Developer support in NDS Labs

- Background
- Notes/Feedback/Ideas From Developer Discussions
  - Motivating case: TERRA Tool Server
- Mockups
  - Overview
  - What is a "development environment"?
  - Home page
  - Sign up (request account)
  - Sign-in
  - Account approval
  - View Dashboard
    - Integrated File Manager
  - Manage catalog
    - Add / Edit Catalog Entry
    - Import Catalog Entry
    - Export Catalog Entry
    - Service catalog organization
  - Add Stack
    - A more compact design
  - View/Edit Stack
  - Manage Stacks
    - Stopped
    - Started
    - Starting
    - Stopping
  - Create/Choose Development Environment
  - Open Developer environment
  - Run/Build Workflow
  - Publish
  - Build (nds docker build)
  - Launch
  - Import development environment
  - Catalogs, publishing and sharing
  - Volumes
  - List of initial development environments:
    - Initial environments should try to support existing NDSC services
  - Developer Quickstart
  - Historic
    - Draft Requirements
    - Tool Audit
    - Development test-run notes
      - Clowder
      - Dataverse
- WIP Mockups
  - Environments
    - View Environments
- Historic
  - Examples of category- and filter-based catalogs
  - Categories

This page captures design and requirements to support developer workflows in NDS Labs.

Background

An outcome of one of the many "What is NDS Labs?" discussions was the idea of supporting end-to-end development (cloud-native) using the NDS Labs "Workbench" service. In a typical scenario today, a developer working on a pilot project would use a local development system (e.g., laptop) or be allocated a small VM to code, compile, deploy, and test their work. For a small development team, multiple VMs would be provisioned – one per developer and often one or more for integration testing. These resources are generally underutilized.

To support end-to-end development in NDS Labs would mean providing the tools necessary for developers to work, mirroring the VM model but minimizing the resource allocation requirements. Required tools might include source control systems (git, bitbucket), IDEs (Eclipse, IntelliJ, TextWrangler, etc), compilers and related dependencies. A must-have will be the developer’s local filesystem and environment customizations/preferences.

Notes/Feedback/Ideas From Developer Discussions

16-06-20: Condensed notes from NDSLabs meeting discussion featuring Rob and Max, condensed and paraphrased by raila.

- Idea of 2-phase approach. Phase 1, NDS Labs provides a small number of general purpose toolsets for developers with instructions, infrastructure, and support that enables them to build Phase 2 - tools for non-expert programmers that are highly bounded, task-oriented in the
context of the stack. For example NDS Labs provides the publishing/sharing system, publishes a set of basic python/java/c++ etc. tools for use by the clowder team. Clowder programmers use the base tools to implement end-user consumable tools designed for non-expert programmers to accomplish specific tasks like implementing a new extractor by simply following patterns. Clowder team would publish ‘end-user’ tools to the clowder ‘channel’ of the catalog and would also use the base tools to refine/develop/publish expert tools for internal and expert professional use.

- Concept of a tool/stack catalog as a single entity with ‘channels’. The preference was for a single catalog system, with the addition of ‘channels’ that can be ‘subscribed’ for specific interests. Channels can be subscribed at-will to pick up tools of the proper sophistication - i.e. scientist using clowder for specific work would subscribe Clowder/user, where she finds only the stacks/configs/tools essential for a clowder consumer role. The concept implied that the entire development environment - the stacks, the tools, the data, and the required expertise vary between developers, a scientists, a novice, an expert (in a tool), and instructor etc. In each case a variation of the stacks and data would be coupled with a role-oriented toolset customized to the task.
- Simplicity and minimalism, tools and environments should be as simple as possible at each 'level' of expertise.
- Customizable, sharable directly. An environment should be customizable and sharable directly with colleagues immediately, even if there is review/approval for publishing and environment. 'The system' should not prohibit agility.
- System should not lock-in to github or any specific tooling, and should be supportive of private repositories.

Motivating case: TERRA Tool Server

DeveloperA has been assigned to the TERRA project and tasked with enhancing the Clowder service to support launching external tools via the new ToolServer service. DeveloperB has also been assigned to the TERRA project and tasked with creating the ToolServer service. DeveloperB is also tasked to work on the PSI project. DeveloperA and DeveloperB require frequent collaboration. DeveloperA needs to be able to change/run/test their Clowder enhancements while running a stable draft of the ToolServer. DeveloperB needs to be able to change/run/test their ToolServer while running a stable draft of Clowder. DeveloperA and DeveloperB need to be able to deploy a stable draft of both tools to a stable demo/integration testing instance, while still working on their local versions.

Mockups

The following mockups are based on ongoing discussions from 6/17 - 6/27

The below mockups were drafted considering the following scenarios:

1. Current workbench user (no development): This is the PI demoing or test-driving an instance; non-developer workshop participants
2. Developer creating a new service
3. Developer extending an existing service
4. Remote developer using a local IDE, but deploying services to Labs for testing.

Overview

Below is a very rough diagram of the use case flow/relationships represented by the below mockups.
What is a "development environment"?

- At least an IDE or filesystem/console container
- May have development tools installed in container or in associated container
- Volume mount ("home directory")
- Associated services (standalone services required by the thing you're building)
- May be unopinionated (tools for a language) or opinionated (tools for a service, e.g., Clowder)

Home page

- Needs to be refined/redesigned to include Sign-up and Sign-in links

Sign up (request account)

- User can request access to the NDS Labs system. This assumes the notion of an account, which replaces our current "project."
- The user can either request an NDS Labs account or sign-up with an existing Oauth-enabled provider.
- User is given default resource limits
- For new NDS Labs account (not Oauth), user is sent email-confirmation message.
- CAPTCHA?
- See also:
  - Account approval

**Gliffy Macro Error**

You do not have permission to view this diagram.

Sign-in

- The user can sign-in using either their NDS Labs credentials or Oauth provider, as selected during account sign-up

**Gliffy Macro Error**

You do not have permission to view this diagram.
Account approval

- After sign-up and the user has confirmed email, account is in an "unapproved" state.
- Email is sent to NDS Labs support or cluster administrator to approve.
- In the simplest approach, email contains an "approval" link with simple confirmation page.
- In more complex scenario, admin has access to an admin interface that supports approval workflow.
- See also:
  - NDS-212 - Getting issue details... STATUS
  - Account Approval

View Dashboard

- This is the initial dashboard/landing page
- User can:
  - Access home directory using a basic file browser/console interface. The home directory will be mounted to every running container and allow the user to copy files to/from containers in stacks.
  - View/manage running stacks (view/edit config, start/stop, add stack, add service spec to personal catalog)
  - View/manage running development environments (create new, open existing, configure volumes)
- This covers the developer and non-developer cases; the non-developer will have no development environments and will interact primarily with the stacks section.
- Open issue: Are development environments just stacks? Do we really need to have a separate panel?
- You can imagine storage and usage information here as well, along with "recently used" or preview information.

Integrated File Manager
Manage catalog

- Allows the user to
  - View spec details in catalog
  - Add new spec
  - Import spec (JSON)
  - Edit existing spec (if owner)
  - Share spec

Add / Edit Catalog Entry
Import Catalog Entry

Paste your JSON spec here:

```json
{
    "key": "mongo",
    "label": "MongoDB",
    "image": "mongo:3.2.4",
    "description": "A cross-platform document-oriented NoSQL database",
    "display": "standalone",
    "access": "internal",
    "ports": [
        { "port": 27017, "protocol": "tcp" }
    ],
    "volumesMounts": [
        { "mountPath": "/data/db", "name": "mongo" }
    ],
    "resourceLimits": {
    }
}
```

Export Catalog Entry
Service catalog organization

- User can view the catalog of available services
- User can search for services (full-text)
- User can filter services based on one or more tags/labels
- User can view both global and personal catalogs
Preliminary list of tags/labels from: https://s3-eu-west-1.amazonaws.com/pfigshare-u-files/3202163/RDMTI.pdf

For example:

- Repository
- Archive
- Active data management
- Collaboration
- Catalog
- Current research information system
- Workflow management
- Data transfer
- Identifier services
- Identity services

Also include tags related to language/technology/platform:

- Java, Scala, C++, etc

Add Stack

- Similar to current Workbench flow for add/configure/start/stop stack
- Add stack opens service catalog view.
- Catalog contains global and user-specific services
- User selects a service from the catalog and the wizard collects configuration information
- On completion, user is taken to dashboard.
- The catalog is assumed to be a combination of global, project, and user-specific services.
A more compact design
View/Edit Stack

- Opens "Manage Stacks" with panel open for currently selected stack

Manage Stacks

- This is the current workbench stack view – allowing the user to view/edit/start/stop multiple stacks in a single page.
- This is a multiple stack view

Stopped
Lorem Ipsum

Started

Lorem Ipsum

Starting
Create/Choose Development Environment

- The user can select from a list of existing development environments/bundles (see list of supported default environments below)
- If a development environment does not meet the user’s need they can create/import their own
- The user can map volumes from their home directory to a logical/known paths in the development container.
Open Developer environment

- When the user selects "Open" next to the developer environment name in the dashboard it opens the associated environment.
- The user uses the selected environment to modify code, access console, build and deploy new services (via command line).
- If possible, we will try to overlay the "Build" and "Publish" (TBD) links over this interface.

Run/Build Workflow

- Two cases:
  - Local: edit/build/run/debug – enabled by tools in development environment (e.g., IDE, make, maven).
  - Running in cluster under control of a spec.
- We have discussed the following run/build workflow:
  - User opens development environment, imports code from source control system.
  - User does some work (edit, build, debug, repeat). This is the "local" build/run workflow and is done entirely in-container, using development-environment specific tools.
When the development work is complete, the user will prepare any release (snapshot) artifacts (e.g., mvn package, make dist, etc).
Build: this is effectively docker build, tag, push. There are various mechanisms for this in-cluster:
- A Docker build kubernetes job deployer (see docker-build-push.yml), establishing a personal docker environment per-person (seems to "big"), leveraging docker hub auto builds, etc.
- Note: do we need to support multiple Docker images per development environment, or should we assume only one?

Publish
- This is effectively "Add spec" to the user's local or global catalog
  - Local:
    - Fill in details via wizard or create JSON spec by hand
  - Global:
    - Pull request – manual (documentation) or automated
    - Cannot have "latest" version

Build (nds docker build)
We've discussed several options for building. The main concern is that we don't want to give users access to the cluster docker socket:
- Jenkins:
  - Install Jenkins container
  - Use API to add jobs via CLI or GUI
- Running docker build/docker via (isolation)
  - Sidecar container
  - Privileged pod
- Run contained instance of Docker for each user

Launch
- The user can launch images created from the "build image" action via the standard start/stop stack workflow (above).

Import development environment
- If the user is willing to create the development environment spec and associated images outside of NDS Labs, they are free to do so and import their configuration.

Catalogs, publishing and sharing
- We all agree that there will be a global "catalog" and user-specific catalogs
- The concept of "channels" (publish/subscribe) has been discussed
- TBD:
  - What does it mean to publish (e.g., add image to registry, spec to catalog?)
  - How do we share services between users?

Volumes
- Currently, volume support is based on one large mounted filesystem
- We are assuming that each account will have a home directory and an "app data" directory for stack-related volumes
- TBD: Does this need to change for Beta?
- Notes from discussions 6-22
  - Volumes covered by global FS and home dirs.
  - Stack volumes automatic (no GUI involvement), created by UUID in user home/dir path (My Volumes)/2344-2284-9999/...
  - Could add a (My Volumes)/by_name/ for navigating, or put something in view deployed to find the data under home dir
  - Quotas, if any are only on a users home-dir
  - User is responsible for managing their own data in the space

List of initial development environments:

<table>
<thead>
<tr>
<th>Environment</th>
<th>Used By</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Dataverse, Dspace</td>
<td>Need standalone and webapp support (Tomcat, Glassfish, etc)</td>
</tr>
<tr>
<td>PHP</td>
<td>Hubzero</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td>Zenodo, yt?, Clowder</td>
<td></td>
</tr>
</tbody>
</table>
Ruby
Node
Out-of-the-box in Cloud9, but maybe we add Bower, Grunt, etc.
Go
Working prototype in Cloud9
C/C++
iRODS, OpenCL, Cuda
Scala?
Clowder, Latis
Groovy
Latis

Developer Quickstart

If a service has an opinionated development environment and known repositories, it should be possible for a user to quickly launch the associated development environment with minimal configuration.

- Select "Developer Quickstart" for a particular service
- User is prompted to 1) clone, 2) fork, or 3) manually check out associated service. If 1 or 2, user is prompted to select path where code will be checked out.
- Development environment is started with default IDE and configuration and repositories pre-imported.

Historic

Keeping these notes around, but out of date as of 6/17

Draft Requirements

Notes from 6/2 meeting

To review (find out how their terms/concepts map to ours):

- CodeEvny
- OpenShift
- Deis
- Heroku
- Fabric8
- Eclipse CHE/IDES
- Terminology+

Roles:

- Site admin
- Cluster admin
- Project admin
- Project developer

Requirements

- Access, authentication and authorization
  - Project admin grants access to a developer to the project (project=shared storage, compute, ?project specific service catalog)
  - User/group/permissions (groups/permissions – grouper?)
  - Developer is notified (registration workflow)
  - Project admin can view resource usage (what resources are used by each team member)
  - Project admin can manage groups and users (create/edit/delete)
  - Developer can belong to more than one project
  - Developer can login, change password, recover password (oauth?)
  - Project admin can configure authentication provider ? (Github)

- Developer view
  - Developer: first time access sees welcome + tutorial
  - Developer can view landing page/dashboard (system status, projects, workspaces, recent containers, tc)

- Bundles
  - Project admin or developer can create a project "bundle"?
  - Developer can use a project "bundle"?
  - Developer can save configuration (e.g., bundle)

- Developer actions
  - Developer can configure environment (e.g., git accounts, bash_profile, etc)
  - Developer can view/manage filesystem
  - Developer can access shell/prompt with common tools (git, vim, nano, etc)
  - Developer can view tool catalog (Eclipse, Jupyter, Rstudio, Pyclipse, etc)
  - Developer can launch a development tool
Tool Audit

IDEs and analysis tools:

- CodeEnvy/Eclipse Che
- Cloud9
- Koding
- CodeAnywhere

Paas/Application platforms:

- Deis
- Heroku
- Fabric8
- OpenShift Online
- VS Team Services

Development test-run notes

Notes from trying to develop existing tools using the Codenvy and Cloud9 services.

Clowder

Tried developing Clowder in both IntelliJ and Eclipse. Will need a quick tutorial from the ISDA team.

Dataverse

Dataverse is a Maven-based project, which should be ideal for Codenvy. Created new project and imported code directly from Github as a Maven project. The project builds without problem. Created new Runner Glassfish 4.6+Java 7. Startup failed with error:

- remote failure: Error occurred during deployment: Exception while deploying the app [application] : Class edu.harvard.iq.datadeposit.SwordAuth has unsupported major or minor version numbers, which are greater than those found in the Java Runtime Environment version 1.7.0_55. Please see server.log for more details.
- [STDERR] deploydir command deprecated. Please use deploy command instead.

Basically, Dataverse is using new JRE/JDK than supported by Codenvy. With locally hosted Che, we could create our own Java 8.

Created a new blank project in Cloud9, imported Dataverse. The default cloud9 container doesn't have JDK/JRE or Maven.

- sudo add-apt-repository ppa:webupd8team/java
- sudo apt-get update -y
- sudo apt-get install maven
- sudo apt-get install oracle-java8-installer -y
- mvn package

There is no built-in Glassfish or other app container support in Cloud9, so you need to install Glassfish/Tomcat and any other required dependencies. Dataverse has a complex installation process (Glassfish settings, database pools, API calls, default data, Solr config, etc) – that doesn't lend itself to a rapid build/deploy process. What we really want is a set of steps to setup the application container (connecting Glassfish to Postgres, Solr, etc) and simply be able to drop the new war file in the Glassfish webapps directory.

WIP Mockups

Below are some interactive mock-ups to demonstrate how the proposed UI pages and functions will relate.
Feel free to click on some links or buttons and let us know what you think!

Environments

View Environments

Historic

Examples of category- and filter-based catalogs

Some examples of decent category-based selectors (from catalog organization discussion), muon, redhat system-config-packages, synaptic:

Categories

NOTE: (item) denotes a service that we do not yet support, but could easily be imagined
• Any new services offered that do not easily fit into one of these categories may yield new categories
• Tabs would allow the user to select from global / user (/shared):
• User can choose create their own categories for their personal catalog - this is not required for beta
• Selecting a category name brings up a list of all service under that category
  • Drilling down into subcategories further refines results
  • Categories offered might include the following:

1. All
2. Storage
   a. Databases
      i. Relational
         1. MySQL
         2. PostgreSQL
         3. (MariaDB)
         4. (InfluxDB)
      ii. Document
         1. MongoDB
         2. (CouchDB)
      iii. (Key-Value)
         1. (etcd)
         2. (Riak)
         3. (Redis)
      iv. (Column)
         1. (Cassandra)
         2. (Clusterpoint)
      v. (Graph)
         1. (Neo4j)
         2. (Stardog)
       vi. (Multi-Model)
         1. (ArangoDB)
         2. (FoundationDB)
   b. Files
      i. iRODS ICAT / iRODS CloudBrowser
      ii. ownCloud
      iii. (Pydio)
3. Computation
   a. Clowder
   b. Extractors
      i. Audio
         1. audio-preview
         2. speech2text
      ii. Image
         1. image-preview
         2. image-metadata
         3. plantcv
      iii. Video
         1. video-preview
   c. Dataverse
   d. Rserve?
4. Development Environments
   a. Go
   b. Node.js
   c. (Java)
   d. (Java + Scala?)
   e. (C/C++)
   f. (Python)
   g. (PHP)
5. Message Brokers
   a. RabbitMQ
   b. (ActiveMQ)
   c. (Kafka)
   d. (Kestrel)
6. Logging
   a. Kibana
   b. (graylog2)
7. (Performance Monitoring / Profiling)
   a. (grafana)
   b. (graphite)
8. Search Engines
   a. Elasticsearch 2.0
   b. (Lucene)
   c. (Riak Search 2.0)
   d. (Yokozuna)