Setup

- Turn on Caffeine.
- Turn on Do Not Disturb.
- Open Chrome Guest Window from user portrait.
- In Visual Studio Code, open `bats.py` and navigate to line 116.
- Open integrated terminal with `control+backtick` .
- `cd rachel-src/birbybot/slides`
- `reveal-md slides.md`
- `Control+shift+backtick` to open new integrated terminal.
- `cd rachel-src/birbybot/`
- `source start.sh`
- `Command+k` to clear terminal.
- Quit unneeded programs.
- Maximize Chrome and `cmd+shift+f` to hide address bar.
- Vocal warmup: "boots n' cats n' boots n' cats"
- Face stretch: mouse scrunch, warrior tongue

Slide 0

Building a Twitter Bot with Flickr and GCP

Hello, what's up GDG San Francisco? How's everyone's DevFest going so far?

My name is Rachel Ramsay, and today I'll be sharing what I learned while building a Twitter image bot with the Flickr API and Google Cloud Platform, particularly the Cloud Vision API.

I should warn you that I'm on call right now, so apologies in advance if I get paged.

Slide 1

Meet your bird nerd

I'm rachelbuilds on Twitter, and as it says in my bio, I care about birds the normal amount.

My journey into bot-making began when I tweeted a simple question:

Slide 2

A Simple Question

"What if I made a bot that exclusively posted pictures of plovers and their babies?"

(If you don't know what a plover is, it's a shore bird, and its babies are adorable.)

Slide 3

@beachbirbys

Thus was born beachbirbys, the Twitter account for all your beach bird baby needs.

Today, I'm going to tell you what I learned while building beachbirbys and share some resources for building your own bot.

I hope that you will leave this talk with the inspiration and guidance to build something that makes you feel good.

Slide 4

A Simple Plan

My goals for this bot were simple:

1. Get pictures of plover babies.
2. Post those pictures to Twitter.
3. Feel the good chemical spraying everywhere in my brain.

It turned out to be not quite that simple.

Slide 5

Problem #1

When I started this project, I had to decide where I'd get my plover pictures from, and that meant I had to decide how I would handle copyrighted material.

Some people's approach is more blasé — they figure they'll get some DMCA takedowns, and if they get enough strikes, their bot will be suspended; but, eh, whatever happens, happens.

But, for me, not reposting copyrighted images and crediting photographers were priorities.

As a follower of image bots myself, I'm often frustrated by not knowing how to find more work by whoever created what I'm seeing, so I wanted my bot to link back to its sources.

Slide 6

Solution #1

I chose Flickr as my image source because of its strong support for attribution and Creative Commons licenses.

The downside of this is that Flickr's API is idiosyncratic. It was initially released in 2010, it's and not exactly RESTful.

Luckily, there's a good Python client library for Flickr.

Slide 7

Searching Flickr

So, with the goal of seeing more pictures of baby plovers on my timeline, I said to Flickr, "Show me baby plovers."

And what Flickr showed me was...

Slide 8

That's not a bird...

A baby turtle.

Flickr's API *did* return baby plovers, but it also returned this turtle. As cute as this turtle is, I don't want my bot to post it.

So, why did it appear in my results?

Slide 9

[Read] Flickr returns photos that contain the search term in their title, description, or tags

This image is titled "*Baby* bog turtle," and its description includes the word "plover".

Slide 10

...threatened and endangered species that occur in Connecticut, including the threatened bog turtle, piping plover, and Puritan tiger beetle...

If you were building a tiger image bot, this turtle might appear in those search results, too!

So, now we have a problem.

Slide 11

Problem #2

How do I make sure that my bot only tweets bird photos?

There wasn't a good solution within the Flickr API to make sure that my bot never tweeted a non-bird photo by accident. It *is* possible to sort results by relevance, but then how do I know when results stop being relevant?

To be fair, there weren't *that* many search results. I *could* have sorted the birds from the non-birds by hand.

But I didn't do that, because when given a choice between doing the thing myself and spending many hours making the computer do the thing, I will always choose the latter.

Slide 12

Cloud Vision API

Ultimately, I decided to take all of the results returned by Flickr and run them through Google's Cloud Vision API.

Cloud Vision is great because it already knows what a bird is; it already knows what a lot of things are because

its image recognition models are already trained. Using it, I can benefit from machine learning without building the machine myself.

My bot leverages two Cloud Vision features: Label Detection and Object Localization.

Slide 13

Approach #1

Label detection evaluates the image as a whole.

When we run our baby box turtle through label detection, we know it's not a bird.

Slide 14

Definitely not a bird

Those numbers are scores showing how confident the model is that the label accurately reflects the image.

But because label detection evaluates the image as a whole, it can miss things.

Slide 15

But this is a bird

For this image, label detection only sees sand; it misses the adorable piping plover chick that I can plainly see with my human eyes.

In order for Cloud Vision to see the plover, I turned to Object Localization.

Slide 16

Approach #2

In addition to detecting objects, object localization does some rudimentary categorization.

Slide 17

Well, it's definitely something...

Sometimes object localization *does* recognize an object as a bird, but for our photo of a baby plover on the beach, it was like, "Eh, I think we got an animal here."

But I can use that bounding poly to crop the image and run it through label detection again.

Slide 18

Solution #2

When I do *that*, label detection recognizes the plover as a bird, which is good enough for my bot to use it.

That said,

Slide 19

Problem #3

Detecting and labeling objects is not bulletproof.

It's a good strategy, but it can still fail.

Slide 20

Doesn't look like anything to me

Unfortunately, nature has this thing called camouflage.

While my human eyes can pick out the spotted sandpiper hatchlings, Cloud Vision is only vaguely sensing the presence of fauna.

Slide 21

After cropping

When I crop this time, I actually get further away from correct classification: now Cloud Vision detects no animal presence at all; we've lost that fauna label.

Slide 22

Cloud AutoML Vision

Enter Cloud AutoML for Vision, which is designed to simplify training your own image classification models.

Slide 23

Solution #3?

I have not yet trained a model to distinguish camouflaged fauna from flora.

After all, I used the Cloud Vision API precisely because I didn't want to train my own models.

Given the scope of this project, I think I've reached a good-enough solution.

(Until I decide that, actually, I *am* interested in learning to train my own models.

After all, part of the fun of side projects is going down rabbit holes.)

Slide 24

Birds, But Make It Spooky

That's enough theory; let me show you the code.

In this demo, we're going to build up a simple script that will find and tweet photos of bats because it's almost Halloween!

[Cmd+tab to switch to Visual Studio Code.]

Demo

This script will:

1. Search Flickr for bat photos.
2. Classify each search result as either bat or non-bat.
3. Store image data in Cloud Datastore, a NoSQL database.

*[Control+g **116** Enter]*

The first thing we're doing is instantiating our clients (Flickr, Twitter, Cloud Datastore, and Cloud Vision) and searching Flickr for bats *[line 132]*.

The license parameter is selecting various Creative Commons licenses...

We've got safe search turned on, for what it's worth...

And I've picked dates that I hope will surface some good cases.

Then we'll take these results and use them to create entities for Cloud Datastore *[line 147]*.

Cloud Datastore is a NoSQL database with data objects called entities. Entities have Keys made up of a Kind and a Name. We're going to call this kind of entity Photos *[line 150]*, and we're going to make the name part of the key their source API plus the UUID from that API.

Everything returned by Flickr, we dump into this entity, then we add a few extra properties.

For example, `is_classified` allows us the option of splitting our image search and image classification steps.

Flickr has a few different image sizes, but they were introduced at different times, so not all photos come in all sizes — that's why we have a helper function to pick the best download url available.

Entities of the same kind don't need to have the same properties, and an entity's values for a given property don't need to be the same data type. This gives me the flexibility to add results from different image APIs later on: those Photo entities will have different properties, and that's okay.

*[Control+g **303** Enter]*

So, we'll get our search results for the word "bat," then we'll turn those results into entities.

We're also going to open our photos so that we know what it is we have to classify.

I'm also going to log the Vision API response so that we can reference it while going over our classifier function.

Once we have our entities, we *could* write them to the datastore, then come back later and ask the datastore for entities where is_classified is False, but for today, we'll classify them first and write them after that.

Your code block should look like

```
    photos = search_flickr("bat")
    entities = photos_to_entities(photos)

    show_photos(entities)
    classify_as_resp_only(["bat"], entities)
```

[Command+s to save.]

[Control+backtick to switch focus to terminal: `python bats.py` *]*

Flickr gave us some cute bats, but it also gave us some non-bats. That's why we need Cloud Vision!

So, our label annotations are: *[read labels]*.

And our localized object annotations are: *[read names]*.

We don't know we have a bat yet!

[Control+backtick to switch focus to editor.]

[Command+/ to comment out `show photos` *and* `classify as resp only` *]*

[Uncomment `classify as` *and* `with ds client` *block.]*

Your code block should look like

```
    photos = search_flickr("bat")
    entities = photos_to_entities(photos)

    # show_photos(entities)
    # classify_as_resp_only(["bat"], entities[3:6])

    classify_as(["bat"], entities)
    with ds_client.batch():
        ds_client.put_multi(entities)
```

[Command+s to save.]

During this step, we're going to classify our Flickr results and write them to the datastore.

*[Control+g **182** Enter]*

This function, `classify as` , iterates over the entities. For each entity, it downloads the image from Flickr and puts that image on a request to detect labels and localize objects *[line 197]*.

Hopefully, we get back label annotations and object localizations. Sometimes the response does not include the features we asked for — especially object localizations because sometimes the API can't detect any objects in the image — hence the if blocks *[line 209]*.

We take any labels and object names that we get and put them in a set.

If we *did* get localized object annotations, then we have those bounding polys. Most Cloud Vision API features return verticies in pixels, but for some reason the object localization feature returns verticies that have been normalized between zero and 1. With a little math, we can transform those normalized verticies into coordinates that we can crop to later.

Now we come to our helper function, `is_a` *[line 223]*. It takes a list of terms — in our case, we'll pass a list with only one element, the word "bat" — as well as a collection of labels. `is_a` checks if "bat" is in that collection of labels and returns a boolean.

So, if it's not a bat, but we did get back crop coordinates, we're going to:

- crop the image to just that object *[line 227]*
- convert the cropped image from a Pillow Image type to a Vision Image type (PIL stands for Python Image Library) *[line 232]*
- and then make a new request to the Vision API to label our cropped image *[line 235]*

Hopefully, we get back new labels *[line 240]*. Then we:

- add them to our set
- check if we've found a bat yet
- if we have found a bat, we won't crop to the rest of the objects. We don't want to call the API if we don't have to.

Now that we've done what we can to identify the image as either bat or non-bat, we update the entity *[line 249]* and mark it as having been classified.

Moment of truth...

[Control+backtick to switch focus to terminal: `python bats.py` *]*

Great, we found our bat(s).

Time to tweet!

[Control+backtick to switch focus to editor.]

*[Control+g **303** Enter]*

*[Comment out **everything**]*

[Uncommon below `Get bats...`]

We're going to ask the datastore for all the bat photos. Then we're going to choose one and tweet it. For us, loading a few entities into memory is not a big deal, but we could also make the query keys-only, then randomly choose the key and get just that entitity.

Your code block should look like this

```
    # photos = search_flickr("bat")
    # entities = photos_to_entities(photos)

    # show_photos(entities)
    # classify_as_resp_only(["bat"], entities)

    # classify_as(["bat"], entities)
    # with ds_client.batch():
        # ds_client.put_multi(entities)

    # Get bats from Cloud Datastore.
    query = ds_client.query(kind="Photo")
    query.add_filter("is_bat", "=", True)
    bats = list(query.fetch())

    # Time to tweet!
    bat = random.choice(bats)
    tweet_photo_entity(bat)
```

[Command+s to save.]

*[Control+g **266** Enter]*

First, we're gonna write a message.

We're gonna include the title, we're gonna credit the photographer, we're gonna link back to the Flickr page...

What's the hashtag for this event? `#devfest18` ? *[If possible, add after `#HappyHalloween` .]*

Tweeting a photo is actually a two-step process:

- First, we upload the photo to Twitter *[line 281]*.
- Then, we attach it plus our message to our tweet *[line 283]*.

Finally, we'll use the response from Twitter to record when we tweeted in the datastore *[line 294]*. We can use this property with a composite index to only pull bat photos that we haven't tweeted recently — that way, our bot won't repeat itself too often.

All right, let's tweet!

[Command+s to save.]

[Control+backtick to switch focus to terminal: `python bats.py` *]*

[Command+tab to Chrome. Next slide!]

Let's see how it looks...

Slide 25

@beachbirbys

[Command-click link to open in new tab, command-2 to switch to that tab.]

Awesome! Spooky!

[Command-1 to switch back.]

Slide 26

Composite Indexes

I'd like to shoutout the engineers who wrote an error message that tells you how to fix itself!

If I want to ask the datastore for photos of birds — or bats — that I haven't tweeted recently, I need to create a composite index.

Adding composite indexes to your datastore is easy: you define your indexes in a YAML file and deploy it with the gcloud tool. My YAML file looks just like what this helpful error message suggested!

Slide 27

Problem #4

Now I can find photos, classify them, and tweet them, but where am I going to host my scripts?

Slide 28

Solution #4

I'm currently hosting my bot at PythonAnyhwere.com, which is a great online IDE and web hosting service. If you use Python, check it out! The free tier includes one cron job!

Slide 29

What do you care about the normal amount?

But enough about me. What about you?

Maybe you're not obsessed with birds (I mean, maybe you don't care about birds the normal amount).

Slide 30

What do you want to see?

But I bet there *is* something that you *do* want to see more of.

Slide 31

What are you going to build?

So, what are you going to build?

And how are you going to build it?

Slide 32

Image Sources

This is a non-exhaustive list of APIs that you can use to find Creative Commons-licensed images.

Unsplash is a particularly good source for aesthetic photos — maybe I'll build a hipster bot next! Their license is incredibly permissive: you can use Unsplash photos in commercial projects without credit; you just can't scrape their photos to create your own Unsplash.

Slide 33

Cloud Vision API Features

There's way more to Cloud Vision than Label Detection and Object Localization.

Think about what you can build with this feature set:

- You can detect – but not *recognize* – faces, including the emotions displayed on those faces.
- You can detect landmarks in an image, such as the Golden Gate Bridge.
- There's also logo detection.
- Or if you have a photo of a storefront, you can read the name of the store from its sign.
- There's a more robust OCR endpoint for documents.
- Safe Search Detection is crucial for making sure that your bot doesn't tweet anything gory or racy.
- Image Properties can tell you the dominant colors in an image, which would come in handy if you were building something centered around a particular color aesthetic.
- Crop Hints can help you format images for different aspect ratios suited to different social media platforms.
- Web Detection can do things like find websites that use an image, or give you URLs to visually similar images. So, it's like reverse image search.

But what if you've already pivoted to video?

Slide 34

What about video?

Flickr can return not just photos but also video, so if you are dedicated to tweeting only the choicest cute animal *videos*, there are Cloud Platform APIs to help with that, too!

Cloud Video Intelligence has a label detection feature that's similar to Cloud Vision's.

There's also Explicit Content Detection, which is like Safe Search Detection.

One of the challenges of working with video is programatically detecting "good clips" — you know, visually interesting, not too long. Cloud Video Intelligence has a Shot Change Detection feature, which can help you roll your own clipping strategy.

Speech Transcription is currently in beta.

You can combine those transcripts with the Natual Language API, or the Translation API.

Slide 35

More GCP

One of the reasons why I used Cloud Datastore was that I was already on Google Cloud Platform due to Cloud Vision. It's worth exploring the rest of what Google has to offer.

You may have noticed that our bats-to-Twitter script downloaded photos locally. That's not required -- you could get the image from its URL every time you wanted to work with it, or you can always store images with Cloud Storage.

As for hosting your scripts, App Engine has a cron service, while Cloud Scheduler was announced back in July and is currently in alpha.

Slide 36

Twitter

At the end of July, Twitter started requiring developers to answer a questionnaire about how they plan to use Twitter's API and its data before permitting developers to create an API key.

Currently, Twitter asks four questions:

1. What is the core use case, intent, or purpose for your use of Twitter's APIs?
2. Do you intend to analyze Tweets, Twitter users, or their content? If so, share details about how you'll do this.
3. Does your use case involve Tweeting, Retweeting, or liking content? If so, share how you will interact with Twitter users or their content.
4. How will Twitter data be displayed to users of your solution? Will individual Tweets and Twitter content be displayed, or will information about Tweets or Twitter content be displayed in aggregate?

This *is* a hoop to jump through, but you can do it.

(Also, Flickr already had a similar requirement, so I had my "Please just let me post cute animal photos" spiel ready to go.)

But maybe you're done with Twitter...

Slide 37

Mastodon

Mastodon is an open source, decentralized social network. It has a few nice design features — for example, you can't search arbitrary strings, so while you can still find posts via hashtags, it's harder for people to search for something they want to fight about.

Plus, there's a Mastodon instance just for bots called Bots. In. Spaaaace!

(My next mini-project will probably be updating beachbirbys to crosspost to Mastodon.)

In addition to Twitter and Mastodon, Tumblr is a visually-focused social network with an API, so that's a possibility for you, too.

Slide 38

Build what makes you smile

Thank you for listening. I hope that you feel inspired to mash together some APIs to create something that makes you smile.

One of my favorite parts of being on Twitter is seeing what other people are making. sailorhg led a technical zine–making workshop at Write/Speak/Code, and I absolutely loved seeing everyone post their efforts to Twitter.

So, if you do build a Twitter bot, please at me and let me know!

Slide 39

Rachel Ramsay

I am Rachel Ramsay, and I care about birds the normal amount.

You can find me on Twitter at rachelbuilds, and you can find today's slides and code on my GitHub.

Slide 40

Photo Credits

Thanks to the photographers who took these turtle, bird, and bat photos!