Service Catalog, Configuration, and Deployment (CCD) System

Description

The service catalog binds together the K8s service files for deploying the service, service descriptions for project admins to configure a service with options via a CLI/GUI, and provides interfaces to configure service repositories and to manage and maintain the per-cluster service catalog that is cached in etcd.

Design Session Notes - 16/02/16:

Resolved:

- Dev side: Separate docker image builds from catalog/configure/deploy files
    - proposed repo names: ndslabs/images.git, ndslabs/ccd.git
- CCD:
    - Catalog file - Declarative
        - static service description
        - composite dependencies for composites
        - Config options (simple)
    - Configure -
        - Assign options and provision resources
    - Deploy -
        - generate K8s and run

Unresolved:

- Workflow and format for CCD spec files
- Detailed defn and workflow
    - Catalog generation from catalogs in URL
        - Simple aggregation of static data - yes?
    - Configure time - options vs resources
        - Options
            - validatable
            - May influence subordinates
                - Other options - ex. (log(yes|no)->yes)->logfile(input(name))
                - Resources - ex. (logvol(named)->yes)->vols.map(logvol(lookup(name)))
        - Resources
            - Need lookup in system
                - volumes, etc.
                - need consistent naming/identification scheme
        - Generating K8s
            - Unknown, but could leverage other declarative->declarative transformation tools

---

Design Notes - 16/02/15:

- **Goals**

1. Simplicity - Easy to understand and use for cluster admins, project admins, and tool/service developers
    a. Glean as much optional information as possible from build/images/K8s manifests
        i. Volumes, ports, scalability(replication controllers), etc. -
    b. Sensible and simple defaults, with expert override support for Project deployments
2. Automation - Set the catalog references and run - updates/changes appear without intervention
3. Extensible - Multiple catalogs - nds, other-public and private catalogs per-cluster, support experimental/dev ad-hoc use in shared and person dev environments
4. Independent - Not tightly bound to NDS or docker.io, does not require administrative intervention for developers
5. Trusted - support signing/verification as an option for service repos and individual services

- **Workflow Support:**

1. Develop - create descriptions with source
2. Publish - publish catalog service descriptions when publishing image
3. Catalog to Cluster - Cluster admins add catalog refs to cluster, catalog automatically available
4. Service Configuration - Project admin selects from cluster catalog, resolves service options in (CLI/GUI) to generate a deployment on cluster
5. Service Deploy - Project admin deploys configured service in cluster run-time
6. Manage - Track per-project deploys and feedback configuration to CLI/GUI for reconfig (scaling, data mgmt, etc.)

- **Data Types and Objects**

1. Kubernetes Pod manifest, Service manifest, and Replication Controller Manifest
    a. Container images (by reference)
2. Catalog-specific info and configuration data (to be specified)

    a. Configuration options
3. Etcd catalog: Above information cached under /nds/service/catalog/<catalog-ref>/<service-name>
4. Project Admin Input: The optional configuration information provided by the project admin on configure/deploy

- **Implementation Possibilities**

1. Service and Configuration file locations:
    a. Service/config distributed inside the implementation container(s) in a specified path: /usr/local/etc/nds/...
    b. Service/config outside container, in parallel catalog server system (Juju model)
    c. Service/Config in related containers, using name-associations (<service>, <service>-ndscat)
    d. Service/Config in related containers with parallel service and config docker repos: ndslabs-service with ndslabs
    e. Service/Config In image tags via Docker, (LABEL in Dockerfile)
    f. Service/Config in git repositories

Discussion:

    a. Simple, docker model, single-image, spec/image not independent
        i. Assume immutable container - service spec's immutable
        ii. Requires pulling containers to access catalog/service info
        iii. startup overhead to pull everything NDS at cluster/catalog start
        iv. Needs catalog-side pull/add for new/changed images
    b. Complex custom model - requires most development
        i. Service spec independent of image
        ii. Requires moderate implementation
            1. catalog server and population process
            2. cluster-side catalog sync for image additions and updates
            3. Difficult independence for devs and 3rd parties - requires catalog server deployment
    c. Simpler with svc-spec independence and docker model
        i. Harnesses docker repos
        ii. Needs docker naming scheme that is searchable via docker cli
        iii. Catalog update - search/pull from repos and store - simple
    d. Complex, not entirely docker-model with svc/container independence
        i. Dual-repo version of #3
        ii. Push to 2 repos involves parallel tagging and dual repo creds
        iii. Prefer #3 for simplicity?
    e. Simple, docker model, svc/image not independent
        i. Similar to #1 without entering container
        ii. Efficient if it's possible to pull only image meta without layers
        iii. Can leverage other meta-data
        iv. Image meta also contains definitive volumes and ports spec's which we need
    f. Simple, svc/image independent, may complicate developers
        i. Easy - parallel git repo for services
        ii. Nice to keep service/configurator/K8s together - single repos

Other Ideas:

- Use an active configurator object/container provided by developer (Juju model)
    - Probably impractical and likely to be misused (shortcuts)