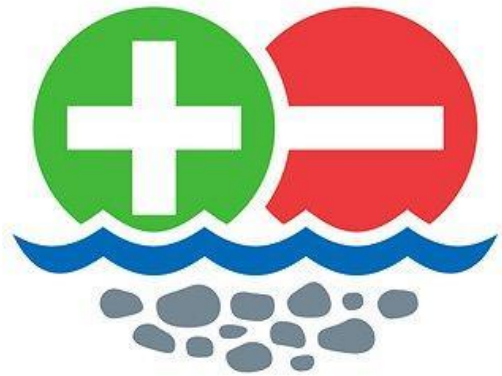
# GAE: плюсы, минусы, подводные камни




- Я - Егор Назаркин
- Обычно - пишу на Python под GAE (и не только)
- Email: nimnull@gmail.com
- Twitter: @nimnull


http://www.slideshare.net/nimnull/gae-10292539

# GAE - что за неведомая фигня?

# GAE это...

BigTable

Cloud Platform

Go

JAVA

**Webapp hosting**

**PaaS**

automatic scaling

Google managed

**Python**

# GAE это...

Игровая площадка Гвидо

# Ограничения GAE

- Время выполнения запроса - 60 секунд (лучше - 30)
- Данные на FS - только чтение
- Сохранение файлов - Google Datastore
- Связь с внешними хостами - только AppEngine URL Fetch

Квоты:

- Безопасность
- Платные
- OverQuotaError

http://www.google.com/enterprise/cloud/appengine/pricing.html

python™

## Алиасы модулей:

cPickle => pickle
cStringIO => StringIO
marshal, imp, ftplib, tempfile => None

## Библиотеки:

- logging
- Django 1.2
- jinja2 2.6
- lxml 2.3
- MarkupSafe 0.15
- NumPy 1.6.1
- PyCrypto 2.3
- PyYAML 3.10
- Python Imaging Library (PIL) 1.1.7
- setuptools 0.6c11
- webapp2 2.3
- WebOb 1.1.

# MVC?

view

model

controller

# Backends

нет ограничений на время обработки

ручная конфигурация памяти/CPU

собственный поддомен

резидентность

приватные по умолчанию
API доступа

# Datastore

ORM

GQL

Schemaless

Транзакции

Метаданные

Статистика

Асинхронность

# ORM

```
class Rate(db.Model):
    price = db.IntegerProperty()
    user = db.UserProperty(required=True)
    uptime = db.DateTimeProperty(auto_now=True)
    trade_unit = db.ReferenceProperty(TradeUnit, required=False)

    def put(self):
        q = Rate.all().filter('trade_unit = ', self.trade_unit.key())
        q.order("-price")
        previous_rate = q.get()
        previous_price = previous_rate.price if previous_rate else 0
        price = previous_price + self.trade_unit.increase_rate

        if self.price < price:
            self.price = price

        super(Rate, self).put()
```

# GQL

```
SELECT [* | __key__]
  FROM <kind>
  WHERE <condition> [AND <condition> ...]
  ORDER BY <property> [ASC | DESC ] [, <condition> [ASC | DESC]... ]
  LIMIT [<offset>, ] <count> ]
  OFFSET <offset>
```

- DATETIME(*year*, *month*, *day*, *hour*, *minute*, *second*)
- DATE(*year*, *month*, *day*)
- TIME(*hour*, *minute*, *second*)
- KEY ('encoded key')
- USER ('email address')
- GEOPT ('lat', 'long')

# Transactions

```
class Rate(db.Model):
    price = db.IntegerProperty(default=0)
    user = db.UserProperty(required=True)
    uptime = db.DateTimeProperty(auto_now=True)
    trade_unit = db.ReferenceProperty(TradeUnit, required=False)


    def put_with_next_price(self, price=None):

def in_transaction(rate, next_price=None):
q = Rate.all().filter('trade_unit = ', rate.trade_unit.key())
q.order("-price")
previous_rate = q.get()
previous_price = previous_rate.price if previous_rate else next_price
price = previous_price + rate.trade_unit.increase_rate
            if rate.price < price:
                rate.price = price
            rate.put()

        db.run_in_transaction(in_transaction, self, price)
```

# Entities and Keys

Key:

Kind - объединяет по признаку (classname)
ID - генерируется Datastore или задаётся вручную

*Key.from_path(kind, id_or_name, parent=none, namespace=None, **kwds)*

```
item = TradeUnit()
item.put()
rate = Rate(parent=item)
# ..............
rate = Rate(parent=item.key())
```

# Async

get
put
delete

*_async()

q = Model1.filter(...).run()

*# выполняем какую-то работу ...*

<object>.get_result()

for e in q:
    ...

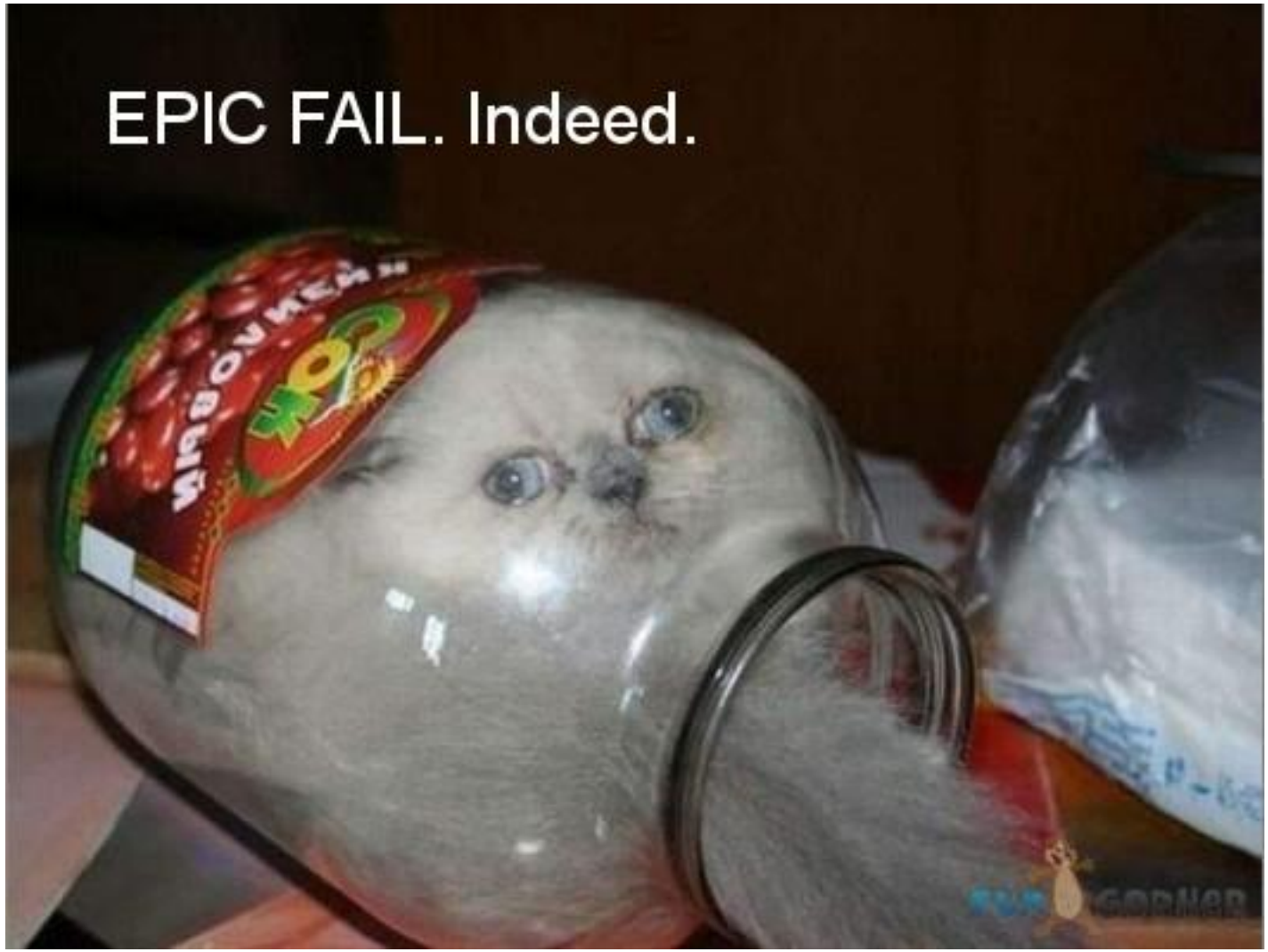
# NDB

- Расширенное описание полей
- Дополнительные типы полей db.Model
- Вменяемый синтаксис запросов
- Кэширование
- Триггеры (hooks)

```
class Contact(Model):
    name = StringProperty()
    addresses = StructuredProperty(Address, repeated=True)

guido = Contact(name='Guido', addresses=[
Address(type='home', city='Amsterdam'),
Address(type='work', city='SF', street='Spear St'),
])

qry = Article.query(Article.tags.IN(['python', 'GAE', 'ndb']))
```


EPIC FAIL. Indeed.

# Google Cloud SQL

*from google.appengine.api import rdbms*

Google SQL Service поддерживает DB-API 2.0

etc...

# Батарейки в комплекте:

Mail

**Task queues**

**Memcached**

Images

Message channels

**Cloud storage**

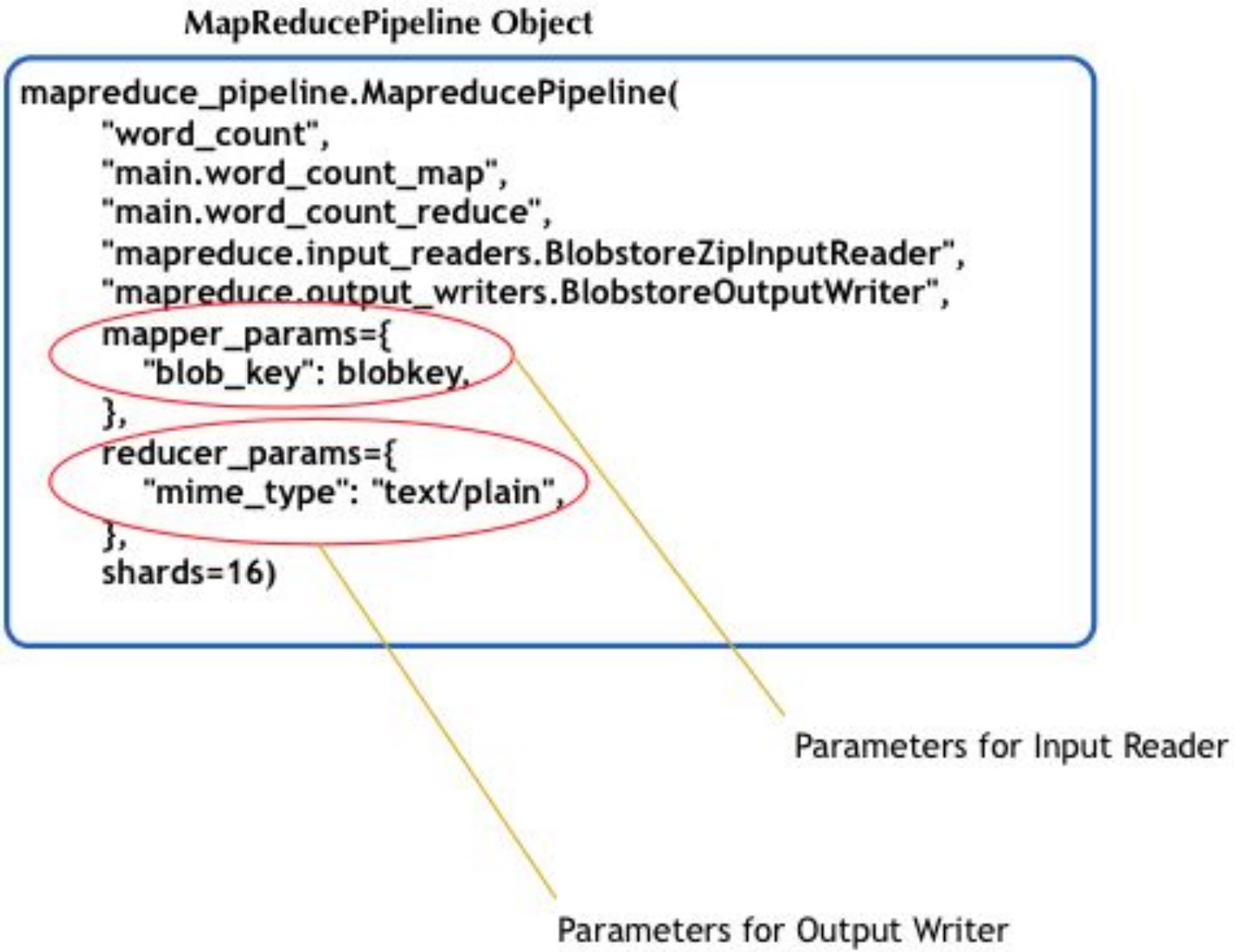
OAuth

URL fetch

**Blobstore**

MapReduce

# MapReduce:


MapReducePipeline Object
mapreduce_pipeline.MapreducePipeline(
"word_count",
"main.word_count_map",
"main.word_count_reduce",
"mapreduce.input_readers.BlobstoreZipInputReader",
"mapreduce.output_writers.BlobstoreOutputWriter",
mapper_params={
"blob_key": blobkey,
},
reducer_params={
"mime_type": "text/plain",
},
shards=16)
Parameters for Input Reader
Parameters for Output Writer

# Cloud storage:

Сервис общего назначения (используется в google)

AppEngine - RESTful API

Может использоваться разными приложениями

Trial-only для бесплатной подписки

file-like доступ

# Channel API:

*JS View (coffee-script sample)*

```
_sock = new goog.appengine.Channel @token
@socket = _sock.open()
# bind socket events
@socket.onopen = =>
  @subscribe options
@socket.onmessage = (evt) =>
  console.log "realtime: ", evt
@socket.onerror = (evt) =>
  @socket = null
@socket.onclose = (evt) =>
  [@clients, @socket, @token] = [{}, null, null]
```

*Python back-end:*

```
key = str(uuid.uuid4())

def subscribe(request):
    token = channel.create_channel(request.user.id + key)
    return {
        'token': token,
    }

def send_message(request):
    channel.send_message(request.user.id + key, 'message')
```

# На этом всё

# Вопросы?