Build Foundry

A Registry System for Cloud Foundry

Cloud Foundry has had exponential growth and, as the project has grown, some base fundamentals have not grown to the same degree. This project is to enable public and private environments to easily deploy build packs / containers at runtime, deploy the necessary services to Cloud Foundry, and transform a dispersed ecosystem to allow centralization.

The Problem

Here is a list of the current issues with deployment and service discovery that this project will work to resolve:

- There are 45+ buildpacks in the community and only the core build packs are maintained.
- All current core buildpacks are part of the Cloud Foundry (CF) BOSH deployment process. If customers do not upgrade CF then their applications are running with legacy build packs that could have security or performance issues.
- There is currently no rating system for build packs that allows users to know the success/failure percentage and usage.
- Customers requiring CA certification of build packs are building one-offs, which is causing more separation from the core. By having a system that can handle the CA requirements via deployment, it will fix the issue.
- Services have no centralized registry to register with and be discoverable.
- Services have no rating system to know which ones are used and how often.

The Solution

The solution is to provide a open source / community-driven registry system that allows for a public repository of services and buildpacks. It should also enable companies, which have the ability to control their own mirror of this with a process, to maintain CA and control over what is used in their environment.

Platform Architecture

The platform architecture is based on Node.js as it is a web service with a web interface. Members are able to authenticate using their github or other authentication methods using auth0's authentication service.

The service contains a registry, web interface, and CLI for publishing and maintaining these services.

Registry maintains all buildpacks and services with how many times they are deployed, failure rates, success rates, rating, comments, how to access, and so on - as with NPM.

The web interface will be built in react and be a full module that interacts with the API layer, allowing it to be fully customizable for vendors if needed.

The CLI will be a plugin for cf-cli and allow for publishing, lookup, and also the deployment of buildpack and services via the CLI. This gives the standard cf-cli super powers.

The API consists of the following core objects:

- Profiles: profiles have their standard information and links to what they have published but also relative information such as organizations and roles, their gravatar and more.
- Organizations: Organizations allows you to setup oauth (if you are running a local foundry) and also allow for publishing like an individual, but with a team involved.
- Package: Package will control the rating, comments, author's, success / failure, download rates, and organization of services or buildpacks as a type.

Server Architecture

Server architecture is based on using Cloud Foundry to host the platform. What is different with this is that the caching nodes will locally cache packages, such as buildpacks and services. This will scale using Cloud Foundry but you will also need to allow a CDN on the public side to handle the load of deployments worldwide.

For private environments, the ability to run a local build foundry instance is allowed and this can do pulls from the internet. When that pull happens, we will allow a “serverless” processing of each build so that they can handle CA and other items needed for local validation. Once it is deployed locally, each of the local build foundry instances can manage what is allowed to be viewed and used for a local CF install via the CLI and Cloud Foundry instance.

(SHOW SERVER ARCHITECTURE MAP HERE)

Buildpacks

Buildpacks are very straightforward as they have been around the longest. Currently, the core buildpacks are maintained by Cloud Foundry and will be auto published to the registry. Where the registry becomes beneficial is that if users do not upgrade their environment fast enough, the registry will do the buildpacks for them by using the latest version.

We will then allow users to upload and publish their buildpacks into the system. They will also be able to publish “test” applications that the foundry will verify when deployed on a daily basis and show the results of these across the latest CF build and other service providers that are Cloud Foundry Certified.

One thing that the build foundry needs to fix is to certify as many of the components as possible via hash and other methods (CA?) so that enterprises can be certain of the build.

When a user goes to deploy an application, they will be able to reference a buildpack or now a registry namespace:

```
cf push my-new-app -b  core-dotnet
```

In this scenario, not only will it validate it, but it will also pull from the correct location and deploy it into Cloud Foundry. If Cloud Foundry allows for caching of this, it will also do this.

Services

Most of the work of this system will be to support services as this is very new. The plan is to open source and support a packaging format based on the Pivotal tiles that allow them to deploy their services.

The goal will be to provide the spec on this and open source it with the base level needed to deploy services. This will also require a **services gateway and services registry system** that, most likely, will end up being a side project of this core.

With this new package format, the goal will be to allow vendors to publish their services as deployable in any Cloud Foundry environment. By doing this, it will be the next marketplace - but only for open source.

NOTE: We will need to make it so they can handle licensing for software that is deployed

For 3rd party services, this deployment package will wire up the ability to easily enable the service and charges associated with it via this new **services gateway and services registry system.**

Most likely, this process will use either BOSH or standard CF push methods to deploy a service. This is basically giving the option for inline vs. outline deployments.

Cloud Foundry Integration

The following integration will be required for integration into Cloud Foundry:

- The ability to be a community member and send success, failure, and downloads in an anonymous fashion.
- Caching capability for Cloud Foundry to use the registries as their default.
- Ability to use a local registry as the defacto standard and limit custom or other packages to be deployed to cloud foundry.
- Ability to see via command what buildpacks are out of date.
- Ability to update buildpacks to their most current builds.
- Ability to update services to their most current builds.
- Ability to see services that are out of date.

Cloud Foundry CLI Integration

- CLI plugin integration with build foundry to allow it to act as an ambassador for the registry.
- To publish to a specified registry.
- To update to a specified registry.
- To delete to a specified registry.
- To get an endpoint of what registry is valid for this cloud foundry instance (local or public).
- To validate the buildpack or service against other cloud providers by specifying a test system.
- To validate the sources via hash.