Charmworld Documentation

Release 0.0

Team Charmworld

November 22, 2013

Contents

Contents:

CHAPTER 1

Deploying to staging/production

1.1 Using the Incantation

The incantation script will deploy charmworld as manage.jujucharms.com with apache2, squid, and haproxy front-ends.

cd <workspace> bzr branch lp:~charming-devs/canonical-marshal/charmworld-incantation cd charmworld-incantation/script ./deploy.sh

CHAPTER 2

Hacking on Charmworld

2.1 Adding Dependency

Charmworld uses virtualenv to isolate the environment. In order to add a new package as a requirement you need to add it to the *requirements.txt* file and make sure you lock the version.

The make target *make deps* will then install the new dependency.

Note The package must exist in the download cache. See *lp:~juju-jitsu/charmworld/download-cache*.

2.2 CSS Work

Charmworld is currently using the Ubuntu CSS framework. Additional classes may be added in static/css/base.css but that should be avoided unless absolutely necessary.

2.2.1 Auto Updating

You can use a tool like watchdog to auto update the css when you're hacking on it. A sample watchdog command would be:

```
bin/watchmedo shell-command \
--patterns="*.less" \
--command='make css' charmworld/static/css/
```

2.2.2 Hacking with Canonistack

```
# Canonistack
$ source ~/.canonistack/novarc
$ juju bootstrap
$ juju status

# Wait for environment to come up
$ juju deploy mongodb --constraints instance-type=m1.small
$ juju set mongodb dbpath=/mnt/mongodb
$ juju deploy cs:~charming-dev/precise/elasticsearch --constraints instance-type=m1.small
```

```
$ juju deploy cs:~juju-jitsu/precise/charmworld --constraints instance-type=m1.small
$ juju set charmworld error-email=<your email address>
$ juju add-relation charmworld mongodb
$ juju add-relation charmworld elasticsearch:essearch

# If giving a public address (which is rare)
$ juju expose charmworld
$ euca-describe-instances
$ euca-associate-address -i i-XXXXX XX.XX.XX.XX

# Wait 15-20 minutes to complete the first ingest.
$ sshuttle -r <charmworld ip> <charmworld ip>
```

2.2.3 Hacking with local (lxc)

```
$ juju bootstrap
$ juju status
$ juju deploy mongodb
$ juju set mongodb dbpath=/mnt/mongodb
$ juju deploy cs:~charming-dev/precise/elasticsearch
$ juju deploy cs:~juju-jitsu/precise/charmworld
$ juju set charmworld error-email=<your email address>
$ juju add-relation charmworld mongodb
$ juju add-relation charmworld elasticsearch:essearch

# Wait 15-20 minutes to complete the first ingest.
$ sshuttle -r <charmworld ip> <charmworld ip>
```

2.2.4 Hacking locally

Running locally will install a number of packages, but if you don't mind it is a fine way to proceed.

:: $ make sysdeps $ make install $ make run $ bin/enqueue –prefix=~charmers/charms/precise –limit=100 $ bin/ingest-queued

CHAPTER 3

Charmworld API 3 Docs

The charmworld API is a restful, versioned API. All functionality for a given API version can be accessed from paths under `$site-root/api/$version`. For example, if charmworld is deployed at `manage.jujucharms.com`, the API root for version 1 is at `http://manage.jujucharms.com/api/1/`. All API endpoints are sub-paths underneath this.

3.1 Terminology

"Approved", "reviewed", "official" "recommended" and "promulgated" all mean basically the same thing– that the charm has been examined and found worthy. This is controlled on Launchpad by linking the charm branch to the corresponding sourcepackage in the "charms" distribution.

"Community" charms and bundles are those which have not been examined, or have not been found worthy.

`charm-id` The path portion of a Charm Store URL. For promulgated charms, this is of the form `$series/$name(-$revision)`, and for non-promulgated charms, this is of the form `~$owner/series/$name(-$revision)`. Revision may be omitted, in which case the head revision will be used.

`bundle-id` An identifier of a bundle. Three forms are accepted: `~owner/basket/revision/bundle` is the canonical, unchanging bundle id, and works for promulgated and unpromulgated bundles. `~owner/basket/bundle` refers to the head revision of a bundle. It appears to be intended to support unpromulgated charms, but it currently supports only promulgated charms (bug #1218949). `basket/bundle` refers to the head revision of a promulgated bundle.

3.2 Endpoint output

The responses from API endpoints typically wrap the results with dicts which contain the item itself and a metadata field indicating the doctype, and possibly other data.

```
{
  "metadata": {
    "doctype": "charm",
  }
  "charm" {
    /* actual charm output (see below)*/
```

```
  }
}
```

3.3 Bundle output format

```
{
    /* The bundle id */
    'id': '~bac/byobu/4/bat',
    /* The sourcepackage name of the basket branch. */
    'basket_name': 'byobu',
    /* The bzr revision of the basket's branch. */
    'basket_revision': 4,
    /* The bundle name */
    'name': 'bat',
    /* The owner of the basket's branch. /*
    'owner': 'bac',
    /* True if the branch has been deleted since it was first registered. */
    'branch_deleted': False,
    /* The actual bundle, in JSON.
    'data': {
        'series': 'precise',
        'services': {},
        'relations': {}
    },
    /* A user-supplied title. */
    'title': '',
    /* A user-supplied description */
    'description': '',
    /* A URL that the user should be able to use to deploy the bundle. */
    'permanent_url': 'bundle:~bac/byobu/4/bat',
    /* See Terminology */
    'promulgated': False,
}
```

3.4 Charm output format

```
{
  /* The charm-id for this charm. */
  "id": "precise/apache2-10",
  /* The owner of the charm's Launchpad branch.  Unlike maintainer, this is
   * enforced by Launchpad. */
  "owner": "charmers",
  /* The name of the charm */
  "name": "apache2",
  /* The series the charm is associated with, in its distribution */
  "distro_series": "precise",
  /* The maintainer of the charm, according to metadata.yaml. */
  "maintainer": {
    /* The email address of the charm maintainer. */
    "email": "liam.young@canonical.com",
    /* The full name of the charm maintainer */
    "name": "Liam Young"
  },
```

```
/* The date the charm was created (i.e. the date its branch was created on
 * Launchpad), in JS-compatible ISO8601 */
"date_created": "2013-02-01T21:15:47Z",
/* The description of the charm from its metadata. */
"description": "The Apache Software Foundation's goal is to build a
  secure, efficient\nand extensible HTTP server ...",
/* The charm revision as seen in the revision file. */
"revision": 5,
/* The summary from metadata.yaml */
"summary": "Apache HTTP Server metapackage",
/* The Juju providers tested with the charm and their Jenkins status.
 * Possible values: 'SUCCESS', 'FAILURE', 'UNSTABLE', 'ABORTED'. */
"tested_providers": {
  "ec2": "SUCCESS",
  "local": "SUCCESS",
  "openstack": "SUCCESS"
},
/* The URL of the charm in the store. */
"url": "cs:precise/apache2-10",
/* The number of downloads of this charm */
"downloads": 51,
/* The number of downloads of this charm in the past 30 days */
"downloads_in_past_30_days": 41,
/* See Terminology */
"is_approved": true,
/* If true, the charm is a subordinate charm. */
"is_subordinate": false,
/* Always 0 */
"rating_denominator": 0,
/* Always 0 */
"rating_numerator": 0,
/* The options of this charm, as seen in config.yaml (but represented as
 * JSON). */
/* Information related to the charm source files */
"code_source": {
  /* A link to where bugs on the charm can be reported. */
  "bugs_link": "https://bugs.launchpad.net/charms/+source/apache2",
  /* The last commit message for the charm. */
  "last_log": "[hloeung,r=mthaddon] Fix error below when SSL chain
      certificate file is missing, and create cert from template for
      self-signed cert",
  /* The location where the charm can be found. */
  "location": "lp:~charmers/charms/precise/apache2/trunk",
  /* The vcs revision, as stored in the charm store. */
  "revision": "44",
  /* A list of vcs revisions, from newest to oldest.  At least 10 revisions
     are included, and at least the past 30 days are covered, so this list
     can be used to display the last 10 revisions or count the number of
     commits in the past 30 days. */
  /* The source type, typically "bzr" */
  "type": "bzr",
  /* A list of revisions, applicable when the "type" is "bzr". */
  "revisions": [
    {
      /* A list of the authors of this revision.  With Bazaar, there is
       * typically one author, and this is typically the committer, but
       * this can be overridden in bzr with the --author parameter. */
      "authors": [
```

```
          {
            /* The email address of the author.  It is impolite to display
             * this.  Its main function is to uniquely identify the author.
             */
            "email": "tom.haddon@canonical.com",
            /* The full name of the author. */
            "name": "Tom Haddon"
          }
        ],
        /* The date and time the commit was made, in the
         * JavaScript-compatible subset of ISO8601 in UTC. */
        "date": "2013-05-02T10:05:32Z",
        /* The commit message */
        "message": "[hloeung,r=mthaddon] Fix error below when SSL chain
          certificate file is missing, and create cert from template for
          self-signed cert",
        /* The vcs revision number of the revision */
        "revno": 44
      }
    ],
  },
  /* A list of the files in this charm that are accessible, as paths from
   * the charm root. */
  "files": [
    "hooks/website-relation-joined",
    "hooks/balancer-relation-broken",
  ],
  "options": {
    "config_change_command": {
      "default": "reload",
      "description": "The command to run whenever config has changed.",
      "type": "string"
    }
  },
  /* A mapping of the charm's relations, as seen in metadata.yaml */
  "relations": {
    "provides": {
      "nrpe-external-master": {
        "interface": "nrpe-external-master",
        "scope": "container"
      }
    },
    "requires": {
      "balancer": {
        "interface": "http"
      }
    }
  }
}
```

3.5 Listing charms and bundles

Charm and bundle listings are provided by the `search` endpoint using the GET method. This lists all well-formed charms and bundles that match query parameters.

Query parameter	Description
text	Text that the item must match. See below for more details.
autocomplete	Search for autocompletions. This will match only on the name field, and will match on prefixes of the name.
categories, name, owner, provides, provider, requires, series, summary	Used as a filter. Each filter parameter may be repeated. Items matching any of the selected values for a filter are selected, so name=1&name=2 would match items whose name was either 1 or 2. However, if multiple filters are specified, the charm must match all of them, so name=1&series=2 will only match charms whose name is 1 *and* whose series is 2. Filters on lists of values (categories, requires, provides) match on a single element of the list. So if a charm requires both http and ftp, requires=http will match it.
type	If specified, must be appproved or community. (See Terminology)
limit	The maximum number of items to return. By default, all matching items are returned.

3.5.1 Text matching

For charms, the following fields are considered for partial matches: summary, description, config option description, relations, store_url, and interface names. Additionally, the following fields will match if the text exactly matches the contents of that field: name, owner, series.

For bundles, the following fields are considered for partial matches: basket, description, title, charm names. Additionally, the following fields will match if the text exactly matches the contents of that field: name, owner, series.

3.6 Listing front-page charms and bundles

The front-page items are retrieved by performing a GET to search/interesting.

The result is a mapping of new, featured and popular charms and bundles. new is a list of the 10 most recently-created items (according to their Launchpad branches), from most to least-recently created. popular is a list of the 10 most-downloaded items, from most to least-downloaded. featured is a list of all items that have been manually selected as featured, in unspecified order.

3.7 Retrieving an individual charm

An individual charm can be retrieved by invoking GET on charm/$charm-id, where the charm-id is appended to the path without escaping. So if the charm-id is precise/foobar, the path is charm/precise/foobar, not charm/precise%2ffoobar.

3.8 Retrieving related charms

Information about which charms are related to a charm can be retrieved by invoking GET on charm/$charm-id/related. For example, if the charm-id is precise/foobar, the path to the end-point is charm/precise/foobar/related.

Charms are considered "related" to a given charm if

1. They provide an interface that the charm requires, or
2. They require an interface that the charm provides, and
3. Their series is the same as the charm's [1] and
4. Their name is different from the charm's.

The API call returns a mapping in which `result.provides` includes all charms that provide interfaces used by the charm, organized by interface, and `result.requires` includes all charms that require interfaces provided by the charm.

As an example:

```
{
  result: {
    requires: {
      http: [/* list of charms that require http*/],
      sftp: [/* list of charms that require sftp*/]
    }
    provides: {
      bluetooth: [/* list of charms that provide bluetooth */]
    }
  }
}
```

An abbreviated form is used for related charms:

```
{
  /* The charm-id of this charm */
  "id": "precise/limesurvey-4",
  /* The name of this charm */
  "name": "limesurvey",
  /* True if the charm has an icon.svg file. */
  "has_icon": true,
  /* The categories associated with the charm. */
  "categories": [],
  /* The number of downloads of this charm */
  "downloads": 51,
  /* The number of downloads of this charm in the past 30 days */
  "downloads_in_past_30_days": 41,
  /* The number of commits to this charm in the past 30 days */
  "commits_in_past_30_days": 5,
  /* See Terminology_ */
  "is_approved": true,
  /* The weight of this charm, for sorting purposes.
   * Weights are relative to other weights-- their
   * absolute value has no meaning. */
  "weight": 10.0
},
```

3.9 Retrieving the files that make up a charm

The contents of a file in a charm can be retrieved by invoking GET on `charm/$charm-id/file/$path`. For example, if the file is `hooks/install` and the charm-id is `precise/foobar`, the path is `charm/precise/foobar/file/hooks/install`.

[1] While it is technically possible for charms in different series to be deployed in the same environment, we believe this is an unusual thing to do, and an unhelpful thing to suggest.

Not all charm files are available. The available files are listed in the `files` member of the charm, and include `icon.svg`, `readme.*`, `config.yaml`, `metadata.yaml`, `revision`, `hooks/*`.

3.10 Retrieving the icon for a charm

The icon for a charm can be one of several.

- If the charm provided an icon and it is a reviewed charm we return the actual svg provided.
- If the charm is not reviewed, but has a category then we create a redirect to a fallback icon.
- Else we redirect to a generic icon.

3.10.1 Retrieving the icon for a charm given a charm id

`charm/$charmid/file/icon.svg` will respond with either an icon or a 302 redirect to an icon that can be used for the charm.

3.10.2 Resolving an icon without a charm id

You can still get an icon for a charm without a charm id. Bundles may specify the charm in them in many formats. The API supports getting directed to the correct charm icon in any of the following ways:

- store_url: `charm/resolve-icon/cs:series/charm-revision`
- launchpad branch: `charm/resolve-icon/lp:xxxxxxxx`
- launchpad branch with revision: `charm/resolve-icon/lp:xxxxxxxx@revison`
- latest revision of a promulgated charm: `charm/resolve-icon/series/name`

3.11 Retrieving QA data

The results of QA review of a charm can be retrieved by invoking GET on `charm/$charm-id/qa`. The return value is formed like this:

```
{
  "result": {
    /* A list of the current questions */
    "questions": [
      /* A question heading */
      {
        /* The display name of the heading */
        "description": "Data Handling",
        /* The unique name of the heading */
        "name": "data_handling",
        /* A list of the questions for this heading */
        "questions": [
          {
            /* A description of the question */
            "description": "Integrate data storage best practices",
            /* More questions */
            "extended_description": "Backups based on service usage",
            /* The question id */
```

```
            "id": "data_handling_0",
            /* The point score for this question */
            "points": 1
          },
        ]
      },
    ],
    "scores": {
      /* Scores for a heading */
      "data_handling": {
        /* Answer to a question, one of "0" (no), "1" (yes) or "" (unknown).
         */
        "data_handling_0": "1"
      }
    }
  }
```

3.12 Retrieving a bundle

A bundle can be retrieved by invoking GET on `bundle/$bundle-id`, where the bundle-id is appended to the path without escaping. So if the bundle-id is `apache2/with-squid`, the path is `bundle/apache2/with-squid`.

The results are in bundle output format, without the typical endpoint output wrapper.

3.13 Retrieving the icon for a bundle

The icon for a bundle can currently only be the default. You may request it based on the bundle's id.

`bundle/$bundleid/file/icon.svg` will respond with a 302 redirect to an icon that can be used for the bundle.

CHAPTER 4

Migrating data in Charmworld

Charmworld uses MongoDB for its data storage. In order to facilitate changes to the design of the data in Mongo a migration tool is provided in *migrations/migrate.py*.

The migration tool stores the current version in the collection *version*.

4.1 Commands

The *migrate* command is created when charmworld *setup.py* is processed.

In the order of typical usage:

4.1.1 Check the current version of the data store

```
$ bin/migrations current
0
```

4.1.2 Add a new migration script to be run.

```
$ bin/migrations new -d "add new data to mongo"
Created new migration: 001_add_new_data_to_mongo.py
```

4.1.3 Check the latest migration available.

```
$ bin/migrations latest
1
```

4.1.4 Run any migrations not in Mongo

```
# Perform any long-running migrations in temp storage
$ bin/migrations prepare-upgrade
# Swap temp storage into place and run short-running migrations
$ bin/migrations upgrade
Updated the datastore to version: 1
```

4.1.5 Initial Upgrade requires –init

Initially the datastore is not tracked. The first upgrade done needs to include *–init* to version the datastore and then process upgrades.

```
$ bin/migrations prepare-upgrade --init
Updated the datastore to version: 1
```

4.2 Deleting migrations

The most recent migration must not be deleted, because it also serves to identify the current migration version for up-to-date instances. All other migrations can safely be deleted if they are no longer needed. So if all production instances have had the migration applied, it is safe to simply delete it.

4.3 Avoid performing an es-update in migrations

The charm and Makefile ensure that es-update is run before running the code, so it is unnecessary to run es-update in a migration.

CHAPTER 5

ToDo

Charm World Tour (Goals for 13.04)

- manage.jujucharms.com

5.1 Roles

5.1.1 commercial charm authors

who want to be able to upload a private charm and manage access to it via grants to users. Ideally they can also see download counts for those charms and possibly who downloaded/used it.

5.1.2 charm farmers

These are members of the launchpad group. They will be doing quality assessments and reviews on charms. The quality assessment form is for them. We've got some existing tools on the site that we'll be relegating to this location (the charm review queue). New tools that are in the nice to have column are the ability to trigger charm/jenkins test against a particular charm in the review queue.

5.1.3 charm universe stakeholders

These are people that want to see reports on the charm universe ie. what's currently linked off jujucharms.com/reports.

5.2 Checklist

- jujucharms.com (short term january sprint)
- [X] display charm testing results
- [x] display charm quality assessment results
- [X] download counts
- jujucharms.com (long term for 13.04)

- [x] we should have ux by february 1st, 2013. (Arrived March 6)
- [/] display juju gui in charm browser mode on the front page
- [ ] minor ui / envelope changes to match the gui to juju.ubuntu.com
- Charm ingest pipeline
- [X] pull in download stats from the juju store
- [x] switch out to mongodb for queues (see mongoqueue pkg)
- [x] switch out search to elastic search
- [ ] put up some queue ui / alerts for errors
- Deployment
- [x] charm things up
- [x] setup a staging site.

CHAPTER 6

Important Links

- http://staging.jujucharms.com/~juju-jitsu/precise/charmworld
- http://staging.jujucharms.com/

CHAPTER 7

Getting started

7.1 Initial Environment

After you branch charmworld you can get started by running:

```
$ make sysdeps
$ make install
```

You can then run the webapp via the helper:

```
$ make run
```

Note that there isn't any data out of the box.

7.2 Importing Data

Data is imported in two steps: queueing the charms, and processing the queue. (The intent is that separate machines may queue and ingest charms.)

```
$ bin/enqueue
$ bin/ingest-queued
```

Output verbosity is controlled by the logging config in charmworld.ini

CHAPTER 8

Testing

Testing is run via nosetests using:

```
make test
```

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*