aws-xray-sdk Documentation

Release 2.12.0

Amazon Web Services

Aug 18, 2023

CONTENTS

This project is open sourced in Github. Please see: https://github.com/aws/aws-xray-sdk-python.

The AWS X-Ray service accepts application information in the form of trace segments. A trace segment represents the work done by a single machine as a part of the entire task or request. A set of trace segments which share the same trace ID form a trace. A trace represents a full unit of work completed for a single task or request. Learn more about AWS X-Ray service: https://aws.amazon.com/xray/.

The AWS X-Ray SDK for Python (the SDK) enables Python developers to record and emit information from within their applications to the AWS X-Ray service. You can get started in minutes using `pip` or by downloading a zip file.

Currently supported web frameworks and libraries:

- aioboto3/aiobotocore
- aiohttp >=2.3
- boto3/botocore
- Bottle
- Django >=1.10
- Flask
- httplib/http.client
- mysql-connector
- pg8000
- psycopg2
- pymongo
- pymysql
- pynamodb
- requests
- SQLAlchemy
- sqlite3

You must have the X-Ray daemon running to use the SDK. For information about installing and configuring the daemon see: http://docs.aws.amazon.com/xray/latest/devguide/xray-daemon.html.

Contents:

CHAPTER

ONE

BASIC USAGE

The SDK provides a global recorder, `xray_recorder`, to generate segments and subsegments.

1.1 Manually create segment/subsegment

If you're using a web framework or library that is not supported, or you want to define your own structure on segments/subsegments, you can manually create segments and subsegments by using code like the following:

```
from aws_xray_sdk.core import xray_recorder

xray_recorder.begin_segment('name')

# your code here

xray_recorder.begin_subsegment('name')
# some code block you want to record
xray_recorder.end_subsegment()

xray_recorder.end_segment()
```

The `xray_recorder` keeps one segment per thread. Therefore, in manual mode, call `xray_recorder.end_segment()` before creating a new segment, otherwise the new segment overwrites the existing one. To trace a particular code block inside a segment, use a subsegment. If you open a new subsegment while there is already an open subsegment, the new subsegment becomes the child of the existing subsegment.

1.2 Decorator for function auto-capture

A decorator is provided to easily capture basic information as a subsegment on user defined functions. You can use the decorator like the following:

```
@xray_recorder.capture('name')
def my_func():
    #do something
```

`xray_recorder` generates a subsegment for the decorated function, where the name is optional. If the name argument is not provided, the function name is used as the subsegment name. If the function is called without an open segment in the context storage, the subsegment is discarded. Currently the decorator only works with synchronous functions.

1.3 Set annotation or metadata

You can add annotations and metadata to an active segment/subsegment.

Annotations are simple key-value pairs that are indexed for use with filter expressions. Use annotations to record data that you want to use to group traces in the console, or when calling the GetTraceSummaries API. Annotation keys should only use ASCII letters, numbers, and the underscore(_) character.

Metadata are key-value pairs with values of any type, including objects and lists, but that are not indexed. Use metadata to record data you want to store in the trace but don't need to use for searching traces.

You can add annotations/metadata like the following:

```
from aws_xray_sdk.core import xray_recorder

segment = xray_recorder.current_segment()
# value can be string, number or bool
segment.put_annotation('key', value)
# namespace and key must be string and value is an object
# that can be serialized to json
segment.put_metadata('key', json, 'namespace')
```

The `current_segment` and `current_subsegment` functions get the current open segment or subsegment, respectively, from context storage. Put these calls between segment or subsegment begin and end statements.

1.4 AWS Lambda Integration

To integrate with Lambda you must first enable active tracing on a Lambda function. See http://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html#using-x-ray for details.

In your Lambda function, you can only begin and end a subsegment. The Lambda service emits a segment as the root. This segment cannot be mutated. Instrument the SDK as you would in any Python script. Subsegments generated outside of the Lambda handler are discarded.

CHAPTER

TWO

CONFIGURE GLOBAL RECORDER

2.1 Sampling

Sampling is enabled by default. Whenever the global recorder creates a segment, it decides whether to sample this segment. If it does not sample this segment, it is discarded and not sent to the X-Ray daemon.

To turn off sampling, use code like the following:

```
from aws_xray_sdk.core import xray_recorder
xray_recorder.configure(sampling=False)
```

You can also configure the sampling rules:

```
xray_recorder.configure(sampling_rules=your_rules)
```

The input can either be an absolute path to your sampling rule *.json* file or a dictionary.

The following code is an example of a rule configuration:

```
{
"version": 1,
"rules": [
    {
    "description": "Player moves.",
    "service_name": "*",
    "http_method": "*",
    "url_path": "/api/move/*",
    "fixed_target": 0,
    "rate": 0.05
    }
],
"default": {
    "fixed_target": 1,
    "rate": 0.1
    }
}
```

This example defines one custom rule and a default rule. The custom rule applies a five-percent sampling rate with no minimum number of requests to trace for paths under */api/move/*. The default rule traces the first request each second and 10 percent of additional requests. The SDK applies custom rules in the order in which they are defined. If a request matches multiple custom rules, the SDK applies only the first rule. You can use wildcard character "*" and "?" in service_name, http_method and url_path. "*" represents any combination of characters. "?" represents a single character.

Note that sampling configurations have no effect if the application runs in AWS Lambda.

2.2 Plugins

The plugin adds extra metadata for each segment if the app is running on that environment. The SDK provides three plugins:

- Amazon EC2 – EC2Plugin adds the instance ID and Availability Zone.
- Elastic Beanstalk – ElasticBeanstalkPlugin adds the environment name, version label, and deployment ID.
- Amazon ECS – ECSPlugin adds the container host name

To use plugins, use code like the following:

```
# a tuple of strings
plugins = ('elasticbeanstalk_plugin', 'ec2_plugin', 'ecs_plugin')
# alternatively you can use
plugins = ('ElasticBeanstalkPlugin', 'EC2Plugin', 'ECSPlugin')

xray_recorder.configure(plugins=plugins)
```

Order matters in the tuple and the origin of the segment is set from the last plugin. Therefore, in the previous example, if the program runs on ECS, the segment origin is 'AWS::ECS::CONTAINER'. Plugins must be configured before patching any third party libraries to avoid unexpected behavior. Plugins are employed after they are specified.

2.3 Context Missing Strategy

Defines the recorder behavior when your instrumented code attempts to record data when no segment is open. Configure like the following:

```
xray_recorder.configure(context_missing='Your Strategy Name Here')
```

Supported strategies are:

- RUNTIME_ERROR: throw an SegmentNotFoundException
- LOG_ERROR: log an error and continue
- IGNORE_ERROR: do nothing

2.4 Segment Dynamic Naming

For a web application you might want to name the segment using host names. You can pass in a pattern with wildcard character "*" and "?". "*" represents any combination of characters. "?" represents a single character. If the host name from incoming request's header matches the pattern, the host name will be used as the segment name, otherwise it uses fallback name defined in `AWS_XRAY_TRACING_NAME`. To configure dynamic naming, use code like the following:

```
xray_recorder.configure(dynamic_naming='*.example.com')
```

2.5 Environment Variables

There are three supported environment variables to configure the global recorder:

- AWS_XRAY_CONTEXT_MISSING: configure context missing strategy
- AWS_XRAY_TRACING_NAME: default segment name
- AWS_XRAY_DAEMON_ADDRESS: where the recorder sends data to over UDP

Environment variables has higher precedence over `xray_recorder.configure()`

2.6 Logging

The SDK uses Python's built-in `logging` module to perform logging. You can configure the SDK logging just like how you configure other python libraries. An example of set the SDK log level is like the following:

```
logging.basicConfig(level='DEBUG')
logging.getLogger('aws_xray_sdk').setLevel(logging.WARNING)
```

2.7 Context Storage

The global recorder uses threadlocal to store active segments/subsegments. You can override the default context class to implement your own context storage:

```
from aws_xray_sdk.core.context import Context

class MyOwnContext(Context):

    def put_segment(self, segment):
    # store the segment created by ``xray_recorder`` to the context.
    pass

    def end_segment(self, end_time=None):
    # end the segment in the current context.
    pass

    def put_subsegment(self, subsegment):
    # store the subsegment created by ``xray_recorder`` to the context.
    pass

    def end_subsegment(self, end_time=None):
    # end the subsegment in the current context.
    pass

    def get_trace_entity(self):
    # get the current active trace entity(segment or subsegment).
    pass

    def set_trace_entity(self, trace_entity):
    # manually inject a trace entity to the context storage.
```

(continues on next page)

(continued from previous page)

```
    pass

    def clear_trace_entities(self):
    # clean up context storage.
    pass

    def handle_context_missing(self):
    # behavior on no trace entity to access or mutate.
    pass
```

The function `current_segment` and `current_subsegment` on recorder level uses `context.get_trace_entity()` and dynamically return the expected type by using internal references inside segment/subsegment objects.

Then you can pass your own context:

```
my_context=MyOwnContext()
xray_recorder.configure(context=my_context)
```

2.8 Emitter

The default emitter uses non-blocking socket to send data to the X-Ray daemon. It doesn't retry on IOError. To override the default emitter:

```
from aws_xray_sdk.core.emitters.udp_emitter import UDPEmitter

class MyOwnEmitter(UDPEmitter):

    def send_entity(self, entity):
    # send the input segment/subsegment to the X-Ray daemon.
    # Return True on success and False on failure.
    pass

    def set_daemon_address(self, address):
    # parse input full address like 127.0.0.1:8000 to ip and port and
    # store them to the local emitter properties.
    pass
```

Then you can pass your own emitter:

```
my_emitter = MyOwnEmitter()
xray_recorder.configure(emitter=my_emitter)
```

CHAPTER

THREE

THIRD PARTY LIBRARY SUPPORT

3.1 Patching Supported Libraries

The SDK supports aioboto3, aiobotocore, boto3, botocore, pynamodb, requests, sqlite3, httplib and mysql-connector.

To patch, use code like the following in the main app:

```
from aws_xray_sdk.core import patch_all

patch_all()
```

`patch_all` ignores any libraries that are not installed.

To patch specific modules:

```
from aws_xray_sdk.core import patch

i_want_to_patch = ('botocore') # a tuple that contains the libs you want to patch
patch(i_want_to_patch)
```

The following modules are availble to patch:

```
SUPPORTED_MODULES = (
    'aioboto3',
    'aiobotocore',
    'boto3',
    'botocore',
    'pynamodb',
    'requests',
    'sqlite3',
    'mysql',
    'httplib',
)
```

Patching boto3 and botocore are equivalent since boto3 depends on botocore.

Patching pynamodb applies the botocore patch as well, as it uses the logic from the botocore patch to apply the trace header.

3.2 Patching mysql

For mysql, only the mysql-connector module is supported and you have to use code like the following to generate a subsegment for an SQL query:

```
def call_mysql():
    conn = mysql.connector.connect(
        host='host',
        port='some_port',
        user='some_user',
        password='your_password',
        database='your_db_name'
    )

    conn.cursor().execute('SHOW TABLES')
```

3.3 Patching aioboto3 and aiobotocore

On top of patching aioboto3 or aiobotocore, the xray_recorder also needs to be configured to use the `AsyncContext`. The following snippet shows how to set up the X-Ray SDK with an Async Context, bear in mind this requires Python 3.5+:

```
from aws_xray_sdk.core.async_context import AsyncContext
from aws_xray_sdk.core import xray_recorder
# Configure X-Ray to use AsyncContext
xray_recorder.configure(service='service_name', context=AsyncContext())
```

See *Configure Global Recorder* for more information about configuring the `xray_recorder`.

3.4 Patching httplib

httplib is a low-level python module which is used by several third party modules, so by enabling patching to this module you can gain patching of many modules "for free." Some examples of modules that depend on httplib: requests and httplib2

CHAPTER

FOUR

DJANGO

4.1 Configure X-Ray Recorder

Make sure you add `XRayMiddleWare` as the first entry in your Django *settings.py* file, as shown in the following example:

```
MIDDLEWARE = [
    'aws_xray_sdk.ext.django.middleware.XRayMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
]
```

The incoming requests to the Django app are then automatically recorded as a segment.

To get the current segment and add annotations or metadata as needed, use the following statement in your application code when processing request:

```
segment = xray_recorder.current_segment()
```

For more configurations in your Django `settings.py` file, add the following line:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    ...
    'django.contrib.sessions',
    'aws_xray_sdk.ext.django',
]
```

You can configure the X-Ray recorder in a Django app under the 'XRAY_RECORDER' namespace. The default values are as follows:

```
XRAY_RECORDER = {
    'AWS_XRAY_DAEMON_ADDRESS': '127.0.0.1:2000',
    'AUTO_INSTRUMENT': True,  # If turned on built-in database queries and template␣
↪rendering will be recorded as subsegments
    'AWS_XRAY_CONTEXT_MISSING': 'LOG_ERROR',
    'PLUGINS': (),
    'SAMPLING': True,
    'SAMPLING_RULES': None,
    'AWS_XRAY_TRACING_NAME': None, # the segment name for segments generated from␣
↪incoming requests
    'DYNAMIC_NAMING': None, # defines a pattern that host names should match
```

(continues on next page)

(continued from previous page)

```
    'STREAMING_THRESHOLD': None, # defines when a segment starts to stream out its␣
↪children subsegments
}
```

Environment variables have higher precedence over user settings. If neither is set, the defaults values shown previously are used. 'AWS_XRAY_TRACING_NAME' is required unless specified as an environment variable. All other keys are optional. For further information on individual settings, see the *Configure Global Recorder* section.

4.2 Local Development

When doing Django app local development, if you configured Django built-in database with `AUTO_INSTRUMENT` turned-on, the command `manage.py runserver` may fail if `AWS_XRAY_CONTEXT_MISSING` is set to `RUNTIME_ERROR`. This is because the command `runserver` performs migrations check which will generate a subsegment, the `xray_recorder` will raise an error since there is no active segment.

One solution is to set `AWS_XRAY_CONTEXT_MISSING` to `LOG_ERROR` so it only emits a error message on server startup. Alternatively if you have defined your own `ready()` function for code execution at startup you can manually create a segment as a placeholder.

By Django official guide it's recommanded to deploy Django to other servers in production so this particular issue normally doesn't exist in production.

CHAPTER

FIVE

FLASK

To generate segment based on incoming requests, you need to instantiate the X-Ray middleware for flask:

```
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.ext.flask.middleware import XRayMiddleware

app = Flask(__name__)

xray_recorder.configure(service='my_app_name')
XRayMiddleware(app, xray_recorder)
```

Flask built-in template rendering will be wrapped into subsegments. You can configure the recorder, see *Configure Global Recorder* for more details.

CHAPTER

SIX

AIOHTTP

6.1 Server

For X-Ray to create a segment based on an incoming request, you need register some middleware with aiohttp. As aiohttp is an asyncronous framework, X-Ray will also need to be configured with an `AsyncContext` compared to the default threaded version.:

```
import asyncio

from aiohttp import web

from aws_xray_sdk.ext.aiohttp.middleware import middleware
from aws_xray_sdk.core.async_context import AsyncContext
from aws_xray_sdk.core import xray_recorder
# Configure X-Ray to use AsyncContext
xray_recorder.configure(service='service_name', context=AsyncContext())


async def handler(request):
    return web.Response(body='Hello World')

loop = asyncio.get_event_loop()
# Use X-Ray SDK middleware, its crucial the X-Ray middleware comes first
app = web.Application(middlewares=[middleware])
app.router.add_get("/", handler)

web.run_app(app)
```

There are two things to note from the example above. Firstly a middleware corountine from aws-xray-sdk is provided during the creation of an aiohttp server app. Lastly the `xray_recorder` has also been configured with a name and an `AsyncContext`. See *Configure Global Recorder* for more information about configuring the `xray_recorder`.

6.2 Client

Since 3.0.0 Aiohttp provides a generic object that allows third packages to gather the different events ocurred during an HTTP call, X-Ray can be configured to track these requests as subsegments using the *aws_xray_trace_config* function. This will return a valid *TraceConfig* ready to be installed in any *aiohttp.ClientSession*. The following example shows how it can be used.:

```
from aws_xray_sdk.ext.aiohttp.client import aws_xray_trace_config

trace_config = aws_xray_trace_config()
async with ClientSession(loop=loop, trace_configs=[trace_config]) as session:
    async with session.get(url) as resp
        await resp.read()
```

CHAPTER

SEVEN

CHANGELOG

7.1 Unreleased

7.2 2.12.0

- improvement: Default Context Missing Strategy set to Log Error *PR372 https://github.com/aws/aws-xray-sdk-python/pull/372*
- bugfix: Pin tox version to <=3.27.1 to fix CI tests *PR374 https://github.com/aws/aws-xray-sdk-python/pull/374*
- improvement: Sample app dependency update *PR373 https://github.com/aws/aws-xray-sdk-python/pull/373*
- bugfix: Fix pynamodb tests for Python < 3.6 *PR375 https://github.com/aws/aws-xray-sdk-python/pull/375*
- improvement: Use latest GH Actions versions in CI tests *PR365 https://github.com/aws/aws-xray-sdk-python/pull/365*
- improvement: Simplify setup script *PR363 https://github.com/aws/aws-xray-sdk-python/pull/363*
- bugfix: Fix deprecation warnings related to asyncio *PR364 https://github.com/aws/aws-xray-sdk-python/pull/364*
- improvement: Run tests against Python 3.10 and 3.11 *PR376 https://github.com/aws/aws-xray-sdk-python/pull/376*
- improvement: Sample app dependency update *PR380 https://github.com/aws/aws-xray-sdk-python/pull/380*
- bugfix: Pin sqlalchemy version to 1.x to fix tests *PR381 https://github.com/aws/aws-xray-sdk-python/pull/381*
- bugfix: Fix sample app dependencies incompatibility with XRay SDK *PR382 https://github.com/aws/aws-xray-sdk-python/pull/382*
- bugfix: Start MySQL from GH Actions, upgrade Ubuntu, and remove Python versions for unit tests *PR384 https://github.com/aws/aws-xray-sdk-python/pull/384*

7.3 2.11.0

- bugfix: Fix TypeError by patching register_default_jsonb from psycopg2 *PR350 https://github.com/aws/aws-xray-sdk-python/pull/350*
- improvement: Add annotations *PR348 https://github.com/aws/aws-xray-sdk-python/pull/348*
- bugfix: Use service parameter to match centralized sampling rules *PR 353 https://github.com/aws/aws-xray-sdk-python/pull/353*

- bugfix: Implement PEP3134 to discover underlying problems with python3 *PR355 https://github.com/aws/aws-xray-sdk-python/pull/355*
- improvement: Allow list TopicArn for SNS PublishBatch request *PR358 https://github.com/aws/aws-xray-sdk-python/pull/358*
- bugfix: Version pinning flask-sqlalchemy version to 2.5.1 or less *PR360 https://github.com/aws/aws-xray-sdk-python/pull/360*
- bugfix: Fix UnboundLocalError when aiohttp server raises a CancelledError *PR356 https://github.com/aws/aws-xray-sdk-python/pull/356*
- improvement: Instrument httpx >= 0.20 *PR357 https://github.com/aws/aws-xray-sdk-python/pull/357*
- improvement: [LambdaContext] persist original trace header *PR362 https://github.com/aws/aws-xray-sdk-python/pull/362*
- bugfix: Run tests against Django 4.x *PR361 https://github.com/aws/aws-xray-sdk-python/pull/361*
- improvement: Oversampling Mitigation *PR366 https://github.com/aws/aws-xray-sdk-python/pull/366*

7.4 2.10.0

- bugfix: Only import future for py2. PR343.
- bugfix: Defensively copy context entities to async thread. PR340.
- improvement: Added support for IGNORE_ERROR option when context is missing. PR338.

7.5 2.9.0

- bugfix: Change logging behavior to avoid overflow. PR302.
- improvement: Lazy load samplers to speed up cold start in lambda. PR312.
- improvement: Replace slow json file name resolver. PR 306.

7.6 2.8.0

- improvement: feat(sqla-core): Add support for rendering Database Specific queries. PR291.
- bugfix: Fixing broken instrumentation for sqlalchemy >= 1.4.0. PR289.
- feature: no op trace id generation. PR293.
- bugfix: Handle exception when sending entity to Daemon. PR292.
- bugfix: Fixed serialization issue when cause is a string. PR284.
- improvement: Publish metric on distribution availability. PR279.

7.7 2.7.0

- improvement: Only run integration tests on master. PR277.
- improvement: Add distribution channel smoke test. PR276.
- improvement: Replace jsonpickle with json to serialize entity. PR275.
- bugfix: Always close segment in teardown_request handler. PR272.
- improvement: Close segment in only _handle_exception in case of Internal Server Error. PR271.
- bugfix: Handling condition where Entity.cause is not a dict. PR267.
- improvement: Add ability to ignore some requests from httplib. PR263.
- feature: Add support for SQLAlchemy Core. PR264.
- improvement: Added always() to run clean up workflow. PR259.
- improvement: Allow configuring different Sampler in Django App. PR252.
- bugfix: Restore python2 compatibility of EC2 plugin. PR249.
- bugfix: eb solution stack name. PR251.
- improvement: Integration Test Workflow. PR246.
- improvement: Include unicode type for annotation value. PR235.
- improvement: Run tests against Django 3.1 instead of 1.11. PR240.
- bugfix: Generalize error check for pymysql error type. PR239.
- bugfix: SqlAlchemy: Close segment even if error was raised. PR234.

7.8 2.6.0

- bugfix: asyncio.Task.current_task PendingDeprecation fix. PR217.
- bugfix: Added proper TraceID in dummy segments. PR223.
- improvement: Add testing for current Django versions. PR200.
- improvement: IMDSv2 support for EC2 plugin. PR226.
- improvement: Using instance doc to fetch EC2 metadata. Added 2 additional fields. PR227.
- improvement: Added StaleBot. PR228.

7.9 2.5.0

- bugfix: Downgrade Coverage to 4.5.4. PR197.
- bugfix: Unwrap context provided to psycopg2.extensions.quote_ident. PR198.
- feature: extension support as Bottle plugin. PR204.
- bugfix: streaming_threshold not None check. PR205.
- bugfix: Add support for Django 2.0 to 3.0. PR206.
- bugfix: add puttracesegments to boto whitelist avoid a catch 22. PR210.

- feature: Add patch support for pymysql. PR215.

7.10 2.4.3

- bugfix: Downstream Http Calls should use hostname rather than full URL as subsegment name. PR192.
- improvement: Whitelist SageMakerRuntime InvokeEndpoint operation. PR183.
- bugfix: Fix patching for PynamoDB4 with botocore 1.13. PR181.
- bugfix: Add X-Ray client with default empty credentials. PR180.
- improvement: Faster implementation of Wildcard Matching. PR178.
- bugfix: Make patch compatible with PynamoDB4. PR177.
- bugfix: Fix unit tests for newer versions of psycopg2. PR163.
- improvement: Enable tests with python 3.7. PR157.

7.11 2.4.2

- bugfix: Fix exception processing in Django running in Lambda. PR145.
- bugfix: Poller threads block main thread from exiting bug. PR144.

7.12 2.4.1

- bugfix: Middlewares should create subsegments only when in the Lambda context running under a Lambda environment. PR139.

7.13 2.4.0

- feature: Add ability to enable/disable the SDK. PR119.
- feature: Add Serverless Framework Support PR127.
- feature: Bring aiobotocore support back. PR125.
- bugfix: Fix httplib invalid scheme detection for HTTPS. PR122.
- bugfix: Max_trace_back = 0 returns full exception stack trace bug fix. PR123.
- bugfix: Rename incorrect config module name to the correct global name. PR130.
- bugfix: Correctly remove password component from SQLAlchemy URLs, preventing… PR132.

7.14 2.3.0

- feature: Stream Django ORM SQL queries and add flag to toggle their streaming. PR111.
- feature: Recursively patch any given module functions with capture. PR113.
- feature: Add patch support for pg8000 (Pure Python Driver). PR115.
- improvement: Remove the dependency on Requests. PR112.
- bugfix: Fix psycop2 register type. PR95.

7.15 2.2.0

- feature: Added context managers on segment/subsegment capture. PR97.
- feature: Added AWS SNS topic ARN to the default whitelist file. PR93.
- bugfix: Fixed an issue on *psycopg2* to support all keywords. PR91.
- bugfix: Fixed an issue on *endSegment* when there is context missing. ISSUE98.
- bugfix: Fixed the package description rendered on PyPI. PR101.
- bugfix: Fixed an issue where *patch_all* could patch the same module multiple times. ISSUE99.
- bugfix: Fixed the *datetime* to *epoch* conversion on Windows OS. ISSUE103.
- bugfix: Fixed a wrong segment json key where it should be *sampling_rule_name* rather than *rule_name*.

7.16 2.1.0

- feature: Added support for *psycopg2*. PR83.
- feature: Added support for *pynamodb* >= 3.3.1. PR88.
- improvement: Improved stack trace recording when exception is thrown in decorators. PR70.
- bugfix: Argument *sampling_req* in LocalSampler *should_trace* method now becomes optional. PR89.
- bugfix: Fixed a wrong test setup and leftover poller threads in recorder unit test.

7.17 2.0.1

- bugfix: Fixed a issue where manually *begin_segment* might break when making sampling decisions. PR82.

7.18 2.0.0

- **Breaking**: The default sampler now launches background tasks to poll sampling rules from X-Ray backend. See the new default sampling strategy in more details here: https://docs.aws.amazon.com/xray/latest/devguide/xray-sdk-python-configuration.html#xray-sdk-python-configuration-sampling.
- **Breaking**: The *should_trace* function in the sampler now takes a dictionary for sampling rule matching.
- **Breaking**: The original sampling modules for local defined rules are moved from *models.sampling* to *models.sampling.local*.
- **Breaking**: The default behavior of *patch_all* changed to selectively patches libraries to avoid double patching. You can use *patch_all(double_patch=True)* to force it to patch ALL supported libraries. See more details on ISSUE63
- **Breaking**: The latest *botocore* that has new X-Ray service API *GetSamplingRules* and *GetSamplingTargets* are required.
- **Breaking**: Version 2.x doesn't support pynamodb and aiobotocore as it requires botocore >= 1.11.3 which isn't currently supported by the pynamodb and aiobotocore libraries. Please continue to use version 1.x if you're using pynamodb or aiobotocore until those haven been updated to use botocore > = 1.11.3.
- feature: Environment variable *AWS_XRAY_DAEMON_ADDRESS* now takes an additional notation in *tcp:127.0.0.1:2000 udp:127.0.0.2:2001* to set TCP and UDP destination separately. By default it assumes a X-Ray daemon listening to both UDP and TCP traffic on *127.0.0.1:2000*.
- feature: Added MongoDB python client support. PR65.
- bugfix: Support binding connection in sqlalchemy as well as engine. PR78.
- bugfix: Flask middleware safe request teardown. ISSUE75.

7.19 1.1.2

- bugfix: Fixed an issue on PynamoDB patcher where the capture didn't handle client timeout.

7.20 1.1.1

- bugfix: Handle Aiohttp Exceptions as valid responses PR59.

7.21 1.1

- feature: Added Sqlalchemy parameterized query capture. PR34
- bugfix: Allow standalone sqlalchemy integrations without flask_sqlalchemy. PR53
- bugfix: Give up aiohttp client tracing when there is no open segment and LOG_ERROR is configured. PR58
- bugfix: Handle missing subsegment when rendering a Django template. PR54
- Typo fixes on comments and docs.

7.22 1.0

- Changed development status to *5 - Production/Stable* and removed beta tag.
- feature: Added S3 API parameters to the default whitelist.
- feature: Added new recorder APIs to add annotations/metadata.
- feature: The recorder now adds more runtime and version information to sampled segments.
- feature: Django, Flask and Aiohttp middleware now inject trace header to response headers.
- feature: Added a new API to configure maximum captured stack trace.
- feature: Modularized subsegments streaming logic and now it can be overriden with custom implementation.
- bugfix(**Breaking**): Subsegment *set_user* API is removed since this attribute is not supported by X-Ray back-end.
- bugfix: Fixed an issue where arbitrary fields in trace header being dropped when calling downstream.
- bugfix: Fixed a compatibility issue between botocore and httplib patcher. ISSUE48.
- bugfix: Fixed a typo in sqlalchemy decorators. PR50.
- Updated *README* with more usage examples.

7.23 0.97

- feature: Support aiohttp client tracing for aiohttp 3.x. PR42.
- feature: Use the official middleware pattern for Aiohttp ext. PR29.
- bugfix: Aiohttp middleware serialized URL values incorrectly. PR37
- bugfix: Don't overwrite plugins list on each *.configure* call. PR38
- bugfix: Do not swallow *return_value* when context is missing and *LOG_ERROR* is set. PR44
- bugfix: Loose entity name validation. ISSUE36
- bugfix: Fix PyPI project page being rendered incorrectly. ISSUE30

7.24 0.96

- feature: Add support for SQLAlchemy and Flask-SQLAlcemy. PR14.
- feature: Add support for PynamoDB calls to DynamoDB. PR13.
- feature: Add support for httplib calls. PR19.
- feature: Make streaming threshold configurable through public interface. ISSUE21.
- bugfix: Drop invalid annotation keys and log a warning. PR22.
- bugfix: Respect *with* statement on cursor objects in dbapi2 patcher. PR17.
- bugfix: Don't throw error from built in subsegment capture when *LOG_ERROR* is set. ISSUE4.

7.25 0.95

- **Breaking**: AWS API parameter whitelist json file is moved to path *aws_xray_sdk/ext/resources/aws_para_whitelist.json* in PR6.
- Added aiobotocore/aioboto3 support and async function capture. PR6
- Added logic to removing segment/subsegment name invalid characters. PR9
- Temporarily disabled tests run on Django2.0. PR10
- Code cleanup. PR11

7.26 0.94

- Added aiohttp support. PR3

7.27 0.93

- The X-Ray SDK for Python is now an open source project. You can follow the project and submit issues and pull requests on GitHub: https://github.com/aws/aws-xray-sdk-python

7.28 0.92.2

- bugfix: Fixed an issue that caused the X-Ray recorder to omit the origin when recording segments with a service plugin. This caused the service's type to not appear on the service map in the X-Ray console.

7.29 0.92.1

- bugfix: Fixed an issue that caused all calls to Amazon DynamoDB tables to be grouped under a single node in the service map. With this update, each table gets a separate node.

7.30 0.92

- feature: Add Flask support
- feature: Add dynamic naming on segment name

7.31 0.91.1

- bugfix: The SDK has been released as a universal wheel

CHAPTER

EIGHT

LICENSE

Please see Github page on https://github.com/aws/aws-xray-sdk-python/blob/master/LICENSE.

CHAPTER

NINE

INDICES AND TABLES

- modindex
- search