GeoWatch Util Documentation

Release 1.0

GeoWatch Developers

September 13, 2016

Contents

Contents:

CHAPTER 1

Installation

1.1 New Installation

```
pip install git+git://github.com/geowatch/geowatch-util.git@master
```

1.2 Upgrade

```
pip install git+git://github.com/geowatch/geowatch-util.git@master --upgrade
```

CHAPTER 2

GeoWatch Util / Getting Started!

Welcome to the documentation for GeoWatch Util, the engine of GeoWatch. To install, follow the installation instructions.

2.1 What is GeoWatch?

GeoWatch is a spatially-enabled distributed message broker. GeoWatch includes a suite of libraries that streamline information flow across a heterogeneous set streaming and batch data sources.

The primary requirement of GeoWatch is to connect streaming GeoJSON services and traditional batch processing GIS system, and vice versa.

2.2 What does it do?

GeoWatch provies an abstraction layer on top of a heterogenous set of streaming and batch data sources. With a commonm API, GeoWatch is able to pass messages between streaming systems, buffer streaming messages for batch processing, send notifications, and generally enable system-to-system spatially-enabled communication.

2.3 What does it support?

GeoWatch supports multple channels and stores, including the following:

Streaming Channels

1. Apache Kafka - http://kafka.apache.org
2. AWS Kinesis - https://aws.amazon.com/kinesis/
3. AWS SNS - https://aws.amazon.com/sns/
4. AWS SQS - https://aws.amazon.com/sqs/
5. Slack - https://slack.com

Batch Stores

1. Memcached - http://memcached.org/
2. WFS - https://en.wikipedia.org/wiki/Web_Feature_Service
3. AWS S3 - https://aws.amazon.com/s3/

4. File

If using Django, consider using geowatch-django, too. GeoWatch Django provides a consistent way to provision geowatch objects based on Django settings. Therefore, you don't need to create your own settings.

Below is a list of simple patterns for provisioning consumers, producers, and stores by using the functions in *runtime.py* (`geowatchutil.runtime`). More complex provisioing routines can be executed by directly calling the respective factory.py functions *build_client*, *build_consumer*, *build_producer*, and *build_stores*.

Note: Multiple examples of GeoWatch in action are includec in the documentation. These examples provide brokers for multiple examples.

Note: Read a more in-depth explanation of the API.

Or skip directly to the source code documenation. Most public API functions are contained in `geowatchutil.runtime`.

2.4 How does it work?

GeoWatch is built on top of the following key concepts: consumers, producers, stores, brokers, codecs, and mappings.

Consumers

Consumers retrieve and decode messages from an external channel, such as AWS Kinesis. GeoWatch can then send those messages into another streaming channel using producers or into persistent storage using stores.

Currently supported channels include: Apache Kafka, AWS Kinesis, and AWS SQS.

Consumers user the runtime method provision_consumer.

Producers

Producers generate messages. GeoWatch has a variety of codec for messages, such as plain text, list/tab-separated-values, json/dict, GeoJSON, etc. Producer's will add messages to a remote channel, which could be a real "streaming" service such as AWS kinesis or a "one-way" channel such as Slack notifications.

Currently supported channels include: Apache Kafka, AWS Kinesis, AWS SQS, AWS SNS, and Slack.

Producers user the runtime method provision_producer.

Stores

Stores are used for persistant non-streaming storage of messages. They can include file, document, or memory storage. For example, dumping all message traffic to an AWS S3 object for long-term storage. Or buffering the latest batch of messages in memcached for analysis across a multi-tenant infrastructure.

Stores use the runtime method provision_store.

Brokers

Brokers provide an object-oriented way of managing the flow of messages. You can attach message consumers, message producers, and message stores to brokers.

Additionally, the GeoWatchBroker class can be extended to inject arbitray code directly into before/middle/after the message processing chain. For example, extending GeoWatchBroker to set up a complex cron job that parses message data and adds to MongoDB.

Codecs

Codecs convert messages between external and internal data representations. GeoWatch includes codecs for: plain text, JSON, GeoJSON, and WFS.

For example, for JSON, *encode* calls json.dumps(data) and *decode* calls json.loads(data).

Mappings

Mappings convert application-specific objects into dicts/json that are suitable for messaging. Templates are channel-specific, while mappings are for generic use (application <—> geowatch). Therefore, only the base class is in the geowatch-util library.

Note: See GeoWatch in action on the examples page.

CHAPTER 3

GeoWatch Util / API

The source code for `geowatchutil` is broken out into *runtime*, *factory*, and class functions.

See below for a quick explanation.

3.1 Runtime

Most public API functions are contained in `geowatchutil.runtime`. Runtime functions wrap broker, consumer, producer, and store functions with exception handling and multiple tries.

In most use cases, you should use `geowatchutil.runtime` rather than factory functions and certainly more than directly initializing a class.

Note: Unless for custom subclasses, you can describe your information flow as dicts, so that you only ever need to call `geowatchutil.runtime.provision_brokers()`.

Functions

The 4 main runtime functions are:

1. `geowatchutil.runtime.provision_brokers()`
2. `geowatchutil.runtime.provision_consumer()`
3. `geowatchutil.runtime.provision_producer()`
4. `geowatchutil.runtime.provision_store()`

3.2 Factory

Factory functions provide a single entry point for creating a broker, consumer, and producer, store, and codec, respectively.

Functions

The 5 main factory functions are:

1. `geowatchutil.broker.factory.build_broker()`
2. `geowatchutil.consumer.factory.build_consumer()`
3. `geowatchutil.producer.factory.build_producer()`

4. `geowatchutil.store.factory.build_store()`
5. `geowatchutil.codec.factory.build_codec()`

Modules

View all modules @ code.

Note: See GeoWatch in action on the examples page.

CHAPTER 4

GeoWatch Util / Examples

Below are a list of standard examples, configured as dicts and as direct code.

You can download these examples and more on GitHub.

4.1 AWS Kinesis to WFS

local_settings.py

```
GEOWATCH_BROKERS = [
    {
        "enabled": True,
        "name": "Kinesis to WFS",
        "description": "Incoming GeoJSON from AWS Kinesis to WFS",
        "consumers":
        [
            {
                "enabled": True,
                "backend": "kinesis",
                "codec": "json",
                "topic_prefix": "",
                "topic": "geowatch-geonode"
            }
        ],
        "stores_out":
        [
            {
                "enabled": True,
                "backend": "wfs",
                "key": None,
                "codec": "wfs",
                "options": {
                    "url": "http://localhost:8080/geoserver/wfs",
                    "auth_user": "admin",
                    "auth_password": "admin"
                }
            }
        ]
    }
]
```

geowatch-brokers.py

```
from multiprocessing import Process, Lock, Queue, cpu_count

from django.conf import settings

from geowatch.runtime import provision_brokers


def run(broker):
    broker.run()

verbose = True
brokers = provision_brokers(settings.GEOWATCH_BROKERS_CRON)

if not brokers:
    print "Could not provision brokers."
else:
    print str(cpu_count())+" CPUs are available."
    processes = []
    processID = 1
    for broker in brokers:
        process = Process(target=run,args=(broker,))
        process.start()
        processes.append(process)
        processID += 1

    print "Provisioned "+str(len(brokers))+" brokers."
```

4.2 Multiple Apache Kafka to AWS Kinesis

```
GEOWATCH_BROKERS = [
    {
        "enabled": True,
        "name": "Logs",
        "description": "Aggregate incoming logs from multiple Apache topics to AWS Kinesis",
        "consumers":
        [
            {
                "enabled": True,
                "backend": "kafka",
                "codec": "plain",
                "topic": "logs-master",
                "host": "localhost"
            },
            {
                "enabled": True,
                "backend": "kafka",
                "codec": "plain",
                "topic": "logs-worker",
                "host": "localhost"
            }
        ],
        "producers":
        [
            {
                "enabled": True,
                "backend": "kinesis",
```

```
                "codec": "wfs",
                "topic": "logs-aggregate",
                "aws_region"=XXX,
                "aws_access_key_id"=XXX,
                "aws_secret_access_key"=XXX
            }
        ]
    }
]
```

Apache Kafka

```
from geowatchutil.runtime import provision_consumer

client, consumer = provision_consumer(
    "kafka",
    host=None,
    topic=None,
    codec="plain",
    topic_prefix="",
    max_tries=12,
    timeout=5,
    sleep_period=5,
    topic_check=False,
    verbose=False):
```

AWS Kinesis

```
from geowatchutil.runtime import provision_consumer_kafka

client, consumer = provision_consumer(
    "kinesis",
    topic=None,
    codec="GeoWatchCodecPlain",
    aws_region=None,
    aws_access_key_id=None,
    aws_secret_access_key=None,
    shard_id='shardId-000000000000',
    shard_it_type="LATEST",
    client=None,
    topic_prefix="",
    max_tries=12,
    timeout=5,
    sleep_period=5,
    topic_check=False,
    verbose=False):
```

```
from geowatchutil.runtime import provision_producer

client, producer = provision_producer(
    backend,
    topic=None,
    codec="GeoWatchCodecPlain",
    path=None,
    host=None,
    aws_region=None,
    aws_access_key_id=None,
    aws_secret_access_key=None,
    client=None,
```

```
    topic_prefix="",
    max_tries=12,
    timeout=5,
    sleep_period=5,
    topic_check=False,
    verbose=False)
```

```
from geowatchutil.store.factory import provision_store

store_file = provision_store(
    "file",
    settings.STATS_REQUEST_FILE,
    "json",
    which="first")
```

S3 Store

```
from geowatchutil.store.factory import provision_store

store_s3 = provision_store(
    "s3"
    "final_stats.json",
    "json",
    aws_region=settings.AWS_REGION,
    aws_access_key_id=settings.AWS_ACCESS_KEY_ID,
    aws_secret_access_key=settings.AWS_SECRET_ACCESS_KEY,
    aws_bucket="tilejet",
    which="first")
```

Memcached Store

```
from geowatchutil.store.factory import provision_store

store_memcached = provision_store(
    "memcached",
    "stats.json",
    "json",
    client_type="umemcache",
    which="first",
    host="localhost",
    port=11211)
```

WFS Store

```
from geowatchutil.store.factory import provision_store

store_wfs = provision_store(
    "wfs",
    key,
    "wfs",
    url="http://geonode.org/geoserver/geonode/wfs/"
    auth_user="admin",
    auth_password="admin")
```

```
from geowatchutil.broker.base import GeoWatchBroker

broker = GeoWatchBroker(
    stores_out=stores_out,
    sleep_period=5,
```

```
    count=1,
    deduplicate=False,
    filter_last_one=False,
    timeout=5,
    verbose=True)

broker.run(max_cycle=1)  # loop once

broker.run()  # infinite loop
```

4.3 Kinesis to WFS

..code:: python

4.4 Kafka to Slack

CHAPTER 5

Indices and tables

- genindex
- modindex
- search