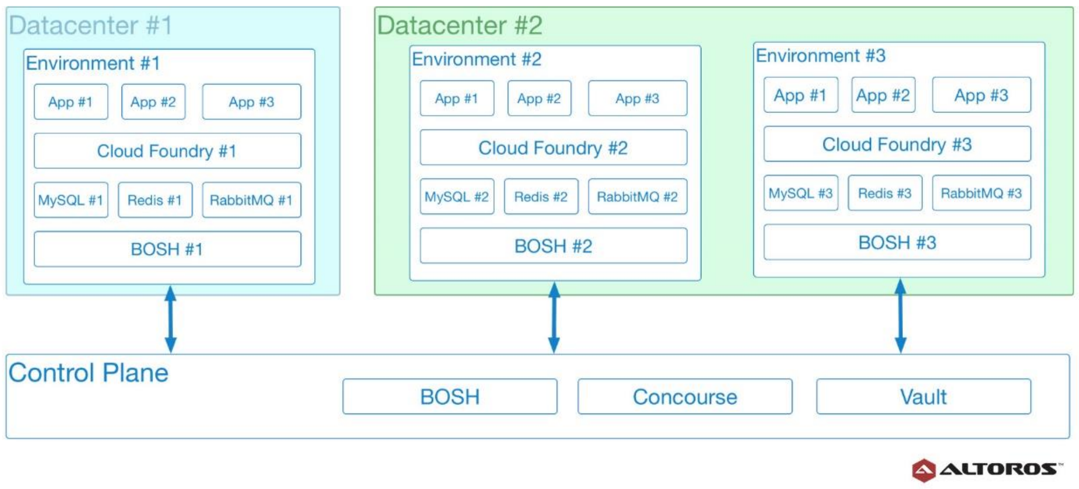

Figure 3.1 *Creating multiple environments distributed across data centers using a control plane*

As mentioned earlier, using Vault allows for easily sharing secrets and configuration data between environments. It also enables engineers to create and synchronize Cloud Foundry deployments as a part of a single system. These deployments are commonly referred to as *active-active* deployments, having a load balancer in front of them as a single entry point for user traffic. The main benefit of such deployments is improved availability: a system like that can easily survive outage of at least one data center. Running system upgrades also becomes more convenient. This approach can be simplified by using Vault and Concourse pipelines.

You can see an example of such deployments on the diagram below. The picture demonstrates an active-active Cloud Foundry deployment in two data centers running vSphere. Each of the layers has its own responsibility, supporting other layers:

1) **The supervisor layer** is represented by the control plane components and is used to create and manage all the other layers. The detailed description of this layer can be found in the following sections.
2) **The virtualization layer** is responsible for spinning up a new infrastructure, with any kind of virtualization supported by BOSH: AWS, vSphere, Google Compute Engine, Microsoft Azure, OpenStack, SoftLayer, RackHD, VMware Photon, etc.
3) **The Ops tools layer**. This one usually contains the components used to maintain an environment. An operator has an access to this layer only through the control plane.
4) **The data services layer** is responsible for storing and sharing state data used by application instances. In order to provide high availability, we deploy active-active clusters of MariaDB and MongoDB to environments in several different data centers.
5) **The runtime layer** is served by Cloud Foundry itself to run applications and bind them with data storage services. Since applications deployed to Cloud Foundry are stateless and share data only through active-active clusters of data services, application data is always up-to-date and accessible.

The consistency between two installations of Cloud Foundry is reached by using special pipelines:

1) *The Orgs, Space, Users, and Quotas Management pipeline* uses a Git repository and LDAP as a source of configuration and manages these resources.

2) *The CF Sync pipeline* checks whether the deployed Cloud Foundry instances have the same resources, sending notifications if the resources are not synchronized.

3) *The Applications Deployment pipelines* used to test and deploy applications—usually, these pipelines are application-specific and developed in cooperation with a customer.

In addition, all these pipelines allow for simplifying the backup procedure and tracking who updated what within the application runtime.

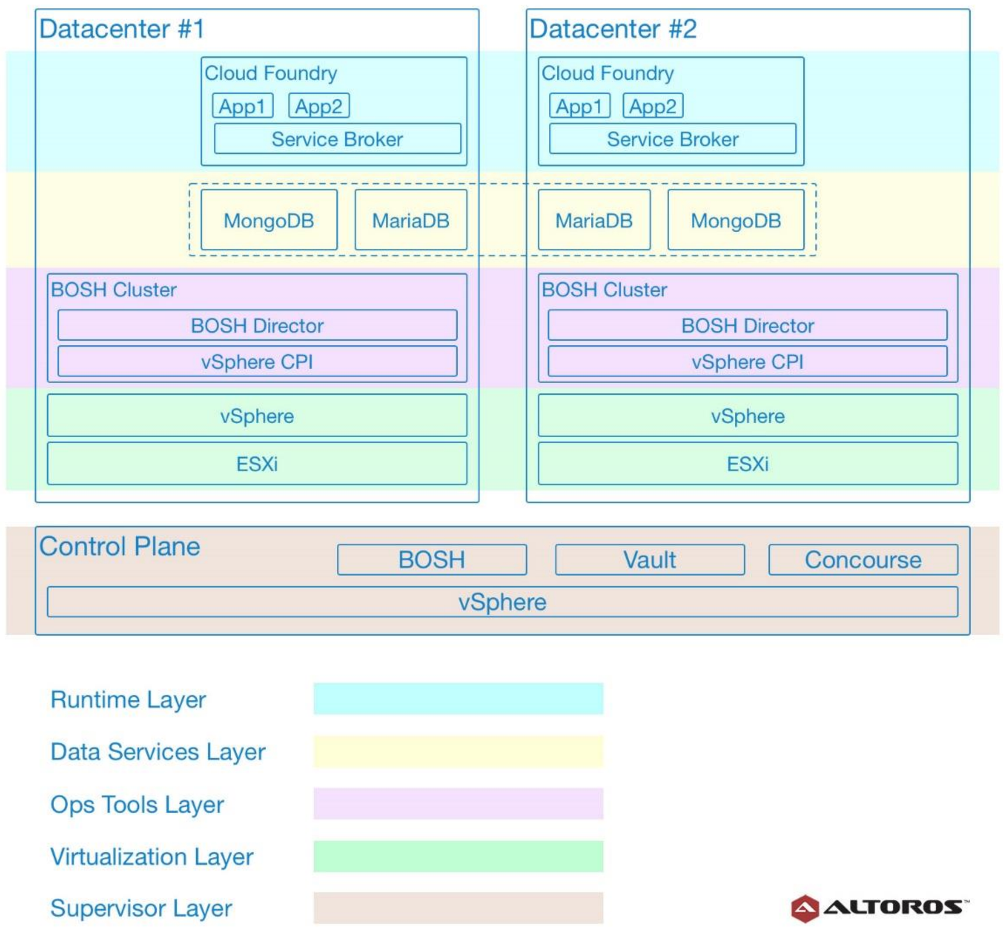

Figure 3.2 *The layers of an active-active Cloud Foundry deployment*

The next diagram describes the process of creating and managing the system, also representing dependencies between the components.

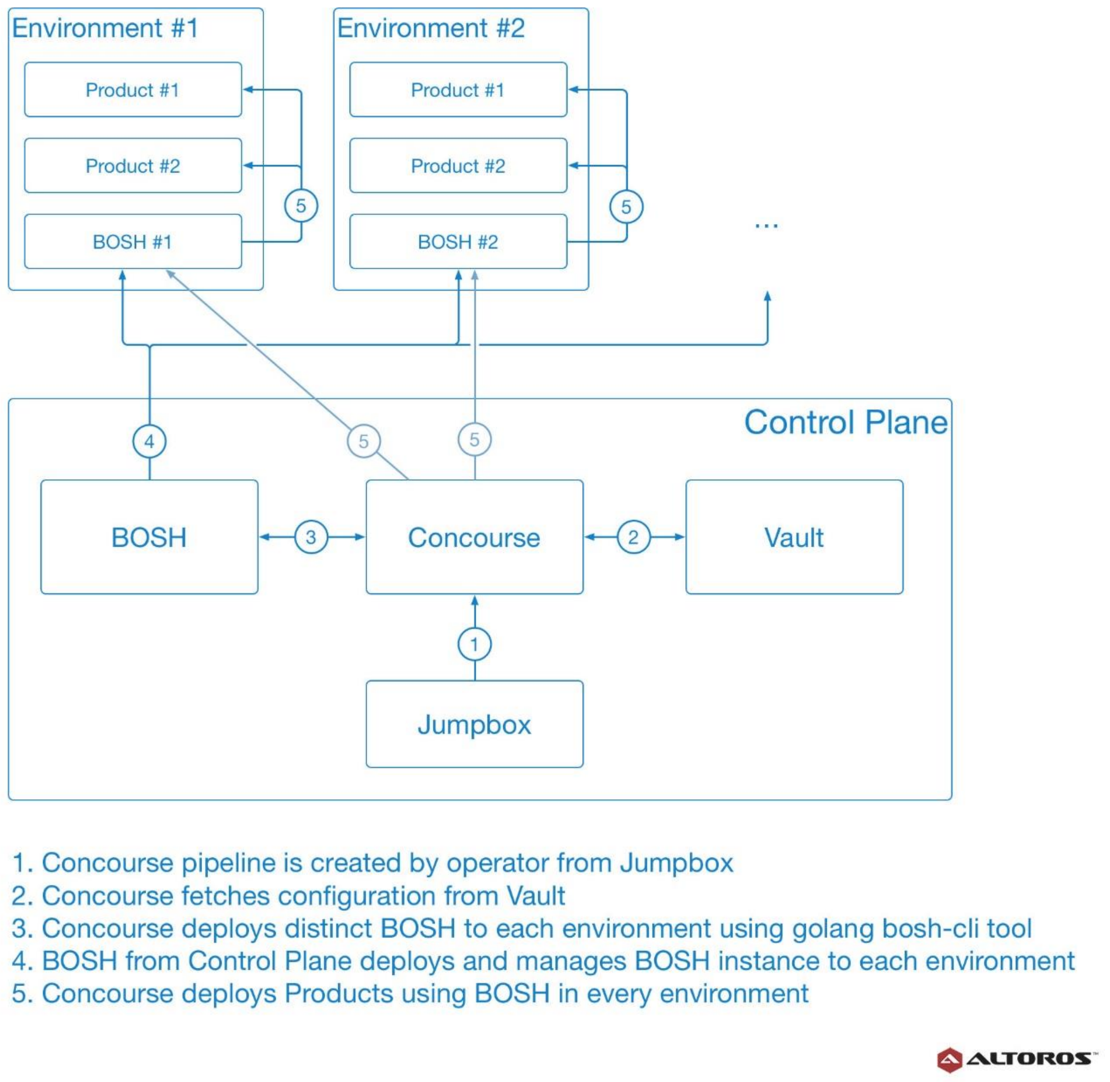
Environment #1
Product #1
Product #2
BOSH #1
Environment #2
Product #1
Product #2
BOSH #2
...
Control Plane
BOSH
Concourse
Vault
Jumpbox
1. Concourse pipeline is created by operator from Jumpbox
2. Concourse fetches configuration from Vault
3. Concourse deploys distinct BOSH to each environment using golang bosh-cli tool
4. BOSH from Control Plane deploys and manages BOSH instance to each environment
5. Concourse deploys Products using BOSH in every environment
ALTOROS

Figure 3.3 *The process of deploying an environment with BOSH, Concourse, and Vault*

4. A control plane

A **control plane** (or a management plane) is a set of instruments that aids engineers in creating and managing components in different environments. Each environment can be represented by a distinct data center or a logical unit. Most likely, the environment will contain Cloud Foundry with integrated services.

Control plane is essential for managing the environments, so it is often deployed in a separate data center or a dedicated physical server.

Three major components of the control plane are:

1) A **jumpbox**, which is a VM with a set of user tools (such as BOSH CLI, Fly CLI, etc.) that are used to manage the control plane. Full access rights are granted only to a limited number of operators due to security reasons.
2) **BOSH**, which is used to create and manage BOSH instances inside of environments that you want to deploy. You can access a Control Plane BOSH instance only from Concourse.
3) **Concourse**, which stores pipelines for managing single or multiple environments. Access to different pipelines can be managed through the UAA and Concourse teams—an internal mechanism that grants access right to groups of engineers.
4) **Vault**, which is used to store configuration for each environment. Only a Concourse instance and a jumpbox should have access to a Vault instance (some config processing components can also access Vault).

All the components on this list can be secured from failure by replication and backup. The easiest ways to create backups are either by using the BOSH snapshot feature (not all IaaS systems support this feature) or by storing backups in a jumpbox or a blob storage.

The control plane manages single or multiple environments. There are different scenarios of how this management process can be performed. Each of them has its benefits and caveats.

You can see a schematic representation of the control plane components deployed to vSphere on the following diagram.

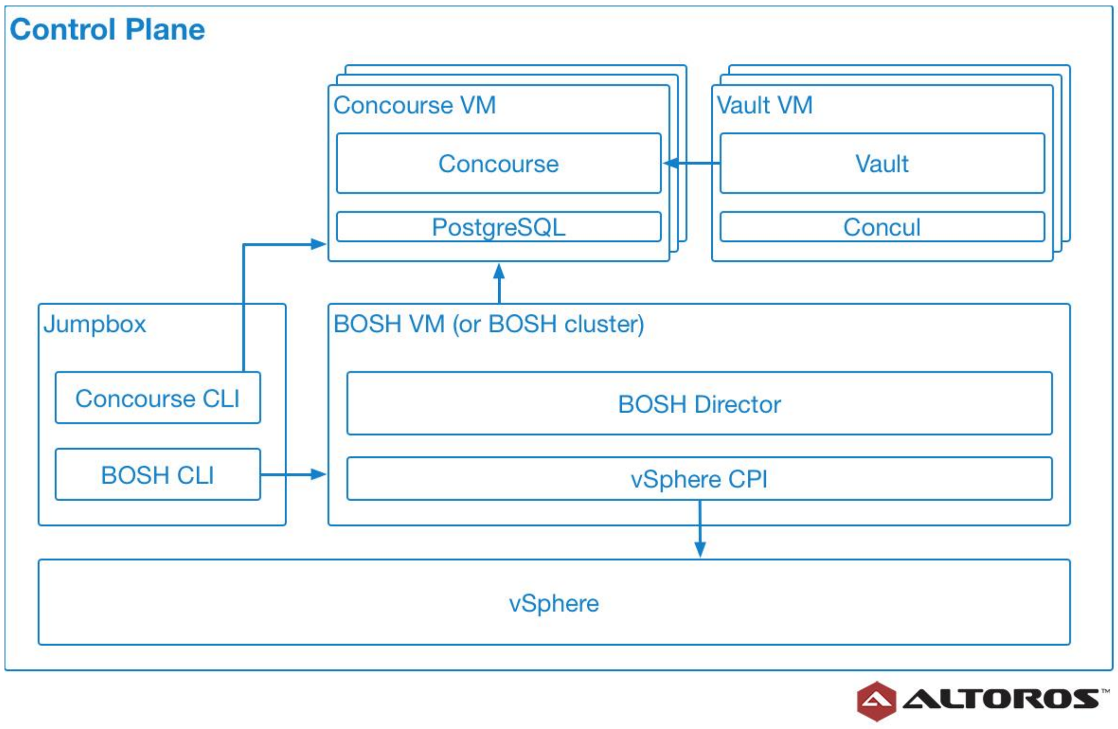

Figure 4.1 *The components of a control plane*

5. Pipelines

After the control plane is installed, we can load the pipelines—which create and manage products in all the environments—to Concourse. Each pipeline serves one product or executes one type of tasks (for instance, orgs and spaces management). We have two major types of pipelines:

1) A pipeline for component **deployment**. (Such components can be Pivotal CF, a Redis cluster with Redis service brokers, MySQL with a service broker, etc.)
2) A pipeline for component **management** (orgs / space / users management; cf audit—checks if there are unemployed quotas, service instances, etc.; cf sync audit for Cloud Foundry active-active deployments—checks if several CF deployments have the same orgs/spaces, etc.)

Here is a list of pipelines that are used to create and support the architecture described above:

1) The **BOSH deployment** pipeline deploys BOSH and stores its properties to Vault; it can also be used to initialize or update other pipelines.
2) The **Cloud Foundry deployment** pipeline deploys or updates a Cloud Foundry installation using an available BOSH instance.
3) The **orgs, space, users, and quotas management** pipeline synchronizes the org/space/user structure between several Cloud Foundry deployments. It uses a Git repository and LDAP as a source of configuration and manages these resources.
4) The **CF Sync** pipeline checks whether Cloud Foundry deployments have the same resources (org, space, users, and applications), sending notifications if the resources are not synchronized.
5) The **Redis deployment** pipeline uses the Cloud Foundry community's Redis service and a service broker.
6) The **MySQL deployment** pipeline uses the `p-mysql` PCF Tile.
7) The **active-active MySQL deployment** pipeline uses the Cloud Foundry community BOSH releases.
8) The **RabbitMQ deployment** pipeline uses the `p-mysql` PCF Tile.
9) The **resilience and availability testing tools** pipelines:
    a. The **Turbulence** pipeline randomly removes VMs created by BOSH, imitates network split, etc. The same principle as with Chaos Monkey.
    b. The **Chaos Loris** pipeline. The same with Turbulence, but for application instances in Cloud Foundry.

All these pipelines are created by using unified approaches with minimized number of inputs. This allows for implementing new pipelines faster and minimizing operator's efforts involved.

Each product that is managed by a pipeline has its dependencies: Cloud Foundry depends on BOSH, Redis depends on BOSH and CF, while Spring Cloud Services depend on BOSH, CF, and MySQL. The dependencies can be viewed on the diagram below.

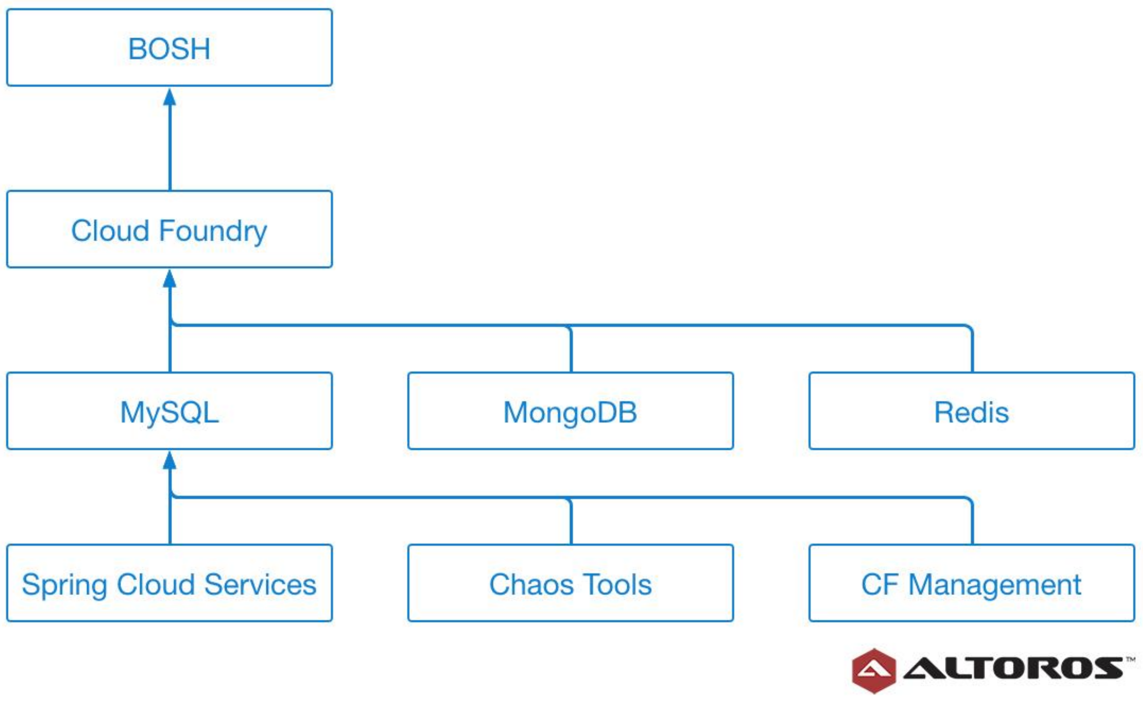

Figure 5.1 *Pipeline dependencies*

6. Backup, recovery, and updates

There are different layers of data that should be backed up. Firstly, a control plane, which is essential to restore all the other runtime environments. All the components of a control plane can be backed up by dumping the corresponding databases. In case with Vault, we can retrieve its content as encrypted data. During the restoring process, Vault will require to go through the unsealing procedure.

In case with runtime environments, we need to dump the Cloud Controller database and store application droplets to be able to restore. This requires us to create additional pipelines for such procedures. However, if the state of the applications is stored in external services protected from data loss (e.g., in an active-active MariaDB installation with replication), the easiest option to recover is to redeploy the applications and bind them with the necessary services.

BOSH allows for rolling out updates of its releases, which results in accessing applications with zero downtime. Still, such updates should be carefully planned and performed in a separate environment. It is essential to have an additional runtime environment to test all the updates. The proposed solution based on BOSH, Concourse, and Vault allows for running repeatable Cloud Foundry deployments easier.

7. Conclusion

The architecture involving a control plane—built upon BOSH, Concourse, and Vault—enables predictable, repeatable, and configurable Cloud Foundry environments across multiple data centers. Furthermore, this approach allows for protecting sensitive data and sharing it across active-active deployments to multiple clouds. It also makes designing active-active and hybrid architectures easier.

To get started with Vault, visit its GitHub repository or the project's documentation.

8. About the author

Alexander Lomov is a Cloud Foundry engineer at Altoros. With extensive experience in Ruby, Go, and Python, he was involved in development of BOSH CPIs and other Cloud Foundry-related projects for Canonical, IBM, and other companies. Alexander is a frequent speaker at various events/meetups, mostly sharing his experience with Cloud Foundry. You may also know him as the author of several blog posts about Cloud Foundry internals.

__Altoros__ brings "software factories" into Global 2000 organizations through consulting, training, deployment, and integration of the solutions offered by the Cloud Foundry ecosystem. With 300+ people strong team across 8 countries in Europe and Americas, the company is behind some of the world's largest Cloud Foundry deployments. Altoros is focused on turning cloud-native app development, customer analytics, blockchain, and AI into products with a sustainable competitive advantage. For more, visit www.altoros.com.

To download more technical guides and tutorials like this one:

- *check out our resources page,*
- *subscribe to the blog,*
- *or follow @altoros for daily updates.*