Containerizing Cloud Foundry

- Preface
- Goals & Non-Goals
- Containerizing BOSH Deployments
  - Building Base Images from BOSH Stemcells
  - Building Images from BOSH Releases
  - Rendering Templates
  - Running Containers
  - Changes needed in BOSH & BOSH Releases
- Runtime
  - Scheduler Requirements
  - A Kubernetes Implementation
    - Lifecycle of a Containerized BOSH Job
- Fissile to evolve towards the containerization proposal
  - Why
  - How
    - Phase 1 - The Start
    - Phase 2 - The Move to BPM
    - Phase 3 - Operator at Runtime
  - FAQ
- Open Points
- Appendix
  - Intro to BOSH Concepts
    - BOSH Releases vs BOSH Runtime
    - BOSH Stemcell
    - BOSH Release
    - BOSH Process Manager (BPM)
    - BOSH DNS
    - BOSH Links & BOSH Links API
    - Config Server API & CredHub
    - BOSH Deployment Manifest
    - BOSH Orchestration
  - Considerations around a BOSH Kubernetes Cloud Provider Interface (CPI)

Preface

With this proposal we want to lay the groundwork to enable deploying BOSH Releases[1] to container schedulers like Kubernetes as "just another" supported deployment target (for BOSH Releases in general and Cloud Foundry in particular).

As shown in prior work (see appendix) the existing extension point for underlying platforms, namely the BOSH Cloud Provider Interface (CPI), does not allow a native integration while retaining the full capabilities of container schedulers. That is why we propose to take BOSH Releases as-is, but replace the orchestration part by something more tailored towards container schedulers. Most of the heavy duty can actually be delegated to the container scheduler.

In order to really have container schedulers as an equally supported option, we propose to integrate our work into the Continuous Integration (CI) of BOSH Releases. As an example, we will create a tool to build container images for BOSH Releases. Those images should be built in CI and published as official Cloud Foundry images[2].

Goals & Non-Goals

We propose to enable the deployment of BOSH Releases to container schedulers (in particular Kubernetes, but the concepts described here should make it possible to support Docker Swarm, Mesos or others as well) using the rich feature set these schedulers provide. If this requires changes to how BOSH Releases are described or built we will try to introduce them in a way that is also beneficial for the non-container world.

We propose to make containerized Cloud Foundry distributions first-class citizens to the community:

- Base images should be built when BOSH Stemcells are built (this could even happen in the same pipeline).
- CF projects should build container images in their CI. This should be made as easy as possible. We would implement Concourse resources and/or cmdline tools to facilitate this.
- The capability to containerize releases should be implemented in the BOSH CLI. Given that BOSH users are accustomed to using the BOSH CLI for interacting with all BOSH capabilities, it would be a natural fit for containerization commands to be included here.

---

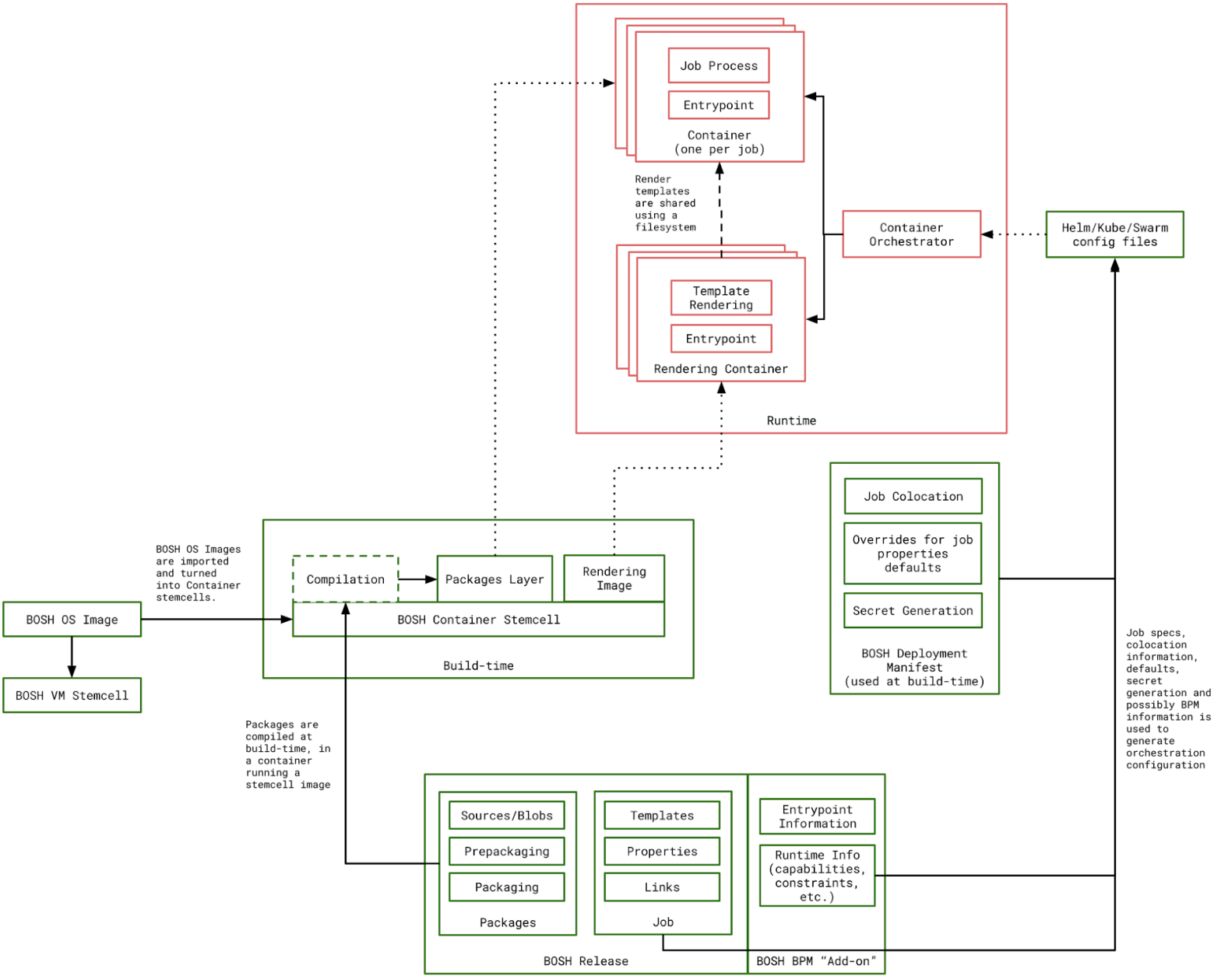
[1] In this document we're going to make a distinction between BOSH Releases (i.e. organizing source code and packages software using BOSH) and BOSH Orchestration (i.e. deploying BOSH Releases using the BOSH Director).

[2] This approach is similar to OpenStack's Kolla.

It is **not a goal** of this proposal to replace BOSH orchestration for classical virtual machine-based infrastructure.

Containerizing BOSH Deployments

There are a few steps we need to take in order to transform a BOSH Deployment into a running system using a BOSH Stemcell and a set of BOSH Releases:

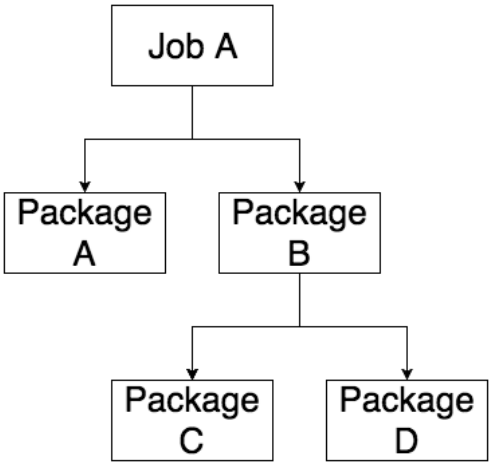


Building Base Images from BOSH Stemcells

Base container images can be built from BOSH Stemcell base OS images produced by `bundle exec rake stemcell:build_os_image` as shown in an example pipeline[3]. This should be integrated with bosh-linux-stemcell-builder pipelines.

Building Images from BOSH Releases

BOSH Releases contain all the metadata we need in order to build container images. Jobs and packages each have a `spec` file that describes — amongst other things — the package dependencies. The bosh-cli repository contains Golang packages to read those metadata and compile packages. We will have to make sure to respect the way BOSH places packages for jobs, and not include dependencies that are not required (see Package C and D below)[4]:



Rendering Templates

Besides packages, BOSH jobs typically provide a number of ERB templates for configuration and/or lifecycle hook scripts. We can render those templates re-using the bosh-template gem that is also used by BOSH[5]. There are basically three sources of values (in order of precedence):

1. User provided values in the deployment manifest.

---

[3] The resulting image will contain utilities (like gcc) that will be removed in most productive CF deployments via BOSH Director property `director.remove_dev_tools`. In the future we should consider separating build time from runtime base image.

[4] Packages C and D are not required at runtime. We should have a resulting image that only contains the direct package dependencies for Job A, i.e. packages A and B. Conceptually multistage builds sound like the right thing to do. moby/buildkit looks like a good starting point to implement building based on BOSH metadata.

[5] http://bosh.io/docs/job-templates.html

2. Defaults provided by the release spec file.
3. Hard-coded values as second parameter to `p()` calls.

Template rendering will be performed by a container image that contains the aforementioned gem capabilities[6]. This container image will be run as a precursor to the actual job containers, rendering the ERB templates into a shared file system.

The container image will require the following inputs in order to render templates:

- Job spec (for property defaults).
- Properties section of the Deployment Manifest.
- Runtime environment of the init container (`spec.*` BOSH primitives).

Running Containers

In order to run a Containerized BOSH Deployment, we need the software (images), container configuration, rendered templates, persistent volumes and connectivity information.
All of the required information is provided by BOSH Releases, BOSH deployment manifests (and any other additional BOSH configuration that may be needed) and should be computed using the following strategy:

- Software images should be based on container stemcell images and pre-compiled packages.
- Container configuration such as an entrypoint, shared volumes, sizing and capability information should be derived from the BPM information.
- Persistent volumes should be derived from the BOSH deployment manifest.
- Exposed ports should be derived from the BOSH Links.

Changes needed in BOSH & BOSH Releases

The goal for these changes is for BOSH Release descriptions to be sufficient so that they can be transformed into artifacts that can be deployed/managed by other mechanisms.

- `bosh-linux-stemcell-builder` should produce container base images.
- BOSH Links should include a list of port ranges as a mandatory field.
- BOSH Releases should provide complete and sufficient BPM information and should not rely on monit features as described:
    - ‘check’ (‘with pidfile’, ‘start’, ‘stop’ and ‘group’).
    - The following features should be supported using BPM:
        - Health description (to replace `if failed host <host> <port> protocol http`).
        - Memory constraints (to replace `if totalmem > <n> MB for 15 cycles, then restart`).
        - Process dependencies (to replace `depends on <process name>`).

[6] Fissile uses configgin for rendering templates.

        - Optional processes and process counts (to replace ERB control structures that turn components on/off, or create copies).
- BOSH Releases should provide bpm.yml.erb files that can be rendered at build time:
    - E.g. they **must not** use `spec.*` properties, because that information is only available at runtime.

Runtime

Scheduler Requirements

There are certain requirements a scheduler needs to fulfill to be able to be a deployment target for the artifacts created by this proposal:

- Ability to co-locate containers in the same networking namespace.
- Ability for containers to share a filesystem.

A Kubernetes Implementation

We propose Kubernetes as a first target implementation for transforming BOSH Releases. Given BOSH Container Images created in the manner described in the previous section, we are tasked with creating a process that can provide us with the configuration required to deploy said images on Kubernetes.

Template rendering will be done using Init Containers. Each Init Container will consume information (job spec, deployment manifest, templates) using ConfigMaps and Secrets.

The equivalent of a BOSH Instance Group will be a StatefulSet or Deployment. Each BOSH Job will be started as a Container inside a Pod.

We will use Kubernetes Jobs to run the equivalent of non co-located BOSH Errands.

Information for BOSH Links is transformed into Kubernetes Services.

Lifecycle of a Containerized BOSH Job

1. Template rendering:
    - Will be done by running Init Containers using a specialized container image. See Rendering Templates for more details.
2. BOSH Job pre-start:
    - These should be run in their own init containers.
3. BOSH Job post-start:

    - post-start for BOSH Jobs are different than Kubernetes PostStart hooks. BOSH will run the post-start script once "monit successfully starts a process"[7].
    - Given a BOSH Job that has a post-start template, we could run the script once the following condition is true: the set of services created for the job is ready (e.g. services can be resolved via DNS).
4. BOSH Job post-deploy:
    - Given a BOSH Job that has a post-deploy template, we assume based on the BOSH docs that it's idempotent, so we can run it as a Kubernetes Job until successful completion.
5. BOSH Job drain:
    - Draining should be performed by a Kubernetes PreStop hook.
6. BOSH Job post-stop:
    - Post stop hooks may still be performed using a Kubernetes PreStop hook.

Fissile to evolve towards the containerization proposal

Recently, we've noticed that every step we take in improving fissile takes us towards the direction of this containerization proposal.
This includes:

- A feature to support "colocated containers" - multiple containers in a Kubernetes pod, each running a set of BOSH jobs:
    - https://github.com/SUSE/fissile/pull/352
- The new approach for pod management - uses stateful sets for all roles, resulting in more fidelity with BOSH lifecycles:
    - https://github.com/SUSE/fissile/pull/354
    - https://github.com/SUSE/fissile/wiki/Pod-Management-using-Role-Manifest-Tags
- Secret management - the new implementation manages secret variables using Kubernetes Secrets and has a runtime component:
    - https://github.com/SUSE/fissile/pull/338
    - https://github.com/SUSE/fissile/wiki/Helm-Secret-Management
- Plans to gradually deprecate the Role Manifest format in lieu of the BOSH deployment manifest format.

Why

1. There are developers actively supporting fissile that can take us in that direction.

[7] https://bosh.io/docs/post-start.html

It's a shame to duplicate effort and to work on two different projects that have the same goals.
2. There are customers with deployments of fissile-based distros that need support. This will help our migration stories and ensure a steady and smooth progress to the ideal scenario.
3. We want fissile to be part of the BOSH/Extensions PMC. We want our work to contribute to the ecosystem.
4. Fissile-based distros work and are certified by the CFF. We are ready to demo this and open this style of running Cloud Foundry to the community.

How

The transition can happen in 3 phases, and at the end of each of these, fissile will get closer to the proposal.

Phase 1 - The Start

- Using the colocated containers implementation, we will first start splitting off the jobs that repeat, such as:
    - metron-agent
    - global-properties
- We will reduce the "outside-BOSH" fissile functionality:
    - Role hooks (to be replaced with BOSH jobs and BOSH hooks).
    - Remove the "processes" key in the run information from the Role Manifest.
- Move properties in the fissile role manifest to be job-namespaced.
- Implement a "no-monit" tag for fissile roles. This allows a container to run without the need for monit. It will use information derived from BPM or information added to the Role Manifest to generate the correct entrypoint.

Phase 2 - The Move to BPM

- Reduce "outside-BOSH" fissile functionality further:
    - The "run" key from roles should be removed in favor of BPM and link information.
- Modify Fissile so that each job gets its own container (1 process per container). The "colocated-container" tag can be removed.
- Add enough information so that all jobs contain enough entrypoint information using BPM. This also means monit should no longer be required, thence removed. The "no-monit" tag should be deprecated, as "monit-less" becomes default behavior.
- Add support for 1-to-1 configuration without fissile templating. This will require changes to configgin so that a deployment manifest file (instead of environment variables) can be used to render templates, instead of environment variables.

Phase 3 - Operator at Runtime

- Full support for configuration via BOSH deployment manifests instead of a Role Manifest.
- Rendering of configurations happens in a separate container that creates config maps.
- Fissile can create and manage kube objects as an operator at runtime, given a BOSH deployment manifest.
- BOSH Releases are individually turned into Docker images that can be referenced from a BOSH deployment manifest. These would contain compiled packages and job sources.

FAQ

Isn't it better to implement the proposal from scratch?
We don't think so. We have multiple teams with experience in running containerized Cloud Foundry and with this approach everyone will work towards the same goal.

How long will it take until I can run a containerized version of Cloud Foundry?
You can run one now. It won't run in accordance with this proposal, but we want to make it so. This is one of the reasons we want to proceed with this plan. The community will have access to a working containerized Cloud Foundry all the way.

How long will it take to evolve fissile in the manner described?
6-12 months.

Following this plan, will it be possible to keep updating a live-system, initially deployed with today's fissile and end up with the state described by the containerizing CF proposal?
Yes!

What's the most difficult piece of this journey?
We believe it's going to be the move from monit to BPM. The necessary and sufficient condition is that all releases contain BPM sufficient information so that we can use it (something we'll need to contribute). There is no need for cf-deployment to move fully to BPM though. We could use BPM while cf-deployment still uses monit.

Why is fissile so different from the proposal?
Fissile was created a few years ago, when deployment manifests looked different, and there was no effort like BPM. The Kubernetes landscape was different as well. There were no

StatefulSets or Operators. BOSH 2.0 is more amenable to what we're trying to do with this proposal.

I don't like the name "fissile". Can we change it?
Sure.

From a config management perspective, the roles,opinions,dark-opinions will become obsolete in favor of a manifest.yml. Would a cloud-config.yml also be required?
Correct - all the fissile-specific config files will be deprecated in favor of deployment manifests. It's possible that a cloud config could be used to also configure kube parameters like the storage class.

Open Points

- How can I see a diff of what I am going to deploy?
- How do we handle backup & restore of a CF deployment?
- How are credentials generated?
- BOSH deployment options
    - How do we support canary deployments?
        - StatefulSet rolling update only provides primitives, i.e. gradually change the value for `partition`.
    - What about `max_in_flight`
    - What about `serial`
- How to support BOSH Links API
- How do we provide logs?
- Collocated BOSH errands

Appendix

Intro to BOSH Concepts

This section provides a recap on existing BOSH concepts since they are relevant for the Containerizing Cloud Foundry discussion. You can skip to the next chapter in case you are already familiar with those concepts.

BOSH Releases vs BOSH Runtime

By **BOSH Releases** we refer to the entire set of information describing how to package and run distributed software that's available at build-time:

- job definitions

- package definitions
- deployment manifests
- stemcells
- bpm.yml definitions

By **BOSH Runtime** we refer to the active components of BOSH, that actively manage deployments at runtime:

- director
- agent
- BPM release
- BOSH DNS Release

BOSH Stemcell

A base operating system image with secure configuration, some common utilities and a BOSH Agent. BOSH currently provides stemcells based on Ubuntu Trusty, CentOS 6.5 and SUSE Linux.

BOSH Release

A BOSH Release consists of jobs and packages. Jobs include metadata about required packages (runtime dependencies), configuration properties and links (e.g. properties provided to or consumed from other jobs) and templates for configuration, control scripts (for start and stop) and lifecycle hook scripts (pre-start, post-start, post-deploy and drain).

Packages include metadata about required packages (compile-time dependencies), source and/or binary blobs, an optional packaging script and a script to compile the package.

BOSH Process Manager (BPM)

BPM is a release providing a collocated job that runs any BOSH job in a separate container. BOSH Releases using it specify a bpm.yml template instead of the usual control script. The bpm.yml specifies executables with parameters, limits (memory, etc.), volumes, capabilities, lifecycle hooks, etc. In short, everything required to run the job in a container.
For now, starting the job in a container is done by BPM, which is triggered and controlled by monit.

BOSH DNS

BOSH will distribute DNS records to the agents. These will be interpreted and provided via a local DNS server with an additional co-located bosh-dns-release.
This will provide a couple of features to BOSH managed software:

- client side load balancing
  `*.database-z1.diego1.cf-cfapps-io2-diego.bosh`

- DNS aliases
  BOSH deployment operators will be able to configure alias DNS records like `bbs.service.cf.internal` → `*.database-z1.diego1.cf-cfapps-io2-diego.bosh`
- health based DNS aliases

BOSH Links & BOSH Links API

With BOSH links releases can 'provide' and 'consume' weakly typed sets of properties.
There are properties provided by default:

- name [String, non-empty]: Instance name as configured in the deployment manifest.
- id [String, non-empty]: Unique ID.
- index [Integer, non-empty]: Unique numeric index. May have gaps.
- az [String or null, non-empty]: AZ associated with the instance.
- address [String, non-empty]: IPv4, IPv6 or DNS address. See Native DNS Support for more details.
- bootstrap [Boolean]: True if the instance is the first instance of its group.

In addition, releases can expose any of their properties, typically ports, usernames and secrets.

Config Server API & CredHub

With the config server API, BOSH introduced variable interpolation and generation capabilities. When the BOSH director discovers a not yet interpolated variable it contacts the configured config server to resolve (and potentially generate) values. The CredHub project is the most prominent implementation of the config server API.
It is likely that more components in the Cloud Foundry ecosystem (in particular service brokers) will get a runtime dependency. An example here is management of credentials which is being outsourced to CredHub.

BOSH Deployment Manifest

A BOSH Deployment Manifest (together with other configuration sources; e.g. cloud and runtime config) describes the desired state of a distributed system. I.e. which BOSH Jobs to deploy, whether to co-locate or separate certain jobs and how they should be configured.

BOSH Orchestration

In order to materialize the desired state BOSH does the following things:

- Present a diff of the old (if any) and new desired state.
- Compile packages based on the given stemcell version.
- Render job templates:
  - Provide properties and links in the rendering context.
  - [Optional] Delegate creation and storage of credentials to a config server (e.g. CredHub).

- Delegate creation of compute nodes and disks to Cloud Provider Interface (CPI).
- Delegate installation of required packages and distribution of rendered templates to the BOSH Agent.
- Remove software that was only required for compilation (e.g. gcc).
- Deploy/update a canary first in order to verify new versions can be installed.
- Check on a regular basis if actual state and current state match and take means to converge the actual state to the desired state if they don't (resurrection).

Considerations around a BOSH Kubernetes Cloud Provider Interface (CPI)

There have been a few approaches to implementing a BOSH Kubernetes CPI:

- https://github.com/cfibmers/kubernetes-cpi
- https://github.com/SAP/bosh-kubernetes-cpi-release
- https://github.com/bosh-cpis/bosh-kubernetes-cpi-release

The biggest problem that all approaches revealed is that nesting containers is a fragile venture at best.
When either CF Garden or Kubernetes changes its overlay file system, a CPI that creates nested file systems might break if the particular combination does not support nesting.

Problems however don't stop there:

1. **Declarative vs. imperative**
   While BOSH from 10.000 feet is a declarative approach to configuration management, the implementation between the BOSH director and the CPI (+ VM agent) is imperative. "Create a VM, then create a disk, then attach the disk, then render templates and push to VMs, then start processes one-by-one, ..."
   In contrast, Kubernetes expects a declarative description for a Pod, i.e. "run me a Pod, that has the following volumes mounted and runs the following containers".
   With Pods being immutable the mapping of BOSH VMs to Kubernetes Pods does not really work out.
2. **Manual vs. dynamic IP**
   When we start to workaround the issue from #1 by e.g. re-creating a Pod with only a volume added we run into the next problem.
   BOSH assumes IPs not to change during the lifetime of a VM. However, Kubernetes gives no control over a Pod's IP.
3. **Stemcell vs. fully defined image**
   BOSH creates VMs from a stemcell image and then installs software required to fulfil specific roles in a deployment.
   In contrast, Kubernetes expects the container images to be immutable. If you want to benefit from higher-level constructs of Kubernetes, like a StatefulSet, Kubernetes will reschedule a Pod with an empty stemcell.

Some of the problems that need to be solved depend on the approach you choose within a continuum between BOSH native and Kubernetes native. Most of the problems can probably be worked around. However, in order to get to a solution that you are confident to recommend for productive use you have to take sides. Either you change BOSH to support using the rich set of features Kubernetes provides for running software, or you reduce Kubernetes to not more than a primitive IaaS. No matter what side you choose, either of the questions will arise "Why Kubernetes?" or "Why BOSH?".