Zooniverse Developer Docs Documentation

Release 1.0.0

The Zooniverse

September 30, 2016

The primary goal for this site is to provide guides on how to work with our platform.

CHAPTER 1

Domain Model

The domain model is actually pretty simple. We typically think most about the following entities:

1.1 User

People are core to the Zooniverse. When talking publically about the Zooniverse we almost always use the term 'citizen scientist' or 'volunteer' because it feels like an appropriate term for someone who donates their time to one of our projects. When writing code however, the shortest descriptive term that makes sense is usually selected so in our domain model the term we use is User.

A User is exactly what you'd expect, it's a person, it has a bunch of information associated with it such as a username, an email address, information about which projects they've helped with and a host of other bits and bobs. Crucially though for us, a User is the same regardless of which project they're working – that is Users are pan-Zooniverse. Whether you're classiying galaxies over at Galaxy Zoo or identifying animals on Snapshot Serengeti we're associating your efforts with the same User record each time which turns out to be useful for a whole bunch of reasons (more later).

1.2 Subject

Just as people are core, as are the things that they're analysing to help us do research. In Old Weather it's a scanned image of a ship log book, in Planet Hunters it's a light curve, whereas in Snapshot Serengeti it is a photograph of an animal. Regardless of the project, internally we call all of these things Subjects. A Subject is the thing that we present to a User when we want to them to do something.

Subjects are then stored in our database with a collection of metadata added from the manifest the user supplied while uploading. For example in Galaxy Zoo we might store some metadata associated with the survey telescope that imaged the galaxy and in Cyclone Center we store information about the date, time and position the image was recorded.

It is important to keep in mind that a Subject is not the same as an image. For some projects, Subjects will have multiple images that a User can cycle between. For others, a Subject will not be an image at all, but might have audio recordings, movies, or pieces of text instead. A Subject could also have multiple different media types at the same time.

1.3 Workflow

The Workflow is another main entity in our system. Most importantly, the Workflow has the configuration of Tasks we ask our volunteers to perform when they are presented with a Subject. In Notes from Nature each step of the transcription (such as 'What is the location?') is a separate Task, in Galaxy Zoo, each step of the decision tree is a

Task too. The Workflow also determines which Subjects are available to work on, and how long they should be worked on.

1.4 Classification

As a User is presented with a Subject and progresses through the Workflow, they produce a Classification. The Classification is the core unit of human effort produced by the Zooniverse community as it represents what a person saw and what they said about it. We collect a lot of these!

1.5 SubjectSet

Sometimes we need to group Subjects together for some higher level function. Perhaps it's to represent a season's worth of images in Snapshot Serengeti or a particular cell dye staining in Cell Slider. Whatever the reason for grouping, the entity we use to describe this is 'SubjectSet'. The assignment of Subjects to Workflows actually happens by assigning SubjectSets to workflows, because a project often has hunderds of thousands of Subjects and assigning each of those individually would be tedious to do by hand.

1.6 Project

The Zooniverse APIs host a whole bunch of different Zooniverse projects so it's probably no surprise that we represent the actual citizen science project within our domain model. No prize for guessing the name of this entity – it's called Project. A Project is really just the overarching named entity that SubjectSets and Workflows are associated with. This is also what you create when you "build a project" on our homepage.

1.7 Finally

Our domain model has also been heavily influenced by the patterns that have emerged working with science teams. In the early years we spent a lot of time abstracting out each step of the User interaction with a Subject into distinct descriptive entities called Annotations. While in theory these were a more 'complete' description of what a User did, the science teams rarely used them and almost never in realtime operations. The vast majority of Zooniverse projects to date collect large numbers of Classifications that are write once, read very much later. Realising this has allowed us to worry less about exactly what we're storing at a given time and focus on storing data structures that are a convenient for the scientists to work with.

CHAPTER 2

Project Builder

CHAPTER 3

Dealing with classifications

When you request an export of the raw classification data using the project builder, some of the columns we return will actually contain values in a format called JSON. We do this, because sadly sometimes the kinds of data we track are too complicated to easily fit into a table structure. However, using Python it's actually really easy to pull out the data you need from those JSON-based columns:

```
import pandas
import json

data = pandas.read_csv("classifications.csv")

data["annotations"]  = data["annotations"].map(json.loads)
data["metadata"]     = data["metadata"].map(json.loads)
data["subject_data"] = data["subject_data"].map(json.loads)
```

This will turn those columns into a normal Python dict. If you know your project has classifications pertaining to a single subject at a time, you can make things even simpler with a further step:

```
def flatten_subject_info(subject_data):
  result = subject_data.values()[0]
  result.update({'id': subject_data.keys()[0]})
  return result

data["subject_data"] = data["subject_data"].map(flatten_subject_info)
```

Note: This example assumes you have installed the Python library called Pandas. Many scientific Python distributions include this library, but you can install this with *pip install pandas* otherwise.

Todo

expand

CHAPTER 4

How to upload externally-hosted images to Project Builder Subject Sets

Panoptes supports the concept of an "external" subject, that is one whose source images are hosted externally rather than being uploaded through the project builder to S3 in the normal manner.

Currently this functionality is not available through the project builder.

Instead you will have do one of the following:

1. Use the "run rake task" job on Jenkins to add your images
2. Gain access to a Panoptes dump worker instance and run the rake task to add your images there
3. Use the subject uploader script (this is not covered in this document).
4. You can write a script to use the API directly - see API reference on creating a subject.

This document explains approaches A and B, as well as the preliminary steps common to all approaches.

4.1 Step 1 - Create your CSV

This is common to both approaches. You'll need to create a CSV, with a header row, where the first column is called `url` and contains the URL of your image. One image per row. (TODO: Can someone add info on how to upload a multiple-image subject?). The remaining fields are metadata, these will be available in the subject viewer interface via the (i) button - so you may want to convert them to "human-friendly" strings rather than database values. Here is an example of a trio of scripts (one, two , three) which, when run in sequence, will create CSVs from Snapshot Serengeti subjects.

Once you have a CSV you should upload it to a publicly available URL. One of the easiest ways to do this is to use gist (command-line uploader here) from github.

Important note: Once you have got the gist link, such as `https://gist.github.com/alexbfree/a913899eb9cef84110c` you need to actually paste it into your address bar, then click raw on the csv file within the github UI to get the expanded URL, something like this `https://gist.githubusercontent.com/alexbfree/a913899eb9cef84110da/raw/dd5e3e859ebfd9c8be06` (note the change of domain). Your upload will fail if you don't do this.

4.2 Step 2 - Prepare subject set and find the necessary IDs

This is also common to both approaches.

You will need to find three IDs - your project ID, your user ID, and the ID of the (probably empty) subject set you've created in your Project Builder Workflow to house the new images.

You can find the project ID by looking at the URL when you are viewing the project in the project builder. For example, `https://www.zooniverse.org/lab/988` tells me my project ID is `988`.

You can find your user ID by using the developer console in your browser to monitor the network traffic when you reload the Zooniverse home page. Look for the request to `/api/me` and it should start with something like `{"users":[{"id":"209","login":"alexbfree"` - this tells me my user ID is `209`.

The user ID you use for the upload will own the images and they will count against the upload limit for that user.

Finally you need to create a new (empty) subject set in the Project Builder (or, you can upload to an existing one if you wish). Once the subject set is created (you'll need to defocus the subject set name box by clicking elsewhere on the page to save it), you can now examine the URL to find the subject set ID. For example `https://www.zooniverse.org/lab/988/subject-set/3941` tells me the subject set ID is `3491`.

Now that you have the project ID, user ID, subject set ID and raw CSV URL, you can proceed to create the rake task, using either of the two approaches:

4.3 Step 3A - Create the rake task from the commandline

This approach takes longer, but is faster and useful if you have more than one file to upload.

First, find the IP of a Panoptes dump worker instance. You can do this by typing `lita aws ip panoptes dump production` on Slack. Pick an instance, such as `ec2-54-174-166-171.compute-1.amazonaws.com`.

Now SSH into that instance (you'll need to be on an authorised network or VPN): `ssh -i /code/Production/keys/zooniverse_1.pem ubuntu@ec2-54-174-166-171.compute-1.amazonaws.com`

Now step inside the docker container (you can validate the container name using `docker ps`): `docker exec -it panoptesdumpworker_panoptes_1 bash`

Now you can run your rake command as follows, where `projectID`, `userID`, `setID`, `url` are replaced by the values you determined above. Don't include the angle brackets. `bundle exec rake subjects:import[projectID,userID,setID,url]`

You can track the progress of your import job in Sidekiq.

4.4 Step 3B - Create the rake task from Jenkins

This approach is easier, but slower, as it has to spin up a VM. It's inefficient if you have many CSVs to upload.

Log into Jenkins, and go the Run rake task job.

Click on `Build with parameters` on the left hand menu.

construct a string like this: `subjects:import[projectID,userID,setID,url]` and paste it into the `RAKE_TASK_NAME` box.

Make sure to select `panoptes-api` from the `INSTANCE_NAME` box. Click Build.

You can monitor your import task like any other Jenkins job, from the dashboard.

CHAPTER 5

How to query Panoptes via Ruby console

Sometimes you will want to get live information from the Panoptes backend that is not available from the APIs. For staff and those with the appropriate level of access, the following approach is possible:

First, find the IP of a Panoptes dump worker instance. You can do this by typing `lita aws ip panoptes dump production` on Slack. Pick an instance, such as `ec2-54-174-166-171.compute-1.amazonaws.com`.

Now SSH into that instance (you'll need to be on an authorised network or VPN and you'll need to have the key for access):

```
ssh -i /code/Production/keys/zooniverse_1.pem ubuntu@ec2-54-174-166-171.compute-1.amazonaws.com
```

Now step inside the docker container (you can validate the container name using `docker ps`):

```
docker exec -it panoptesdumpworker_panoptes_1 bash
```

Now start a ruby console:

```
bundle exec rails c
```

Now you can run commands using the domain model objects defined in the Ruby code.

IMPORTANT: This gives you superuser access, with no constraints. This should only be done for reading and querying, not for modifying data. Great care should be taken when running heavy queries and data manipulation using this "API-bypassing" approach is strongly discouraged.

Here is an example of using this Ruby console to find an accurate count of retired subjects so far for a specific project, given the project ID (in this case `988`) and workflow ID (in this case `512`):

```
p = Project.find(988)
w = Workflow.find(512)
sws = SubjectWorkflowStatus.retired.where(workflow_id: w.id)
sws = sws.where("subject_workflow_counts.created_at >= ?", p.launch_date)
sws.count
```

CHAPTER 6

The Platform

The latest Zooniverse platform consists of a number of services working together.

- **Panoptes** is the main API endpoint in our platform. This is where user logins happen, and what stores data about projects, workflows, subjects, and classifications.
- **Cellect** is a service that selects a few randomly-chosen subjects for a user to work on next. Inside Panoptes, each user has a queue of subjects, and when that starts getting empty, Panoptes will query Cellect for some more subjects to replenish the queue with.
- **Nero** is an auxilliary service that deals with subject retirement rules. Panoptes itself comes with a very simple threshold on the number of classifications per subject. Nero is where we implement any project-specific retirement rules (although obviously we try and generalise when possible).
- **ZooEventStats** gathers metrics on everything in our platform.
- **Aggregation**

6.1 Services still under development

The following services are still being built and are not yet in active use:

- **Warehouse** will take over the exporting of classifications for a project.
- **Notifications** will maintain an open websocket (or long-polling) connection to browsers, so that we get a channel to push notifications to browsers directly.

CHAPTER 7

Client Libraries

7.1 Tools

- Generic command-line client (Python-based)
- Subject uploader (NodeJS-based)

7.2 API Wrappers

- Python
- Ruby
- JavaScript