

NDS Labs API

Overview

The NDSLabs API is a RESTful interface to the NDSLabs system. The API is implemented via the NDSLabs [API Server](#).

Key Concepts

- **Project:** A named configuration that relates an administrative user to a set of resource allocations (quotas), and resources (namespace, pods /services, volumes, configurations).
- **Application (aka stack):** A set of services that are administratively related (i.e., started and stopped together). For example, "clowder" or "dataverse"
- **Service specification:** A specification of a logical "service" that includes a name, description, storage requirements, configuration options, and relationships to other services. An NDSLabs "service" may be composed of multiple related containers.
- **Service library:** A collection of service definitions and procedures for adding official/trusted as well as local/development services.
- **Application instance:** An instance of a service configured and running in a project namespace.
- **Resource quota:** A set of soft and hard limits assigned to a project (storage and compute).
- **Volume mount:** A named, allocated storage resource that counts against the project quota and can be mounted by one or more services.

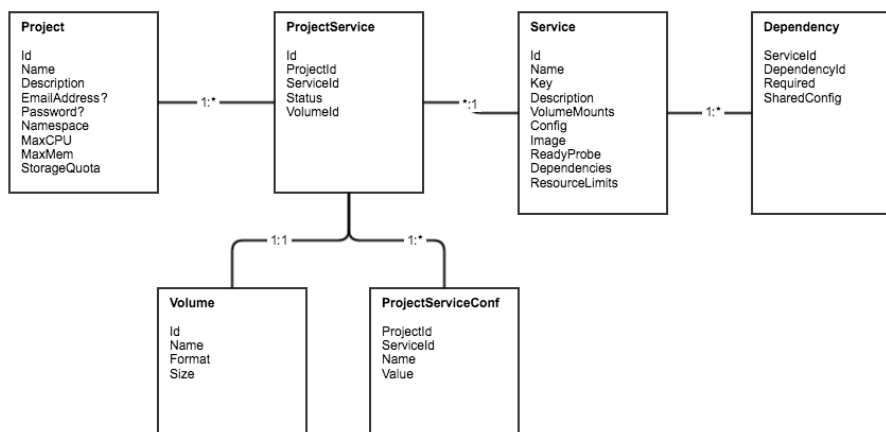
A Simple Use Case

Using the CLI for simplicity: a project administrator wants to add a set of services to their project configuration:

Command	What it does	Response
list services	Lists the services in the service library that can be added to this project.	clowder image-preview
get service clowder	Gets the service specification	<spec>
add service clowder	Adds the specified service to the project	OK
set env smtp-host smtp.ncsa.illinois.edu	Sets a configuration option for the named services	OK
start clowder	Starts the service	<status>
status clowder	Returns the service status	<status>
add service image-preview	Adds the specified service to the project	OK
start image-preview	Starts the image preview service and updates the clowder service?	<status>
stop services	Stops everything	<status>

Entities

The following is a simple entity-relationship diagram intended to capture the entities, attributes, and relationships for the NDSLabs API.



REST API

For actor definitions, see [NDS Labs Use Cases](#).

Base path: /api/

Path	Action	Project Admin	Cluster Admin	Notes
/	List available paths	GET	GET	
/authenticate		PUT	PUT	returns token
/services	List, add site-wide and project-specific services	GET, PUT	PUT	
/services/{service-id}	Get, update, delete site-wide services	GET	PUT, DELETE	
/services/templates/{service-id}/{service controller}	Put, get, delete service templates	GET	PUT, DELETE	
/projects	List, add projects		GET, PUT	
/projects/{project-id}	Get, update, delete project	GET, PUT	DELETE	
/projects/{project-id}/serviceInstances	List, add project services	GET, PUT		
/projects/{project-id}/serviceInstances/{instance-id}	Get, update, delete project service	GET, PUT, DELETE		
/projects/{project-id}/serviceInstances/{instance-id}/status	Get, update, delete project services status	GET, PUT, DELETE		start, stop, restart
/projects/{project-id}/serviceInstance/{instance-id}/config	Get, update, delete service configurations	GET, PUT, DELETE		
/projects/{project-id}/volumes	List, add project volumes	GET, PUT		
/projects/{project-id}/volumes/{volume-id}	Get, update, delete project volumes	GET, PUT, DELETE		
/projects/{project-id}/stack-id	Get, update, delete service stack	GET, PUT, DELETE		
/projects/{project-id}/stack-id/actions/{action}	Start/Stop a stack	GET,PUT		
/projects/{project-id}/stack-id/services/{service-id}	Get, update, delete services in a stack	GET, PUT, DELETE		
/version	Get server version	GET	GET	
/refresh_token	Get a new token	GET	GET	
/register	Post a new project		POST	
/console	Get a Websocket connection to a console	GET		

The latest version of the API specification is available in Github:

<https://github.com/nds-org/ndslabs/blob/master/apis/swagger-spec/ndslabs.yaml>

This can be opened using the Swagger editor demo <http://editor.swagger.io/>, if desired.

Authentication

The API Server currently uses the JSON Web-Token (JWT) approach. The basic flow is as follows:

- POST to /authenticate
 - {"username": "demo", "password": "12345"}
- response
 - {"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE0NTY4NzE3ODgsImkljoiZGVtbYlsm9yaWdfaWF0ljoXNDU2ODY5OTg4fQ.pJ2CQyqXDV675KrAtz3qVzwbM7k-tnZ28Pc0o81GtGU"}
- For all subsequent requests, include the following header:
 - Authorization: Bearer TOKEN
- To refresh the token, simply GET /refresh_token, this will retrieve an updated token

Volumes

- If development environment, creating a volume is just a mkdir on some local path
- If production environment, creating a volume requires a call to the Openstack API (Create, Attach) then a local call to mkfs, then a call to the Openstack API (Detach)

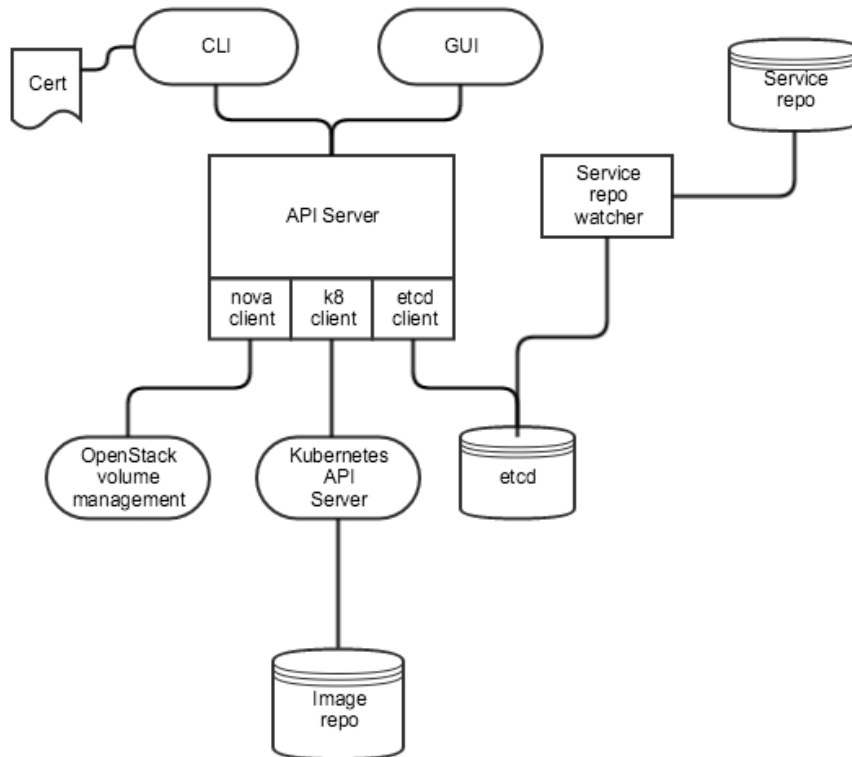
API Server (prototype)

The prototype NDSLabs API Server will be used by the CLI and GUI applications.

Components

We envision the following components:

- CLI
- API Server (either a Jetty application or Go server)
- etcd: Store for service library, project definitions, and services
- Kubernetes API [client](#) (for interaction with Kubernetes services, etc).
- Openstack client to handle volume allocation/initialization tasks



API Authentication

- For the CLI, we could follow the [Kubernetes API server authentication model](#) using client certificates. They use openssl/easyrsa to sign client certificates where the common name is the username (or in our case, project namespace).
- This is not useful for the GUI, which might require user authentication. Perhaps we can pre-generate a password?

Service definitions

How will we manage service definitions going forward so that new services can be added without requiring access to the nds-labs repository? We need to handle two cases: official or trusted services and services under development. The Docker model might suit us: official service definitions are stored in an SCM repo. A repo watcher polls HEAD for changes and imports new definitions into the service library (e.g., etcd). We can use pull requests to handle trust and verification. To handle development, we could store "draft" definitions in a way that are only accessible to a specific project. For example, the developer could issue "cli add service -f myservice.spec" and the service would be added to services/{namespace} or namespace/services. The "cli list services" would return both official and project-local definitions.