Data Migrations in the App Engine Datastore

Ryan Morlok

Co-Founder at Docalytics
@rmorlok
https://plus.google.com/u/0/+RyanMorlok
http://www.linkedin.com/in/ryanmorlok/

This Talk is in Python

Examples Use NDB

```python
from google.appengine.ext import ndb
from google.appengine.ext import db
```

The Datastore

- Schemaless

- Entities of the same type can have different properties

- Most applications express an effective schema using application code

- All queries are served by pre-built indexes

Problem

- No general framework for making mass updates ("schema changes") to entities in the Datastore or helping the in-code model evolve

- Frameworks like Rails and Django (South) have tools to help manage this and are built on top of relational databases that allow SQL to be used to help migrate data

- We will look at techniques for doing this on the non-relational App Engine Datastore

Initial Data Model

```python
from google.appengine.ext import ndb


class BlogPost(ndb.Model):
    title = ndb.StringProperty(required=True)
    blurb = ndb.StringProperty(required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    published = ndb.DateTimeProperty(auto_now_add=True, required=True)


class Comment(ndb.Model):
    blog_post = ndb.KeyProperty(kind=BlogPost, required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    timestamp = ndb.DateTimeProperty(required=True)
```

Add Likes

- Each comment has a count of "likes"

- Displayed with each comment

- No searching/sorting/etc

- Don't care who liked what

Just Add Property with Default Value

```python
class Comment(ndb.Model):
    blog_post = ndb.KeyProperty(kind=BlogPost, required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    timestamp = ndb.DateTimeProperty(required=True)
    likes = ndb.IntegerProperty(default=0)
```

Adding a Post URL Slug

- Want to change URLs from
  `http://example.com/posts/1234`
  to
  `http://example.com/posts/this-is-my-post`

- Need to be able to query Datastore to locate the right post for a given slug

- All Posts need to have slugs populated before we start using them for URL lookups

- Slugs are determined by the title of the post

Steps
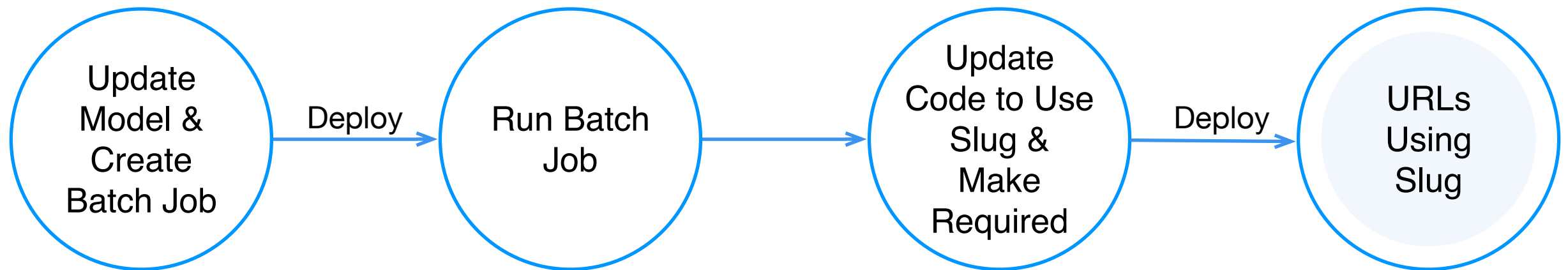
Add Slug to Model

```python
class BlogPost(ndb.Model):
    title = ndb.StringProperty(required=True)
    blurb = ndb.StringProperty(required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    published = ndb.DateTimeProperty(auto_now_add=True, required=True)
    slug = ndb.StringProperty()
```

Create Slugs for All Posts
Deferred Tasks

```python
BATCH_SIZE = 10

def add_slugs(cursor=None):
    posts, next_cursor, is_there_more \
        = BlogPost.query().fetch_page(BATCH_SIZE, start_cursor=cursor)

    for p in posts:
        p.slug = BlogPost.slug_from_title(p.title)

    if len(posts) > 0:
        ndb.put_multi(posts)

    if is_there_more:
        deferred.defer(add_slugs, cursor=next_cursor)
```

Create Slugs for All Posts MapReduce

```python
from mapreduce import base_handler, mapper_pipeline, operation

def create_slug_mapper(post):
    post.slug = BlogPost.slug_from_title(post.title)
    yield operation.db.Put(post)

class CreateSlugsPipeline(base_handler.PipelineBase):
    def run(self):
        yield mapper_pipeline.MapperPipeline(
            job_name="create slug",
            handler_spec="module.create_slug_mapper",
            input_reader_spec=\
                "mapreduce.input_readers.DatastoreInputReader",
            params={
                "entity_kind": "models.BlogPost"
            },
            shards=16)
```

Create Slugs for All Posts
MapReduce

```
pipeline = module.CreateSlugsPipeline()
pipeline.start()
```

Add Number of Comments for Post

- When displaying the preview of a blog post, show the number of comments

- Cannot query for count for pages of multiple posts; too slow

- Need to de-normalize the data

- Need to go through and do computation for all existing posts

Add Comment Count to Model

```python
class BlogPost(ndb.Model):
    title = ndb.StringProperty(required=True)
    blurb = ndb.StringProperty(required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    published = ndb.DateTimeProperty(auto_now_add=True, required=True)
    slug = ndb.StringProperty(required=True)
    number_of_comments = ndb.IntegerProperty(default=0)
```

Count Comments for All Posts Deferred Tasks

```python
BATCH_SIZE = 10

@ndb.toplevel
def count_comments(cursor=None):
    posts, next_cursor, is_there_more \
        = BlogPost.query().fetch_page(BATCH_SIZE, start_cursor=cursor)

    for p in posts:
        p.number_of_comments \
          = Comment.query(Comment.blog_post == p.key).count(limit=1000)
        p.put_async()

    if is_there_more:
        deferred.defer(count_comments, cursor=next_cursor)
```

Count Comments for All Posts MapReduce

```python
from mapreduce import base_handler, operation, \
    mapreduce_pipeline

def count_comments_mapper(comment):
    yield (comment.blog_post.urlsafe(), "")



def count_comments_reducer(keystring, values):
    post = ndb.Key(urlsafe=keystring).get()
    post.number_of_comments = len(values)
    yield operation.db.Put(post)
```

Count Comments for All Posts MapReduce

```python
class CountCommentsPipeline(base_handler.PipelineBase):
    def run(self):
        yield mapreduce_pipeline.MapreducePipeline(
            job_name="count comments",
            mapper_spec="module.count_comments_mapper",
            reducer_spec="module.count_comments_reducer",
            input_reader_spec=\
                    "module.DatastoreInputReader",
            mapper_params={
                "entity_kind": "models.Comment"
            },
            shards=16)
```

Deleting Properties from Models

- Removing a property from your model class doesn't change the data in the DataStore

- May want to delete data to reduce entity size, keep entities consistent, etc

```python
class Cat(ndb.Model):
    tail = ndb.StringProperty()
    wiskers = ndb.StringProperty()
    skin = ndb.StringProperty()
```

Deleting a Property - NDB

```python
c = Cat.get_by_id(1234)

if 'skin' in c._properties:
    del c._properties['skin']
    c.put()
```

- Delete the property from the _properties dictionary

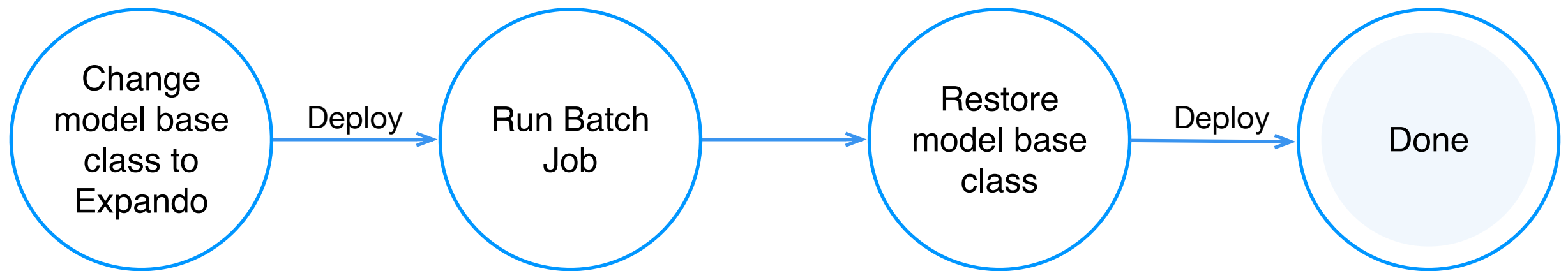- Note that `del cat.skin` would set the property to `None`

Deleting a Property - DB

- Model objects have a _properties dictionary, but it cannot be used to delete a properties

- Two approaches: switch model to Expando or direct Datastore access

Deleting a Property - DB Expando

```python
class Cat(db.Expando):
    pass

c = Cat.get_by_id(12345)

del c.skin
c.put()
```

Deleting a Property - DB Expando

Change model base class to Expando →(Deploy)→ Run Batch Job → Restore model base class →(Deploy)→ Done

Downsides:

- Must change base class; problematic if using custom base class with logic

- Requires two deploys

Deleting a Property - DB Direct Datastore Access

```python
from google.appengine.api import datastore
from google.appengine.api import datastore_errors

def get_entities(keys):
    rpc = datastore.GetRpcFromKwargs({})
    keys, multiple = datastore.NormalizeAndTypeCheckKeys(keys)
    entities = None
    try:
        entities = datastore.Get(keys, rpc=rpc)
    except datastore_errors.EntityNotFoundError:
        assert not multiple

    return entities

def put_entities(entities):
    rpc = datastore.GetRpcFromKwargs({})
    keys = datastore.Put(entities, rpc=rpc)
    return keys
```
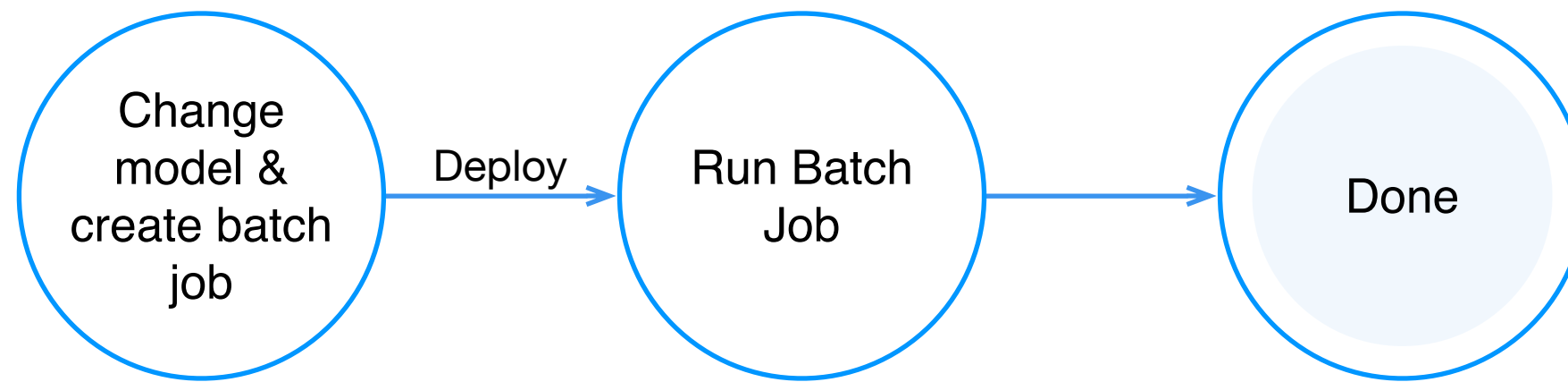
Deleting a Property - DB
Direct Datastore Access

```python
key = db.Key.from_path('Cat', 12345)
c = get_entities([key])[0]

if 'skin' in c:
    del c['skin']
    put_entities([c])
```

Deleting a Property - DB Direct Datastore Access

```
 ( Change          Deploy    ( Run Batch              ( Done )
   model &        ─────────→    Job       )──────────→
   create batch )
   job )
```

Advantages:

- Does not require multiple deploys which can be hard to automate across multiple environments & developers

- Does not interfere with the base class of the model

Renaming Models

- Approach depends on your need

- If you just want to keep the code clean, alias the model in code

  - Be cautious: GQL and other textual references to the model for queries will need to use name in Datastore

- If you want the underlying entities renamed, need to create new entities

  - Problematic if there are keys to the entity or child entities

Rename Blog Post to Article NDB

```python
class Article(ndb.Model):

    @classmethod
    def _get_kind(cls):
        return 'BlogPost'

    title = ndb.StringProperty()
    blurb = ndb.StringProperty()
    content = ndb.StringProperty()
    published = ndb.DateTimeProperty()
    slug = ndb.StringProperty()
    number_of_comments = ndb.IntegerProperty()
```

Rename Blog Post to Article DB

```python
class Article(db.Model):

    @classmethod
    def kind(cls):
      return 'BlogPost'

    title = db.StringProperty()
    blurb = db.StringProperty()
    content = db.StringProperty()
    published = db.DateTimeProperty()
    slug = db.StringProperty()
    number_of_comments = db.IntegerProperty()
```

Renaming Fields

- Approach depends on your need

- If you just want to keep the code clean, alias the model in code

  - Be cautious: GQL and other textual references to the field for queries will need to use name in Datastore

- If you want the underlying fields on the entities renamed, need multi-step migration

Alias url_slug to slug

NDB

```
url_slug = ndb.StringProperty(name='slug')
```

DB

```
url_slug = db.StringProperty(name='slug')
```

Mind Your Indexes

- Make sure added/renamed fields are added index.yaml

- Delete old Indexes that aren't needed

Final Data Model

```python
from google.appengine.ext import ndb


class BlogPost(ndb.Model):
    title = ndb.StringProperty(required=True)
    blurb = ndb.StringProperty(required=True)
    slug = ndb.StringProperty(required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    published = ndb.DateTimeProperty(auto_now_add=True, required=True)
    number_of_comments = ndb.IntegerProperty(default=0)


class Comment(ndb.Model):
    blog_post = ndb.KeyProperty(kind=BlogPost, required=True)
    content = ndb.StringProperty(required=True, indexed=False)
    timestamp = ndb.DateTimeProperty(required=True)
    likes = ndb.IntegerProperty(default=0)
```

Questions?