WHITE PAPER

Unlocking Operational Efficiency with BOSH

By Peter Blum, Jared Ruckle, Fadzi Ushewokunze, and Tony Hansmann

Pivotal

Table of Contents

Pivotal

Introduction

In today's connected world, we have fitness coaches on our wrist. We expect everything to be delivered to our doorstep in two days or less. We watch over a 100,000 hours of videos online without a late fee.

This is innovation powered by developers, new application architectures, and cloud infrastructure. Together, they drive new demands on operations teams: monitoring, failure recovery, security, and zero-downtime updates. For an enterprise to succeed in this rapidly changing world, they must meet these new demands, from the infrastructure upwards.

Dr Nic Williams, a noted contributor to Cloud Foundry and BOSH open-source projects, says success (or failure) comes down to answering questions like:

- How do you upgrade the entire system to a new version?
- How do you test the upgrade of an entire system from one known version to another?
- How do you package the entire system into versioned software that can be systematically deployed?
- How do you resize host machines, replacing older/smaller servers with bigger/newer types?
- How do you resize persistent disk volumes?
- How do you rotate host machines with new base images containing zero day security fixes?
- How do you repave host machines every few hours or days to limit the processing time available to hackers?
- How do you automatically resurrect missing host machines, replacing them with new machines and reattaching persistent volumes to return data services back into operation with minimum downtime?

BOSH is the toolchain that helps operations teams rise to the challenge.

Use BOSH for rapid, flexible infrastructure management. Deploy software faster, and more effectively than ever before. Dynamically update infrastructure. Focus less on preparing for deployments, and spend more time optimizing production.

BOSH deployments are written in an explicit format closely resembling an application's structure, what's often called "infrastructure as code". Consequently, there are no "snowflake" servers, hidden scripts to run, or rogue software packages. Technology operators and developers alike can model what is required and currently deployed on the infrastructure, resulting in higher agility and faster rollout of technology as a whole.

Before we go much further into BOSH, it's worth some introspection. How did we get here?

To Build & Run Modern Software Systems, Think Different

For the last two decades, monolithic applications ruled enterprise IT. Today, these systems are a paradox—crucial to business processes, yet a painful obstacle to tomorrow's requirements. While business demands speed and agility, these systems are slow and cumbersome.

Engineers spend much of their time reconciling these two realities. And it's not an easy task. This conundrum drove developers to experiment with new application architectures running on inexpensive, elastic infrastructure.

These teams quickly bumped into new obstacles. We collectively call these "Day 2 operations":

- Developers create applications that require many more external dependencies than ever before. This increases the management burden for an understaffed operations group.
- Hand-built software, designed to support a single, defined group of developers, are now expected to be on-demand and highly available across multiple regions. These systems also need to scale to millions of users.
- The twelve-to-eighteen month development cycle is no more. New code is pushed every week, even daily. And customers expect to use new features immediately.
- Security risks have increased exponentially. Traditional security practices, focused on slow rates of change, conflict with the cloud-native reality of the business.

These pervasive problems spawned new approaches to wrangling distributed systems. Like most other noteworthy tech of the last two decades, it came from the open-source community.

BOSH: An Open Source Story

BOSH wasn't designed by a single development group in a lab. Instead, it originated with engineers from over 60 organizations. It's a classic open source, community-driven project.

BOSH had humble beginnings in 2011. It's original intent: manage Cloud Foundry deployments. Cloud Foundry is a cloud-native platform, designed with a microservices architecture.

Chef was the original deployment tool for Cloud Foundry. But the community found many areas where it just didn't measure up. Project committers determined what was broken. Then, they began the BOSH project to tackle it. The community's rallying cry: "Cloud Foundry is the largest, most demanding microservice application. We'll create BOSH to manage it."

Over time, the BOSH project grew, and became a flexible tool for release engineering and lifecycle management for distributed systems like Hadoop, RabbitMQ, MySQL, and even Kubernetes.

Now, let's explore why BOSH is so popular with companies running distributed systems.

BOSH & Modern Release Engineering

Software goes through four phases - versioning, packaging, deploying, and maintaining. Each phase has its own challenges. Sure, operators can pluck a point solution from a sea of tools for any one phase. However, BOSH was designed to help operators complete each phase in a consistent, highly integrated manner.

Here's how BOSH measures up to the principles of modern Release Engineering.

RELEASE ENGINEERING PRINCIPLE (SOURCE: WIKIPEDIA)	BOSH ATTRIBUTE
Identifiability: Being able to identify all of the source, tools, environment, and other components that make up a particular release.	BOSH has a concept of a software release which packages up all related source code, binary assets, configuration etc. This allows users to easily track contents of a particular release. In addition to releases BOSH provides a way to capture all Operating System dependencies as one image.
Reproducibility: The ability to integrate source, third party components, data, and deployment externals of a software system in order to guarantee operational stability.	BOSH tool chain provides a centralized server that manages software releases, Operating System images, persistent data, and system configuration. It provides a clear and proven way of operating a deployed system.
Consistency: The mission to provide a stable framework for development, deployment, audit, and accountability for software components.	BOSH software releases workflows are used throughout the development of the software and when the system needs to be deployed. BOSH centralized server allows users to see and track changes made to the deployed system.
Agility: The ongoing research into what are the repercussions of modern software engineering practices on the productivity in the software cycle, i.e. continuous integration.	BOSH tool chain integrates well with current best practices of software engineering (including Continuous Delivery) by providing ways to easily create software releases in an automated way and to update complex deployed systems with simple commands.

The BOSH Deployment Model

BOSH unifies release, deployment, and lifecycle management of entire distributed systems. It encompasses the software layer, as well as the control and management of the underlying infrastructure.

BOSH implements 3 layers of packaging. Together, these create the BOSH deployment model, shown in Figure 1. Starting from the bottom:

- A stemcell is an operating system for the deployment. This can be Linux or Windows Server 2012.
- A release is the source code for what the operator wants to deploy. Examples include Cloud Foundry, Kubernetes, Redis, RabbitMQ, and etcd.

- A manifest is a configuration file that describes how the system should be deployed to its chosen infrastructure target.

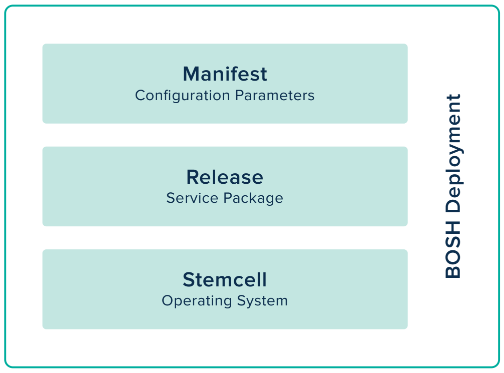


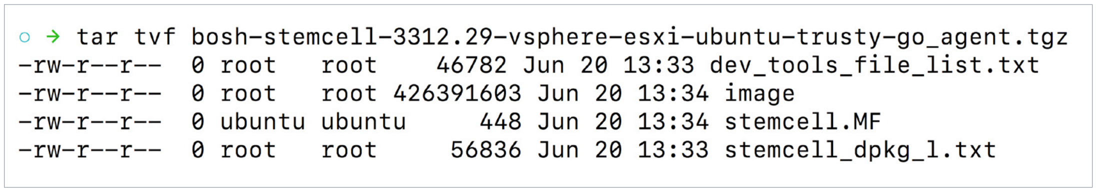
Figure 1: BOSH's flexible deployment model, unifying all components of a software system into a single BOSH deployment

Let's step through these 3 layers, starting with the stemcell.

BOSH Stemcell

A stemcell is the base Operating System (OS) image that powers the release. Stemcells are created with a minimalist mindset; each includes a slimmed-down OS, a few common utilities, and the BOSH agent. Let's consider an example stemcell, shown in Figure 2. Three pieces stand out:

- **`image`** — OS image in a format understood by the target infrastructure (usually a .raw, .qcow, or .ova)
- **`stemcell.MF`** — a YAML file with stemcell metadata
- **`stemcell_dpkg_l.txt`** — a text file that lists of packages installed on the stemcell. This file is simply a reference to tell the operator what's included in the stemcell.



```
○ → tar tvf bosh-stemcell-3312.29-vsphere-esxi-ubuntu-trusty-go_agent.tgz
-rw-r--r--  0 root   root    46782 Jun 20 13:33 dev_tools_file_list.txt
-rw-r--r--  0 root   root 426391603 Jun 20 13:34 image
-rw-r--r--  0 ubuntu ubuntu     448 Jun 20 13:34 stemcell.MF
-rw-r--r--  0 root   root     56836 Jun 20 13:33 stemcell_dpkg_l.txt
```

Figure 2: BOSH's minimal stemcell architecture.

As a first step in a deployment, BOSH creates virtual machines (VMs) and lays down a versioned OS. The BOSH team creates and hardens stemcells daily by configuring them to very strict requirements, thereby reducing their attack surface. As a result, each stemcell has fewer vulnerabilities.

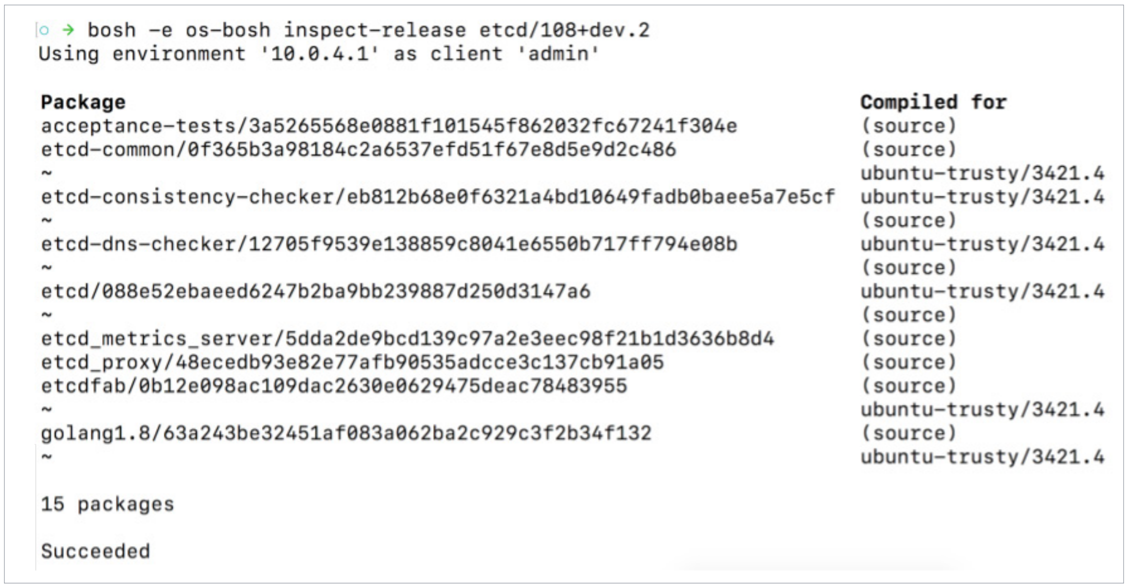
Stemcells provide a powerful separation between the OS and the other software packages bundled in a deployment. Each stemcell — no matter the underlying infrastructure — is exactly the same. This allows for rapid, reliable mobility between different infrastructure targets. Stemcells are distributed and updated via https://bosh.io.

BOSH Release

Releases are self-contained software packages. They include all the components required to startup, run, monitor, and shutdown a given distributed system. BOSH releases have three nifty properties:

- Releases include source code wrapped into packages. Consequently, BOSH ensures that all deployments run across any operating system on any infrastructure. For example, an etcd release must include all source that it depends on. This allows BOSH to compile each release for the desired operating system and infrastructure. Figure 3 shows an etcd release along with each package (dependency) compiled for the Ubuntu operating system.
- The separation and layering of software packages from the operating system allows BOSH to systematically version and update systems iteratively across the entire infrastructure fleet.
- Releases are stored within a blobstore. This ensures that the specific version of a software release is available throughout the lifecycle of the deployment.

Solving all of these problems — with a single solution — is critical to the BOSH deployment model.



```
o → bosh -e os-bosh inspect-release etcd/108+dev.2
Using environment '10.0.4.1' as client 'admin'

Package                                                          Compiled for
acceptance-tests/3a5265568e0881f101545f862032fc67241f304e        (source)
etcd-common/0f365b3a98184c2a6537efd51f67e8d5e9d2c486             (source)
~                                                                ubuntu-trusty/3421.4
etcd-consistency-checker/eb812b68e0f6321a4bd10649fadb0baee5a7e5cf ubuntu-trusty/3421.4
~                                                                (source)
etcd-dns-checker/12705f9539e138859c8041e6550b717ff794e08b        ubuntu-trusty/3421.4
~                                                                (source)
etcd/088e52ebaeed6247b2ba9bb239887d250d3147a6                    ubuntu-trusty/3421.4
~                                                                (source)
etcd_metrics_server/5dda2de9bcd139c97a2e3eec98f21b1d3636b8d4     (source)
etcd_proxy/48ecedb93e82e77afb90535adcce3c137cb91a05              (source)
etcdfab/0b12e098ac109dac2630e0629475deac78483955                 (source)
~                                                                ubuntu-trusty/3421.4
golang1.8/63a243be32451af083a062ba2c929c3f2b34f132               (source)
~                                                                ubuntu-trusty/3421.4

15 packages

Succeeded
```

Figure 3: BOSH allows software to run anywhere. This example shows a release of etcd software packages to run with the Ubuntu operating system.

BOSH Manifest

Manifests are declarative documents written in YAML. They define all components within the distributed system (including stemcells, releases, and configuration) that the operator intends to deploy and configure.

Additionally, the manifest requires that all specifications of the deployment be explicitly described (Figure 4), eliminating all guesswork. This makes the BOSH manifest a contract between the infrastructure and the operator.

```
pgpool:
  databases:
    - name: animals
      users:
        - ((admin_username))
      extensions:
      - pgcypto
  users:
    - username: ((admin_username))
      password: ((admin_password))
  backend:
    port: 6432
```

Figure 4: PostgreSQL server with configuration explicitly defined in a BOSH manifest.

CredHub

Managing manifest credentials is important! BOSH utilizes CredHub for a robust solution. Notice the manifest credentials in Figure 4 are shown as variables. When deploying, BOSH interpolates these variables with user-defined or generated credentials that are stored within CredHub. This ensures centralized and encrypted credential generation, storage, lifecycle management, and access.

BOSH Deployment

A deployment brings together stemcells, releases and a manifest into a single entity. Infrastructure (as virtual machines) is addressed via deployments and therefore specified in the manifest.

The BOSH documentation describes the deployment process well:

> *"The deployment process begins with deciding which Operating System images (stemcells) need to be used and which software (releases) need to be deployed, how to track persistent data while a cluster is updated and transformed, and how to automate steps of deploying images to an IaaS; this includes configuring machines to use said image, and placing correct software versions onto provisioned machines. BOSH builds upon previously introduced primitives (stemcells and releases) by providing a way to state an explicit combination of stemcells, releases, and operator-specified properties in a human readable file. This file is called a deployment manifest."*

Figure 5 shows a three-node MongoDB cluster. It includes Ubuntu 14.04 as the operating system, and MongoDB 3.4 configured into a replica set. The replica set configuration is specified in the manifest file, and helps deliver high availability.

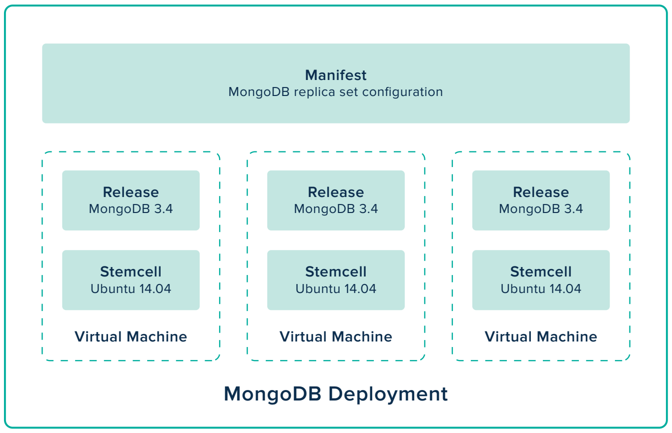
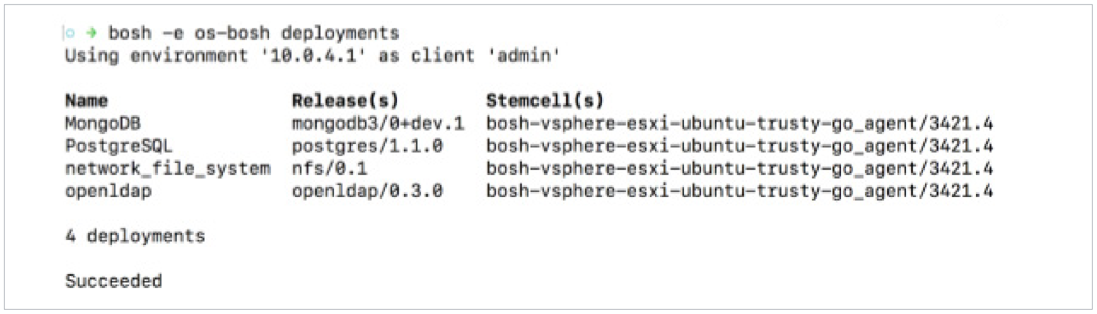


Figure 5: BOSH deployment of MongoDB replica set.

As operators get more familiar with BOSH, they can use it to deploy and manage many different kinds of systems, as seen in Figure 6. Use BOSH to deploy everything!



```
 → bosh -e os-bosh deployments
Using environment '10.0.4.1' as client 'admin'

Name                Release(s)         Stemcell(s)
MongoDB             mongodb3/0+dev.1   bosh-vsphere-esxi-ubuntu-trusty-go_agent/3421.4
PostgreSQL          postgres/1.1.0     bosh-vsphere-esxi-ubuntu-trusty-go_agent/3421.4
network_file_system nfs/0.1            bosh-vsphere-esxi-ubuntu-trusty-go_agent/3421.4
openldap            openldap/0.3.0     bosh-vsphere-esxi-ubuntu-trusty-go_agent/3421.4

4 deployments

Succeeded
```

Figure 6: Multiple systems running as BOSH deployments.

BOSH CPIs Helps You Survive (and Thrive) in a Multi-Cloud World

Devising a multi-cloud strategy is a top priority for nearly every business and IT executive. Let's examine why.

Cloud providers like Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure are evolving. Instead of differentiating on price, they are now differentiating on unique, value-added services further up the stack.

Multi-cloud strategies help the business to capitalize on these exciting application services. Further, this approach enables the business to drive usage to the most appropriate cloud provider based on the requirements of a given application.

If data services and machine learning are important, Google may be your preference. If you're a Microsoft shop, Azure might be your answer. And if you were an early mover, AWS might be your choice. Some workloads may remain on-premises with VMware for the next decade. With a multi-cloud strategy, a business can take advantage of the entire industry, and all that each dominant player has to offer.

At this point, you may be thinking “Managing one cloud provider is plenty difficult. Now you’re telling me to manage three or four?”

Once again, BOSH offers a useful abstraction. Gain the benefits of multi-cloud, and minimize complexity.

Cloud Provider Interfaces (CPIs) Deliver Simplicity and Specialization

Adopting a multi-cloud strategy isn’t easy. Without an experienced operations team to understand each provider, monitor each service, and put out fires, issues can pile up. But the right level of abstraction overcomes these pitfalls.

BOSH aims to provide the benefits of multi-cloud without the hassle. It’s a single tool that simplifies the deployment, monitoring, and updating of cloud infrastructure. Once a BOSH deployment is created, it’s automatically compatible with multiple clouds — no changes required! This property is “inherited” from BOSH itself. With the CPI abstraction (shown in Figure 7), BOSH users can now realize the benefits of each provider, **without** in-depth knowledge of each.

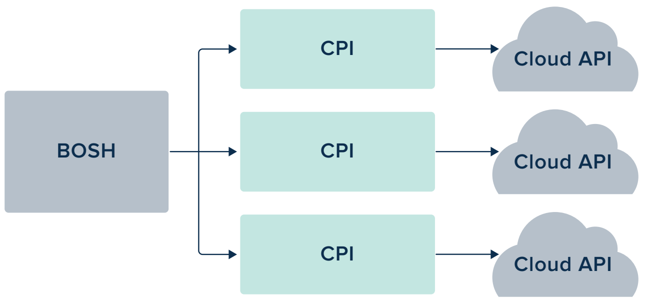


Figure 7: BOSH Cloud Provider Interfaces (CPIs) — a layer of abstraction that allows multi-cloud deployments

At the core of each CPI is a “contract.” This is implemented (and enforced) by each infrastructure provider. It bridges the uniformity of BOSH and the unique infrastructure APIs of each cloud. Each CPI contract must contain the mandatory functions shown in Figure 8.

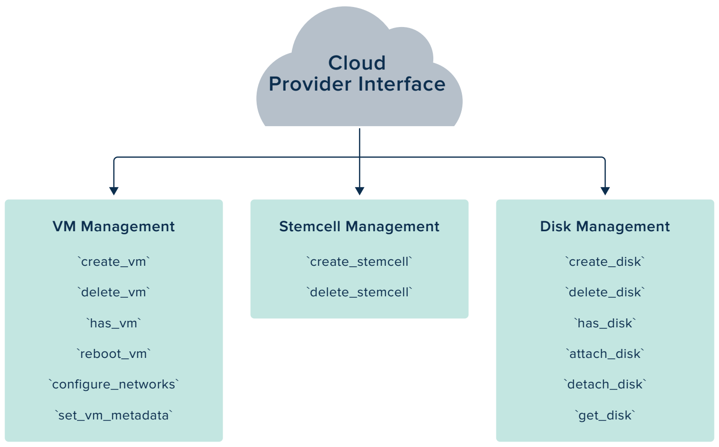


Figure 8: The BOSH CPI Contract allows for abstraction and customization by all cloud providers.

So BOSH supports multiple clouds. Now, let’s explore how it supports all those Day 2 questions we posted earlier.

BOSH and Day 2 Operations

The cost of developing a new app is dwarfed by the cost of running the application in production. Operators must monitor, upgrade, scale, and debug until the application is retired. This is "Day 2 Ops", and the business value of applications can only be realized if Day 2 Ops is successful.

Since BOSH has the responsibility of managing the entire stack (from the infrastructure to the application), it's designed to enable operators to be successful on Day 2.

Now, we'll dive into some of the key attributes of BOSH and help operators on Day 2 and beyond.

Immutable Infrastructure

When infrastructure requirements change, there is a critical decision to be made: modify or replace. BOSH was designed with the approach of automatically **replacing** infrastructure rather than **modifying** it. This concept is now popularly known as "Immutable Infrastructure".

Chad Fowler, who coined the term, explains the concept:

> *"If you absolutely know a system has been created via automation and never changed since the moment of creation, most of the problems...[with infrastructure variation] disappear. Need to upgrade? No problem. Build a new, upgraded system and throw the old one away. New app revision? Same thing. Build a server (or image) with a new revision and throw away the old ones."*

In addition, Randy Bias has a useful summation:

> *"In the old way of doing things, we treat our servers like pets, for example Bob the mail server. If Bob goes down, it's all hands on deck. The CEO can't get his email and it's the end of the world. In the new way, servers are numbered, like cattle in a herd. For example, www001 to www100. When one server goes down, it's taken out back, shot, and replaced on the line."*

The BOSH deployment model — specifically stemcells and releases — provide the automation required for Immutable Infrastructure.

The Canary

Everyone wants zero-downtime. But it's hard! Humans make mistakes, and mistakes lead to outages. Fortunately, BOSH has a solution.

Instead of applying changes to the entire cluster with a giant leap of faith, BOSH tests the changes by applying them to a subset of the cluster — the canary.

Canaries were once used to alert coal miners when harmful gases grew to a level that was toxic to animals and humans. If toxins such as carbon monoxide were present in the coal mine, the canary would perish before any human, providing a signal to leave immediately. As long as songs could be heard from the canary, the miners were safe.

Just as in the coal mine, canaries in BOSH indicate safe — or dangerous — conditions for the cluster. If the canary passes, then BOSH proceeds to apply the changes throughout the cluster.

Users can control the speed of the update by specifying a “max-in-flight” number or maximum number of VMs to update at once.

Cornelia Davis explains why canaries are important:

> *“And we goofed. Yes, we had tests for the tooling code. Yes, the tests were passing. But when we ran things with the production manifests as input, an authorization token was wrong. BOSH did tell us of the change, but we got a bit overzealous and didn’t catch that change until after we said ‘yes’ to the ‘are you sure you want to deploy this’ question. Once we realized our problem, BOSH was already upgrading our marketplace services broker. Doh. Could have been a very bad day. But, thanks to canaries, my heart didn’t even skip a beat. “*

Scaling with BOSH

Scaling is a good problem to have — it means the market is interested in your app! But how do you balance performance with infrastructure cost? Threading the needle to ensure a zippy customer experience while minimizing your infrastructure budget is tricky business. But it gets more complicated.

How do you scale when you have so many things running on a virtual machine? You can’t just blindly throw more compute resources at a sluggish app. You need to ensure consistency across code versions and configuration. Then there’s the infrastructure. Do you need to bring VMs down before you can add new ones? How do you expand disk size without bringing the host down?

Historically, operators have relied on “tribal knowledge” of an environment to ensure smooth, uneventful scaling of resources, and data migration. But when time-to-market counts, this approach falls short.

BOSH has an answer for this as well: just tweak the manifest to scale up, down and horizontally. Operators can easily bump the instance count, expand the disk size, and add memory — all through a single file.

Add-Ons and Extensibility

Agility and automation are wonderful. But they don’t mean much if you can’t also boost compliance and your security posture. Different industries require compliance with different sets of rules. Geographies have their own unique requirements. BOSH gives you the flexibility to customize your environment based on these factors, and many more. How? With add-ons!

BOSH adopters in financial services and healthcare industries look to add-ons like antivirus scanning, IPSec (for data encryption in transit), SSH Login Banners, and Monitoring Agents. IT executives (often with the help of their auditor) can apply add-ons as circumstances warrant. Just decide what's needed, and **`bosh deploy`** from the command line interface.

Enterprise Security

Traditional enterprise security preaches that “going slow reduces risk”. It depends on the idea that change inside a datacenter leads to unbounded risk.

The reality today turns this convention on its head. Savvy security teams are investigating ways they can **go faster to be more secure**. The ability to quickly apply patches, repave environments,

Pivotal

and rotate credentials can thwart many would-be attackers.

BOSH can help you go faster to be more secure by applying the Three "R's" of Security:

Repair vulnerable software as soon as updates are available.
BOSH releases enable software vendors to distribute updates quickly and for operators to apply those updates to the entire datacenter including production at any point with the use of canaries.

Repave servers and applications from a known good state, often.
With the use of BOSH stemcells for base OS images, repaving a server requires nothing other than uploading a new stemcell and **`bosh recreate deployment`**.

Rotate user credentials frequently, so they are only useful for short periods of time.
Reduce the risk from leaked credentials with CredHub. Use CredHub to rotate credentials, then run **`bosh deploy`**.

First Linux — and Now Windows!

BOSH initially targeted the Linux world, and it's now an essential component to countless open-source tools. The Windows and .NET world wanted these same benefits, and the BOSH community responded. BOSH now supports Windows environments too.

Cloud Foundry was once again the key driver.

With Microsoft joining the Cloud Foundry Foundation (the open source foundation that oversees BOSH), the integration between the Microsoft ecosystem and BOSH continues to strengthen. Windows BOSH stemcells based on are openly available across all cloud providers. What's more, Microsoft is continuing to invest with the Azure CPI and additional BOSH features in the Azure Service Broker.

By supporting both Windows and Linux we now change our focus to what developers want, abstractions.

The Four Abstractions

Quoting Stephen O'Grady "developers are the new kingmakers." They want choice in four key areas: data services, container platforms, application platforms, and serverless platforms. Developers want to use the right tool for the job. But where does that leave operations teams? How do they stay sane in this world, let alone security and compliance teams?

The key: providing the best set of developer choices through a single operational toolchain — BOSH! Here's how it works:

- **Application Platforms — help developers get a new feature into production as fast as possible. Drive in the cloud-native fast lane!**
  BOSH's foundation was built upon configuring, deploying, managing, scaling, and upgrading the most demanding application platform, Cloud Foundry.

- **Data Services — enable developers to store and manipulate data in the most efficient way.** BOSH now deploys data services like MySQL, RabbitMQ, Redis, and MongoDB. Enabling developers to use the right data store for the job has never been easier.
- **Container Platforms — give developers access to infrastructure primitives when needed.** Kubo, BOSH-Powered, Web-Scale Release Engineering for Kubernetes brings a uniform way to instantiate, deploy, and manage highly available Kubernetes clusters, on any cloud.
- **Serverless Platforms — allow developers to focus on the direct result they desire.** Cloud Foundry and Spring Cloud Functions both backed by BOSH, provide the first cloud native operating system. Instead of writing code that talks to APIs, serverless code is built to interface with cloud services offered directly through BOSH.

The Key to Unlocking BOSH

Given the power of BOSH described in this paper, you may be wondering how it impacts your organization.

Since BOSH automates tasks and improves productivity, the operations team becomes the **key** to an efficient organization.

Gone are the days of fiddling with software patches and upgrades. Don't waste time wrangling a sprawling infrastructure footprint. Instead, focus on becoming a cloud-native operations team and driving down the number of pager alerts with BOSH automatization. Organizations are once again free to concentrate on the bigger picture, while BOSH eliminates the routine tasks that no operations team should be forced to deal with.

Conclusion

Enterprises that have committed to a software-led digital transformation are counting on five key benefits. BOSH has a role to play in all of them:

- **Speed — "I need to deliver software faster to compete and differentiate."**
  BOSH provides an automated way to easily create software releases and update complex distributed systems with simple workflows.
- **Stability — "I need to deliver enterprise uptime for my apps."**
  BOSH includes health checks, resurrection, and blue/green deployments to keep systems online and available.
- **Scalability — "I need to handle elastic, and unpredictable amounts of traffic".**
  Focused on Day 2 operations, BOSH is flexible enough to work with a single service, a single VM, a single provider to multiple services, thousands of 1000s of VMs, and multiple cloud providers.
- **Savings — "I need to optimize by infrastructure spend and lower legacy software licensing if possible."**
  Managing infrastructure, software releases, OS images, and system configuration with BOSH reduces IT operational capital and operational expenses.
- **Security — "I need to improve my security posture across all IT systems."**
  Changing from a "going slow reduces risk" mindset to a continuous "repair, rotate, and repave" mindset, BOSH greatly reduces datacenter vulnerabilities.

As you consider the universe of operational tools to power your enterprise for the next decade, get to know BOSH!

Recommended Reading

- BOSH Tutorial
- Meet Kubo. BOSH-Powered, Web-Scale Release Engineering for Kubernetes
- Building Cloud Foundry On-Demand Services Just Got a Lot Easier
- The Three Rs of Enterprise Security: Rotate, Repave, and Repair
- Canaries are Great!
- Immutable Infrastructure, Windows, and BOSH
- How We Harden a Cloud Foundry Stemcell (So You Don't Have to)
- A Primer on PCI Compliance with Pivotal Cloud Foundry
- BOSH Turns Five!
- BOSH — Twenty Years of Deployment Lessons in One Tool
- Living in a Multi-Cloud World: Bosh, Kubo and the Open Service Broker API
- Automated Ops; Freedom To Innovate (Part 2)

Appendix A:

Terminology Quick Reference Guide

Add-Ons

Operators typically want to ensure that certain software runs on all VMs managed by the Director. Examples of such software are: security agents like Tripwire, IPsec, etc. anti-viruses like McAfee custom health monitoring agents like Datadog logging agents like Loggregator's Metron. These addons are added as a release job that is then collocated on all VMs managed by the Director.

Cloud Config

A YAML file that defines infrastructure specific configuration used by the Director and all deployments. It allows operators to separate infrastructure specific configuration into its own file and keep deployment manifests infrastructure agnostic.

CPI (Cloud Provider Interface)

An API that the Director uses to interact with an infrastructure to create and manage stemcells, VMs, and disks. A CPI abstracts infrastructure differences from the rest of BOSH.

CredHub

Service deployed along side BOSH to manages credentials like passwords, certificates, certificate authorities, ssh keys, rsa keys and arbitrary values (strings and JSON blobs). CredHub provides a CLI and API to get, set, generate and securely store such credentials.

Deployment

A collection of referenced stemcells, releases, and a deployment manifest that is uploaded to the Director.

Director

The core orchestrating component in BOSH. The Director controls VM creation and deployment, as well as other software and service lifecycle events. The Director creates actionable tasks:

- By translating commands sent by an operator through the CLI
- From scheduled processes like backups or snapshots
- If needed to reconcile the expected state with the actual state of a VM

Once created, the Director adds these tasks to the Task Queue. Worker processes take tasks from the Task Queue and act on them.

Kubo

A BOSH release for Kubernetes. It provides a solution for deploying and managing Kubernetes with BOSH.

Manifest

A YAML file that defines the components and properties of the deployment. When an operator initiates a new deployment using the CLI, the Director receives a version of the deployment manifest and creates a new deployment using this manifest.

Release
A versioned collection of configuration properties, configuration templates, start up scripts, source code, binary artifacts, and anything else required to build and deploy software in a reproducible way. A release is the layer placed on top of a stemcell. They are self-contained and provide very specific software for the purpose of that release.

Resurrector
A plugin that's responsible for automatically recreating VMs that become inaccessible. The Resurrector continuously cross-references VMs expected to be running against the VMs that are sending heartbeats. When resurrector does not receive heartbeats for a VM for a certain period of time, it will kick off a task on the Director (scan and fix task) to try to "resurrect" that VM.

Runtime Config
BOSH has a way to specify global configuration for all VMs in all deployments. The runtime config is a YAML file that defines infrastructure agnostic configuration that applies to all deployments.

Stemcell
A versioned Operating System image wrapped with infrastructure specific packaging. A typical stemcell contains a bare minimum OS skeleton with a few common utilities pre-installed, a BOSH Agent, and a few configuration files to securely configure the OS by default. BOSH uses stemcells to perform release and lifecycle management.

UAA
A multi tenant identity management service, used by BOSH, but also available as a stand alone OAuth2 server. It's primary role is as an OAuth2 provider, issuing tokens for clients to use when using BOSH.

Appendix B:

Deploying Postgres on vSphere using BOSH

In this section, we'll use BOSH to deploy Postgres Server on vSphere.

To get started, you will need the BOSH Command Line Interface (CLI). The Command Line Interface (CLI) is the primary operator interface to BOSH. We will use the CLI to deploy the BOSH Director. The Director is the core orchestrating component in BOSH. The Director controls VM creation and deployment, as well as other software and service lifecycle events.

Install the BOSH CLI

Refer to https://bosh.io/docs/cli-v2#install for Operating System specific installation documentation.

Get the BOSH configuration files

Clone the BOSH deployment repository from Github into a directory on your machine.

```
$ git clone https://github.com/cloudfoundry/bosh-deployment ~/workspace/
bosh-deployment
```

Create a directory working directory

Prepare deployment by creating a new directory to keep the BOSH Director deployment state files.

```
$ mkdir -p ~/deployments/bosh-1
$ cd ~/deployments/bosh-1
```

Deploy a BOSH Director

Deploy the BOSH Director by specifying your vSphere credentials and network information. This creates or updates the specified deployment according to the provided manifest. In the case of other IaaS targets (i.e. AWS or Openstack), refer to the documentation on BOSH deployment repository cloned above.

```
$ bosh create-env ~/workspace/bosh-deployment/bosh.yml \
  -o ~/workspace/bosh-deployment/vsphere/cpi.yml \
  --state=$vars_store_prefix \
  --vars-store ./creds.yml \
  -v director_name=whitepaper \
  -v internal_cidr=.. \
  -v internal_gw=.. \
  -v internal_ip=.. \
  -v network_name=.. \
  -v vcenter_dc=.. \
  -v vcenter_ds=.. \
  -v vcenter_ip=.. \
  -v vcenter_user=.. \
  -v vcenter_password=.. \
  -v vcenter_templates=.. \
  -v vcenter_vms=.. \
  -v vcenter_disks=.. \
  -v vcenter_cluster=..
```

Alias deployed Director

Assign a name to the created environment for easier access in subsequent CLI commands. Instead of specifying a Director location and possibly a CA certificate, subsequent commands can just take given name via --environment flag (-e).

```
$ bosh -e internal_ip --ca-cert <(bosh int ./creds.yml --path /director_
ssl/ca) alias-env bosh-1
```

Log in

Log into the BOSH Director non-interactively, alternatively the login CLI command could be used. NOTE: The non-interactive login procedure is used when running BOSH commands from shell scripts.

```
$ export BOSH_CLIENT=admin
$ export BOSH_CLIENT_SECRET=`bosh int ./creds.yml --path /admin_password`
```

Update cloud config

Update the current cloud-config on the BOSH Director. The cloud config defines IaaS specific configuration used by the Director and all deployments.

```
$ bosh -e bosh-1 update-cloud-config ~/workspace/bosh-deployment/vsphere/
cloud-config.yml \
  -v vcenter_cluster=.. \
  -v internal_cidr=.. \
  -v internal_gw=.. \
  -v network_name=..
```

Upload specific stemcell

Upload a centos-7 stemcell to the Director. The Postgres server will be deployed on CentOS 7 virtual machine created from this stemcell.

```
$ bosh -e bosh-1 upload-stemcell https://s3.amazonaws.com/bosh-core-
stemcells/vsphere/bosh-stemcell-3421.11-vsphere-esxi-centos-7-go_agent.
tgz
```

Clone the Postgres repository

Clone the Postgres release repository from GitHub into a directory on your machine.

```
$ git clone https://github.com/cloudfoundry/postgres-release
```

Upload Postgres BOSH release

Upload the Postgres release to the Director. For a deployment to succeed, all necessary releases must be uploaded to the Director.

```
$ cd postgres-release
$ bosh -e bosh-1 upload-release
```

Create operations file

In your favorite editor, create a file called postgres-ops.yml with the following contents to customize the deployment. In this example, we customize the VM type and the disk type.

```
- type: replace
  path: /instance_groups/name=postgres/persistent_disk_type
  value: large

- type: replace
  path: /instance_groups/name=postgres/vm_type
  value: large
```

Generate deployment manifest

Generate a deployment manifest using the postgres-ops.yml file from the previous step. The manifest will define the components and properties of the Postgres deployment.

```
$ cd postgres-release
$ bosh interpolate templates/v2/postgres.yml
   --ops-file=templates/v2/operations/set_properties.yml
   --ops-file=postgres-ops.yml > postgres.yml
```

Deploy Postgres

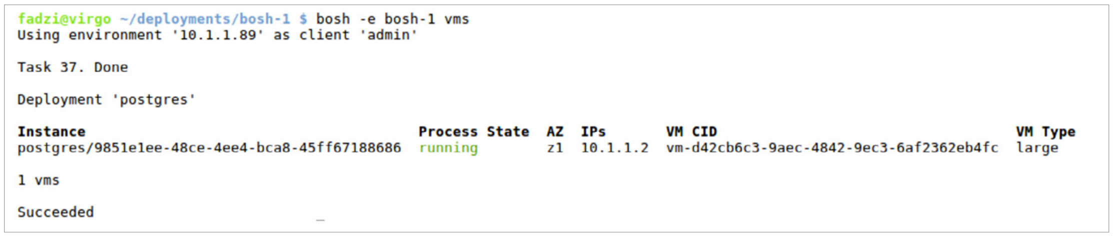
Now execute the BOSH Deploy command for the Postgres database. A virtual machine will be created, and assigned an IP address. The Postgres service will be installed and started.

```
$ bosh -e bosh-1 -d postgres deploy postgres.yml --vars-store ./creds.yml
```

Quick Test

After deployment is completed, verify that a new virtual machine has been successfully created in the specified AZ. Also take note of the IP address.

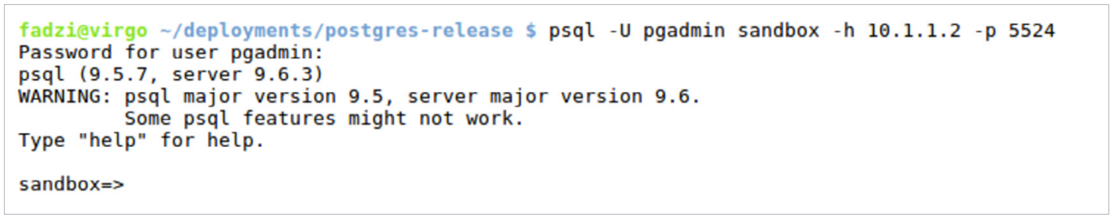
```
$ bosh -e bosh-1 vms
```



```
fadzi@virgo ~/deployments/bosh-1 $ bosh -e bosh-1 vms
Using environment '10.1.1.89' as client 'admin'

Task 37. Done

Deployment 'postgres'

Instance                                      Process State  AZ  IPs       VM CID                                    VM Type
postgres/9851e1ee-48ce-4ee4-bca8-45ff67188686  running        z1  10.1.1.2  vm-d42cb6c3-9aec-4842-9ec3-6af2362eb4fc  large

1 vms

Succeeded
```

Connect to postgres

Open the generated creds.yml file and find the password to login



```
fadzi@virgo ~/deployments/postgres-release $ psql -U pgadmin sandbox -h 10.1.1.2 -p 5524
Password for user pgadmin:
psql (9.5.7, server 9.6.3)
WARNING: psql major version 9.5, server major version 9.6.
         Some psql features might not work.
Type "help" for help.

sandbox=>
```

Congratulations you have BOSH deployed Postgres database!

Appendix C:

Learn from Your Peers - How the Best Companies Use BOSH

Ford Motor Company

Ford established an entirely new business unit to capitalize on cloud native applications - Ford Smart Mobility, LLC. This organization is designed to operate like a software startup, investing in and developing services for emerging opportunities in the $5.4 trillion dollar transportation services market. Ford utilizes BOSH with the Microsoft Azure cloud to deploy and manage RabbitMQ, MySQL, Spring Cloud Services, Pivotal GemFire, HA Proxy, Vault, GitLab Enterprise, Concourse and Ops Metrics. Together these systems power Ford's new FordPass application, delivering business results backed with BOSH.

Financial Services: Secure Hybrid Banking Reference Architectures for Cloud-Native Applications

Running IT for a bank is not for the faint of heart. See how JPMorgan Chase and other leading banks are using Cloud Foundry and BOSH to become digital innovators.

Pivotal's engineers and architects have worked with seven of the top banks in recent years. Our team has refined "best practices" for how to best deploy PCF, and how to best develop modern apps. This insight helps customers reduce risk and deliver high-quality software faster.

Our learnings are encapsulated in this white paper as reference architectures. These designs help banks deliver software continuously, in a secure and scalable way. Read the whitepaper here.