Tonga

Release 0.0.2

Qotto

Nov 03, 2019

CONTENTS:

CHAPTER

ONE

SUMMARIZE

Tonga is a toolkit for building fault-tolerant, distributed, highly concurrent, event-driven applications. Can be used with 0.9+ Kafka brokers

1.1 Installation

1.1.1 Latest Release

Install with your favorite package manager

With Pip:

```
pip install tonga
```

With Pipenv:

```
pipenv install tonga
```

Releases are also listed at https://github.com/Qotto/tonga/releases

1.1.2 Bleeding-Edge

```
git clone https://github.com/Qotto/tonga
pip install ./tonga
```

1.2 Requirements

For development:

```
python_version += 'v3.5.1'
pipenv_version += '2018.11.26'
```

For testing:

```
docker              version : 19.03.0-beta2, build c601560
docker-compose      version : v1.23.2, build 1110ad01
```

1.3 Quickstart

Basic usage

1.3.1 KafkaProducer

KafkaProducer is a high-level, asynchronous message producer.

```
import uvloop
import asyncio
import uuid
import os

from recipes.setup_color_logger import setup_logger

from tonga.services.producer.kafka_producer import KafkaProducer
from tonga.services.serializer.avro import AvroSerializer
from tonga.services.coordinator.partitioner.key_partitioner import KeyPartitioner

from examples.coffee_bar.waiter.models.events.coffee_ordered import CoffeeOrdered
from examples.coffee_bar.waiter.models.handlers.coffee_ordered_handler import␣
↪CoffeeOrderedHandler


async def send_one(kafka_producer):
    coffee_uuid = uuid.uuid4().hex
    coffee_order = CoffeeOrdered(partition_key=coffee_uuid, uuid=coffee_uuid, coffee_
↪for='toto',
                                 coffee_type='Classic', cup_type='Venti', amount=2.5)
    try:
        await kafka_producer.send_and_await(coffee_order, 'waiter-events')
    finally:
        await kafka_producer.stop_producer()


cur_instance = 0
logger = setup_logger()

loop = uvloop.new_event_loop()
asyncio.set_event_loop(loop)

# Creates AvroSerializer
serializer = AvroSerializer(os.path.join(os.path.dirname(os.path.abspath(__file__)),
                                         'examples/coffee_bar/avro_schemas'))
coffee_ordered_handler = CoffeeOrderedHandler()

# Register CoffeeFinished (event class) / CoffeeFinishedHandler (instanced class␣
↪handle all CoffeeFinished event)
serializer.register_class('tonga.waiter.event.CoffeeOrdered', CoffeeOrdered, coffee_
↪ordered_handler)

# Creates KafkaProducer
producer = KafkaProducer(name=f'waiter-{cur_instance}', bootstrap_servers=
↪'localhost:9092',
                         client_id=f'waiter-{cur_instance}', serializer=serializer,
                         loop=loop, partitioner=KeyPartitioner(), acks='all')
```

(continues on next page)

(continued from previous page)

```
loop.run_until_complete(send_one(producer))
```

1.3.2 KafkaConsumer

```
import uvloop
import asyncio
import uuid
import os
from signal import signal, SIGINT

from recipes.setup_color_logger import setup_logger

from tonga.services.consumer.kafka_consumer import KafkaConsumer
from tonga.services.serializer.avro import AvroSerializer

from examples.coffee_bar.waiter.models.events.coffee_ordered import CoffeeOrdered
from examples.coffee_bar.waiter.models.handlers.coffee_ordered_handler import␣
↪CoffeeOrderedHandler

cur_instance = 0
nb_replica = 2
logger = setup_logger()

loop = uvloop.new_event_loop()
asyncio.set_event_loop(loop)

# Creates AvroSerializer
serializer = AvroSerializer(os.path.join(os.path.dirname(os.path.abspath(__file__)),
                                         'examples/coffee_bar/avro_schemas'))
coffee_ordered_handler = CoffeeOrderedHandler()

# Register CoffeeFinished (event class) / CoffeeFinishedHandler (instanced class␣
↪handle all CoffeeFinished event)
serializer.register_class('tonga.waiter.event.CoffeeOrdered', CoffeeOrdered, coffee_
↪ordered_handler)

consumer = KafkaConsumer(name=f'waiter-{cur_instance}', serializer=serializer,
                         bootstrap_servers='localhost:9092', client_id=f'waiter-{cur_
↪instance}',
                         topics=['waiter-events'], loop=loop,
                         assignors_data={'instance': cur_instance,
                                         'nb_replica': nb_replica,
                                         'assignor_policy': 'only_own'}, isolation_
↪level='read_committed')

# Ensures future of KafkaConsumer
asyncio.ensure_future(consumer.listen_event('earliest'), loop=loop)

# Catch SIGINT
signal(SIGINT, lambda s, f: loop.stop())
try:
    # Runs forever
    loop.run_forever()
except Exception:
```

(continues on next page)

(continued from previous page)

```
    # If an exception was raised loop was stopped
    loop.stop()
```

1.4 Examples

In all examples we use coffee bar project (See Tonga examples/coffee_bar folder)

1.4.1 Serializer

Serialize all avro schema to bytes, for register a new schema serializer needs an BaseRecord & BaseHandler. BaseRecord is a serialize class, BaseHandler is call by consumer for handle records.

Register BaseRecord (event / result / command)

```
# Import serializer
from tonga.services.serializer.avro import AvroSerializer

# Import store builder
from tonga.stores.store_builder.store_builder import StoreBuilder

# Import KafkaProducer
from tonga.services.producer.kafka_producer import KafkaProducer

# Import waiter events
from examples.coffee_bar.waiter.models.events.coffee_finished import CoffeeFinished

# Import waiter handlers
from examples.coffee_bar.waiter.models.handlers.coffee_finished_handler import␣
↪CoffeeFinishedHandler

# Creates AvroSerializer
serializer = AvroSerializer(os.path.join(os.path.dirname(os.path.abspath(__file__)),
                            'examples/coffee_bar/avro_schemas'))

# Creates StoreBuilder  (More information below StoreBuilder section)
store_builder = StoreBuilder('Go to StoreBuilder documentation')

# Creates transactional producer (More information below Transactional producer␣
↪section)
transactional_producer = KafkaProducer('Go to KafkaProducer documentation')

# Creates CoffeeFinishedHandler (Waiter example)
coffee_finished_handler = CoffeeFinishedHandler(store_builder, transactional_producer)

# Register CoffeeFinished (event class) / CoffeeFinishedHandler (instanced class␣
↪handle all CoffeeFinished event)
serializer.register_class('tonga.waiter.event.CoffeeFinished', CoffeeFinished, coffee_
↪finished_handler)
```

Register StoreRecord

This is an example with Tonga StoreRecordHandler

```
# Import serializer
from tonga.services.serializer.avro import AvroSerializer

# Import store builder
from tonga.stores.store_builder.store_builder import StoreBuilder

# Import StoreRecord & StoreRecordHandler
from tonga.models.store_record.store_record import StoreRecord
from tonga.models.store_record.store_record_handler import StoreRecordHandler

# Creates AvroSerializer
serializer = AvroSerializer(os.path.join(os.path.dirname(os.path.abspath(__file__)),
                            'examples/coffee_bar/avro_schemas'))

# Creates StoreBuilder  (More information below StoreBuilder section)
store_builder = StoreBuilder('Go to StoreBuilder documentation')

# Creates StoreRecordHandler (This handler consumer store record)
store_record_handler = StoreRecordHandler(store_builder)

# Register StoreRecord (Record class) and StoreRecordHandler (instanced class which␣
↪handle StoreRecord)
serializer.register_event_handler_store_record(StoreRecord, store_record_handler)
```

1.4.2 Consumer / Producer

- Producer send BaseRecord (event / command / result / store) in a Kafka topic.
- Consumer receive BaseRecord (event / command / result / store) from a Kafka topic.

```
import uvloop
from signal import signal, SIGINT

# Import KafkaProducer / KafkaConsumer
from tonga.services.consumer.kafka_consumer import KafkaConsumer
from tonga.services.producer.kafka_producer import KafkaProducer

# Import key partitioner
from tonga.services.coordinator.partitioner.key_partitioner import KeyPartitioner

cur_instance = 0

loop = uvloop.new_event_loop()
asyncio.set_event_loop(loop)

# Creates KafkaProducer
producer = KafkaProducer(name=f'waiter-{cur_instance}', bootstrap_servers=
↪'localhost:9092',
                                         client_id=f'waiter-{cur_instance}',␣
↪serializer=serializer,
                                         loop=waiter_app['loop'],␣
↪partitioner=KeyPartitioner(),
                                         acks='all')
```

(continues on next page)

(continued from previous page)

```
# Creates KafkaConsumer
consumer = KafkaConsumer(name=f'waiter-{cur_instance}', serializer=serializer,
                         bootstrap_servers='localhost:9092', client_id=f'waiter-{cur_
↪instance}',
                         topics=['bartender-events'], loop=loop, group_id='waiter',
                         assignors_data={'instance': cur_instance,
                                         'nb_replica': nb_replica,
                                         'assignor_policy': 'only_own'}, isolation_
↪level='read_committed')

# Ensures future of KafkaConsumer
asyncio.ensure_future(consumer.listen_event('committed'), loop=loop)

# Catch SIGINT
signal(SIGINT, lambda s, f: loop.stop())
try:
    # Runs forever
    loop.run_forever()
except Exception:
    # If an exception was raised loop was stopped
    loop.stop()
```

1.4.3 Transactional Producer

Warning: Transactional producer can't send message on Kafka if is not in a transaction.

```
import uvloop
from signal import signal, SIGINT

# Import KafkaProducer / KafkaConsumer
from tonga.services.consumer.kafka_consumer import KafkaConsumer
from tonga.services.producer.kafka_producer import KafkaProducer

# Import key partitioner
from tonga.services.coordinator.partitioner.key_partitioner import KeyPartitioner

cur_instance = 0

# Creates event loop
loop = uvloop.new_event_loop()
asyncio.set_event_loop(loop)

# Creates transactional KafkaProducer
producer = KafkaProducer(name=f'waiter-{cur_instance}', bootstrap_servers=
↪'localhost:9092',
                                        client_id=f'waiter-{cur_instance}',␣
↪serializer=serializer,
                                        loop=waiter_app['loop'],␣
↪partitioner=KeyPartitioner(),
                                        acks='all', transactional_id=f'waiter')
```

Make transaction

Transaction example from waiter project (tonga/example/coffee-bar/waiter)

```
from aiokafka import TopicPartition

# Import BaseCommandHandler
from tonga.models.handlers.command.command_handler import BaseCommandHandler
# Import BaseCommand
from tonga.models.records.command.command import BaseCommand
# Import BaseProducer
from tonga.services.producer.base import BaseProducer

from typing import Union
# Import MakeCoffeeResult / CoffeeStarted event
from examples.coffee_bar.coffeemaker.models.results.make_coffee_result import␣
↪MakeCoffeeResult
from examples.coffee_bar.coffeemaker.models.events.coffee_started import CoffeeStarted


class MakeCoffeeHandler(BaseCommandHandler):
    _transactional_producer: BaseProducer

    def __init__(self, transactional_producer: BaseProducer, **kwargs) -> None:
        super().__init__(**kwargs)
        self._transactional_producer = transactional_producer

    async def execute(self, command: BaseCommand, tp: TopicPartition, group_id: str,␣
↪offset: int) -> Union[str, None]:

       if not self._transactional_producer.is_running():
           await self._transactional_producer.start_producer()

        async with self._transactional_producer.init_transaction():
            # Creates commit_offsets dict

           commit_offsets = {tp: offset + 1}
             # Creates CoffeeStarted event and MakeCoffeeResult result
            coffee_started = CoffeeStarted(command.uuid, context=command.context)
            make_coffee_result = MakeCoffeeResult(command.uuid, context=command.
↪context)

             # Sends CoffeeFinished event
            await self._transactional_producer.send_and_await(coffee_started, 'coffee-
↪maker-events')
            await self._transactional_producer.send_and_await(make_coffee_result,
↪'coffee-maker-results')

             # End transaction
            await self._transactional_producer.end_transaction(commit_offsets, group_
↪id)
        return 'transaction'

    @classmethod
    def handler_name(cls) -> str:
        return 'tonga.coffeemaker.command.MakeCoffee'
```

1.4.4 StoreBuilder

Store builder need an AvroSerializer which contains a StoreRecord and StoreRecordHandler

```
import uvloop
import asyncio
from kafka import KafkaAdminClient
from kafka.cluster import ClusterMetadata

# Import local & global store memory
from tonga.stores.local.memory import LocalStoreMemory
from tonga.stores.globall.memory import GlobalStoreMemory
# Import store builder
from tonga.stores.store_builder.store_builder import StoreBuilder

cur_instance = 0
nb_replica = 2

# Creates event loop
loop = uvloop.new_event_loop()
asyncio.set_event_loop(loop)

# Creates local store memory / global store memory
local_store = LocalStoreMemory(name=f'waiter-{cur_instance}-local-memory')
global_store = GlobalStoreMemory(name=f'waiter-{cur_instance}-global-memory')

cluster_admin = KafkaAdminClient(bootstrap_servers='localhost:9092', client_id=f
↪'waiter-{cur_instance}')
cluster_metadata = ClusterMetadata(bootstrap_servers='localhost:9092')

# Creates store builder
store_builder = StoreBuilder(name=f'waiter-{cur_instance}-store-builder', current_
↪instance=cur_instance,
                             nb_replica=nb_replica, topic_store='waiter-stores',␣
↪serializer=serializer,
                             local_store=local_store, global_store=global_store,
                             bootstrap_server='localhost:9092', cluster_
↪metadata=cluster_metadata,
                             cluster_admin=cluster_admin, loop=loop, rebuild=True,␣
↪event_sourcing=False)

# Ensures future of KafkaConsumer store builder
store_builder.return_consumer_task()

# Catch SIGINT
signal(SIGINT, lambda s, f: loop.stop())
try:
   # Runs forever
   loop.run_forever()
except Exception:
   # If an exception was raised loop was stopped
   loop.stop()
```

1.5 Changelog

1.5.1 0.1.0 (Not released yet)

- **Added**
  - **Services**
    - **Producer**
      - All producer now work with the new BasePositioning class
    - **Consumer**
      - All consumer now work with the new BasePositioning class
    - **Coordinator**
      - **Client**

        New concept BaseClient, user for initialize Consumer / Producer / StoreManager

        New class KafkaClient, used for initialize KafkaConsumer / KafkaProducer / KafkaStoreManager, (Primitive Obsession refactor)
      - **Transaction**

        New concept BaseTransactionManager & BaseTransactionContext

        New class KafkaTransactionManager & KafkaTransactionContext
      - **Async Coordinator**

        New async coordinator, used by stores for make some asynchronous task
  - **Stores**
    - New concept BaseStoreManager (Manage local & global store)
    - **Persistency**
      - Created BasePersistency
      - Added MemoryPersistency
      - Added ShelvePersistency
      - Added RockDBPersistency
  - **Models**
    - **Record**
      - In BaseRecord two serialization abstract method (to_dict / from_dict) | new method base_dict (return base class in dict)
      - In BaseEvent two serialization abstract method (to_dict / from_dict)
      - In BaseCommand two serialization abstract method (to_dict / from_dict) | new method base_dict (return base class in dict)
      - In BaseResult two serialization abstract method (to_dict / from_dict) | new method base_dict (return base class in dict)
    - **Store**
      - In StoreRecord two serialization method (to_dict / from_dict)

        * **Structure**
            - New concept BasePositioning (manage topic partition offset)
            - New structs KafkaPositioning (replace TopicPartition namedtuple)
            - New concept StoreRecordType used by StoreManager (new StoreRecord operation_type 'set/del') (Primitive Obsession refactor)

    - **General**
        * More documentations
        * More tests

- **Changed**
    - **Models**
        * Renamed events package to records
        * Changed (BaseRecord, BaseEvent, BaseCommand, BaseResult, StoreRecord) serialization logic
        * **Store**
            - Changed 'ctype' variable in BaseStoreRecord to 'operation_type' (type : StoreRecordType)
            - Moved BaseStoreRecord in models.records
            - Moved StoreRecords in models.records.store
            - Moved BaseStoreRecordHandler in models.handler
            - Moved StoreRecordHandler in models.handlers.store
    - **Services**
        * **Producer**
            - BaseProducer inherit form ABCMeta
            - All producer work with the new BasePositioning class
            - Changed KafkaProducer constructor params (removed name, bootstrap_servers)
            - Renamed KafkaProducer 'send_and_await' method to 'send_and_wait'
            - Renamed KafkaProducer 'send_and_await' method return (BasePositioning)
            - Changed KafkaProducer 'send' method return (Awaitable)
        * **Consumer**
            - BaseConsumer inherit form ABCMeta
            - All consumer work with the new BasePositioning class
            - Changed KafkaConsumer constructor params (removed name, bootstrap_servers)
            - Renamed KafkaConsumer 'listen_event' method to 'listen_records'
        * **Serializer**
            - AvroSerializer handler_class is now optional in register_class function

  - **Stores**
    - **Manager**
      - Renamed BaseStoreBuilder to BaseStoreManager
      - BaseStoreManager inherit form ABCMeta
      - Renamed StoreBuilder to KafkaStoreManager
      - StoreManager constructor params (removed current_instance, nb_replica, bootstrap_server, cluster_metadata, cluster_admin)
      - Full refactored LocalStore (Now is abstract class using Persistency layer)
      - Full refactored GlobalStore (Now is abstract class using Persistency layer)
      - Local & Global store sends StoreRecords in event bus, after received ack, stores create new asynchronous task for save records
    - **Local & global**
      - BaseStore inherit form ABCMeta
      - Renamed global store package (globall -> global_store)
      - Renamed local store package (local -> local_store)
  - **General**
    - Update aiokafka version 0.5.1 -> 0.5.2
- **Removed**
  - **Models**
    - Removed store_record folder (moved in records / handlers)
    - Removed StoreRecordBase
  - **Store**
    - Removed old Memory LocalStore & GlobalStore
    - Removed Store Metadata
  - **General**
    - Removed all privates variables from documentation
    - Removed all privates methods from documentation
- **Fixed**
  - Some StoreManager bug
  - Store initialization failure
  - Waiter project example
  - Cash-register project example
  - KafkaConsumer crash (without group_id and try to seek committed)
  - KafkaConsumer & KafkaStoreManager bug (Fail to init store if topic / partition have only one record)
  - Tests (new concept adaptation)
  - Possible circular import

1.5.2 0.0.2 (2019-06-05)

- **Added**
  - Handler (events / commands / results)
  - Local & global store (only memory)
  - StoreBuilder
  - Statefulset assignors
  - Key partitioner
  - Statefulset partitioner
  - Tests
- **Changed:**
  - Docker-compose in example/dev_env (1 broker -> 3 broker)
- **Fixed**
  - Some consumer bug
  - Some producer bug
- **Refactored**
  - Exceptions (more explicit)

1.5.3 0.0.1 (2019-04-30)

- Initial release
- **Added**
  - Kafka producer / consumers
  - Avro serializer
  - Schemas model (events / commands / results)
  - Some example (coffee_bar)
  - Docker-compose in example/dev_env (for testing)

1.6 Contributor Covenant Code of Conduct

1.6.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

1.6.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

1.6.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

1.6.4 Scope

This Code of Conduct applies within all project spaces, and it also applies when an individual is representing the project or its community in public spaces. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

1.6.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at **dev@qotto.net**. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

1.6.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

1.7 Contributing

1.8 Tonga API reference

1.8.1 Models

Subpackages

Records

Subpackages

BaseRecord

BaseEvent

BaseCommand

Errors

BaseResult

Handlers

Subpackages

BaseHandler

EventHandler

CommandHandler

ResultHandler

Store

Subpackages

StoreRecord

StoreRecordHandler

Errors

1.8.2 Services

Serializer

Submodules

BaseSerializer

AvroSerializer

KafkaKeySerializer

Errors

Errors

1.8.3 Stores

Subpackages

BaseStore

Global Store

Submodules

BaseGlobalStore

GlobalStoreMemory

GlobalStoreShelve

GlobalStoreRockDB

Local Store

Submodules

BaseLocalStore

LocalStoreMemory

LocalStoreShelve

LocalStoreRockDB

StoreBuilder

Submodules

BaseStoreBuilder

StoreBuilder

Errors

Errors

1.8.4 Utils

Submodules

Decorator

gen_correlation_id

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search