SENTRY

admin July 23, 2019 Leave a comment Edit

Sentry is an open-source **error tracking tool** that helps you monitor and fix crashes in real time.

Sentry captures data by using an SDK within your application's runtime. These are platform specific and allow Sentry to have a deep understanding of both how your application works.

Design → Code & Review → Testing/QA → CI/CD → Issue discovery → Investigation → Remediation

Plays a crucial role in **post-deployment** process of discovery, investigation, and remediation.

Sentry service level configuration for **spring**. It is configured as part of NextService jar.

MICROSERVICES:

Sentry for microservices is configured as part of **NextServiceArchetype**. It involves creation of

1. **SentryServletInitializer** bean
2. **SentryExceptionResolver** bean

They are both initialized as part of **SentryConfiguration** component in archetype.

To create a new dashboard for a project :

1. From the admin panel, create a new dashboard
2. Get the client dsn for the dashboard
3. Assign this value to “**Sentry.project.url”** in corresponding properties file of the service.

SETUP:

Minimum services to be running before, setting up sentry :

1. PostgreSQL
2. Redis

Sentry also supports horizontal scaling, i,e running on a cluster of machines.

Things to consider before setting up of sentry :

1. TTL for events (Storage of historical data)
2. Average event throughput
3. How many events get grouped together

Always restart Sentry services **web, worker, cron** after **upgrading**.

Two essential config files required for sentry configuration :

1. Config.yml [Configure core sentry attributes]
2. Sentry.conf.py [Python file is loaded after all configuration files are loaded]

Configurable parameters as part of **config.yml**

GENERAL

1. system.admin-email [reported to sentry team as part of beacon]
2. system.url-prefix [Reference in UI as well as outbound notifications]

3. system.secret-key [Used for session signing]

REDIS

1. redis.clusters [referenced by cache, digests, and TSDB backends]

MAIL

Configurable parameters as part of **sentry.python.conf**

WEB SERVER

1. SENTRY_WEB_HOST
2. SENTRY_WEB_PORT,
3. SENTRY_WEB_OPTIONS (dictionary of additional configuration options to pass to uwsgi.)]

SMTP SERVER

BEACON

1. SENTRY_BEACON – [allows communication with remote sentry servers]

Sentry provides an abstraction called ‘filestore’ which is used for storing files. The default backend stores files on the local disk in /tmp/sentry-files which is not suitable for production use. In **config.yml**, following options can be placed for attaching a file storage system.

filestore.backend: ‘S3’

filestore.options:

access_key: '…'

secret_key: '…'

bucket_name: '…'

ASYNCHRONOUS WORKERS :

1. Sentry comes with a built-in queue to process tasks in a more asynchronous fashion.
   For example when an event comes in instead of writing it to the database immediately, it sends a job to the queue so that the request can be returned right away, and the background workers handle actually saving that data.
2. Sentry also needs a cron process.

Above operations can be managed by using **supervisor**.

Sentry currently supports two primary backends : Redis, RabbitMq

Drawback with Redis : all pending work must fit in memory

Eg: BROKER_URL = "redis://:password@localhost:6379/0"

WRITE BUFFERS :

Sentry manages database row contention by buffering writes and flushing bulk changes to the database over a period of time. This is extremely helpful if you have high concurrency, especially if they're frequently the same event.

For example, if you happen to receive 100,000 events/second, and 10% of those are reporting a connection issue to the database (where they'd get grouped together),

enabling a buffer backend will change things so that each count update is actually put into a queue, and all updates are performed at the rate of how fast the queue can keep up.

Sentry provides a service to store **time-series data**. Primarily this is used to display aggregate information for events and projects, as well as calculating (in real-time) the rates of events.

It will use the default **Redis cluster** configured unless specifically configured to use another one.

Historically, SENTRY_CONF or –config was pointed directly to your sentry.conf.py. Now, SENTRY_CONF should be pointed to the parent directory that contains both the python file and the yaml file. sentry init will generate the right structure needed for the future

THROTTLES AND RATE LIMITING:

Too much inbound traffic without a good way to drop excess messages.

Event Quota : throttling workloads in Sentry involves setting up event quotas.

It provides rate limiting with a wide range of systems. **Refer**

Sentry relies heavily on **queues,** monitoring them is essential.

Sentry provides several ways to monitor the **system status**.

Sentry has a **CLI** and provides different options to achieve those which are not possible with UI.

CURRENT SETUP:

GENERAL ISSUES:

1. Refer **FAQ** first

We are running SENTRY as a docker container in an EC2 machine.

Inside the EC2 machine we have PostGreSQL and Redis configured necessary for the functioning of Sentry.

Following are the docker commands to run SENTRY :

docker run -it –rm -e SENTRY_URL_PREFIX=’http://54.152.37.40:8090′ -e SENTRY_SECRET_KEY=’+i78z!+x*t*(e#qrpq#yojt5fr#td0ztv#sd!&uiwg!gcuy706′ -e SENTRY_REDIS_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_PORT=’5432′ -e SENTRY_DB_NAME=’nextsentry’ -e SENTRY_DB_USER=’sentryapp’ -e SENTRY_DB_PASSWORD=’Gh75Dc’ -e SENTRY_EMAIL_HOST=’mx.learnnext.com’ -e SENTRY_EMAIL_PORT=’25’ -e SENTRY_SERVER_EMAIL=’systeam@nexteducation.in’ sentry upgrade

#————-- MY SENTRY ——————————-

docker run -d –restart always –name my-sentry -e SENTRY_URL_PREFIX=’http://54.152.37.40:8090′ -e SENTRY_SECRET_KEY=’+i78z!+x*t*(e#qrpq#yojt5fr#td0ztv#sd!&uiwg!gcuy706′ -e SENTRY_REDIS_HOST=’10.0.1.46′ -p 8090:9000 -e SENTRY_POSTGRES_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_PORT=’5432′ -e SENTRY_DB_NAME=’nextsentry’ -e SENTRY_DB_USER=’sentryapp’ -e SENTRY_DB_PASSWORD=’Gh75Dc’ -e SENTRY_EMAIL_HOST=’mx.learnnext.com’ -e SENTRY_EMAIL_PORT=’25’ -e SENTRY_SERVER_EMAIL=’systeam@nexteducation.in’ sentry

#————— SENTRY CRON ———————————-

docker run -d –restart always –name sentry-cron -e SENTRY_URL_PREFIX=’http://54.152.37.40:8090′ -e SENTRY_SECRET_KEY=’+i78z!+x*t*(e#qrpq#yojt5fr#td0ztv#sd!&uiwg!gcuy706′ -e SENTRY_REDIS_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_PORT=’5432′ -e SENTRY_DB_NAME=’nextsentry’ -e SENTRY_DB_USER=’sentryapp’ -e SENTRY_DB_PASSWORD=’Gh75Dc’ -e SENTRY_EMAIL_HOST=’mx.learnnext.com’ -e SENTRY_EMAIL_PORT=’25’ -e SENTRY_SERVER_EMAIL=’systeam@nexteducation.in’ sentry run cron

#————————— SENTRY WORKER ————————————-

docker run -d –restart always –name sentry-worker-1 -e SENTRY_URL_PREFIX=’http://54.152.37.40:8090′ -e SENTRY_SECRET_KEY=’+i78z!+x*t*(e#qrpq#yojt5fr#td0ztv#sd!&uiwg!gcuy706′ -e SENTRY_REDIS_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_HOST=’10.0.1.46′ -e SENTRY_POSTGRES_PORT=’5432′ -e SENTRY_DB_NAME=’nextsentry’ -e SENTRY_DB_USER=’sentryapp’ -e SENTRY_DB_PASSWORD=’Gh75Dc’ -e SENTRY_EMAIL_HOST=’mx.learnnext.com’ -e SENTRY_EMAIL_PORT=’25’ -e SENTRY_SERVER_EMAIL=’systeam@nexteducation.in’ sentry run worker

Things to improve :

1. Add a docker file with environment variables in it.
2. Build a docker image and run it with port mapping 8090:9000
3. Handle database(PostgreSQL) and redis(cache)

Leave a comment

Comment

Post Comment