ROTATIONAL

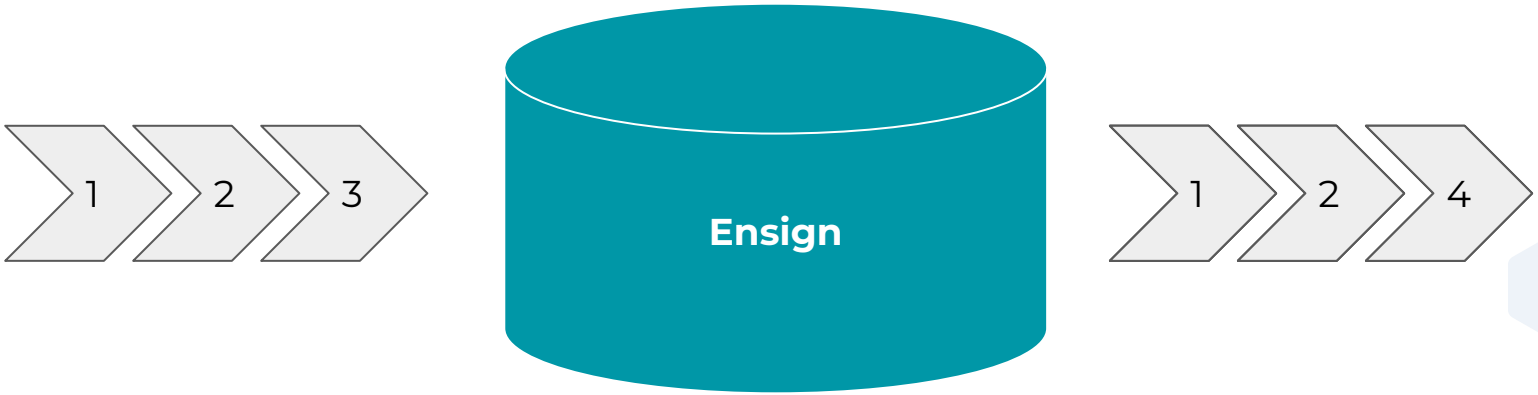
Welcome to Ensign!

Real-time ingestion for data scientists

First, thank you for coming on this journey!

It will require:

- Curiosity
- Courage
- Compassion

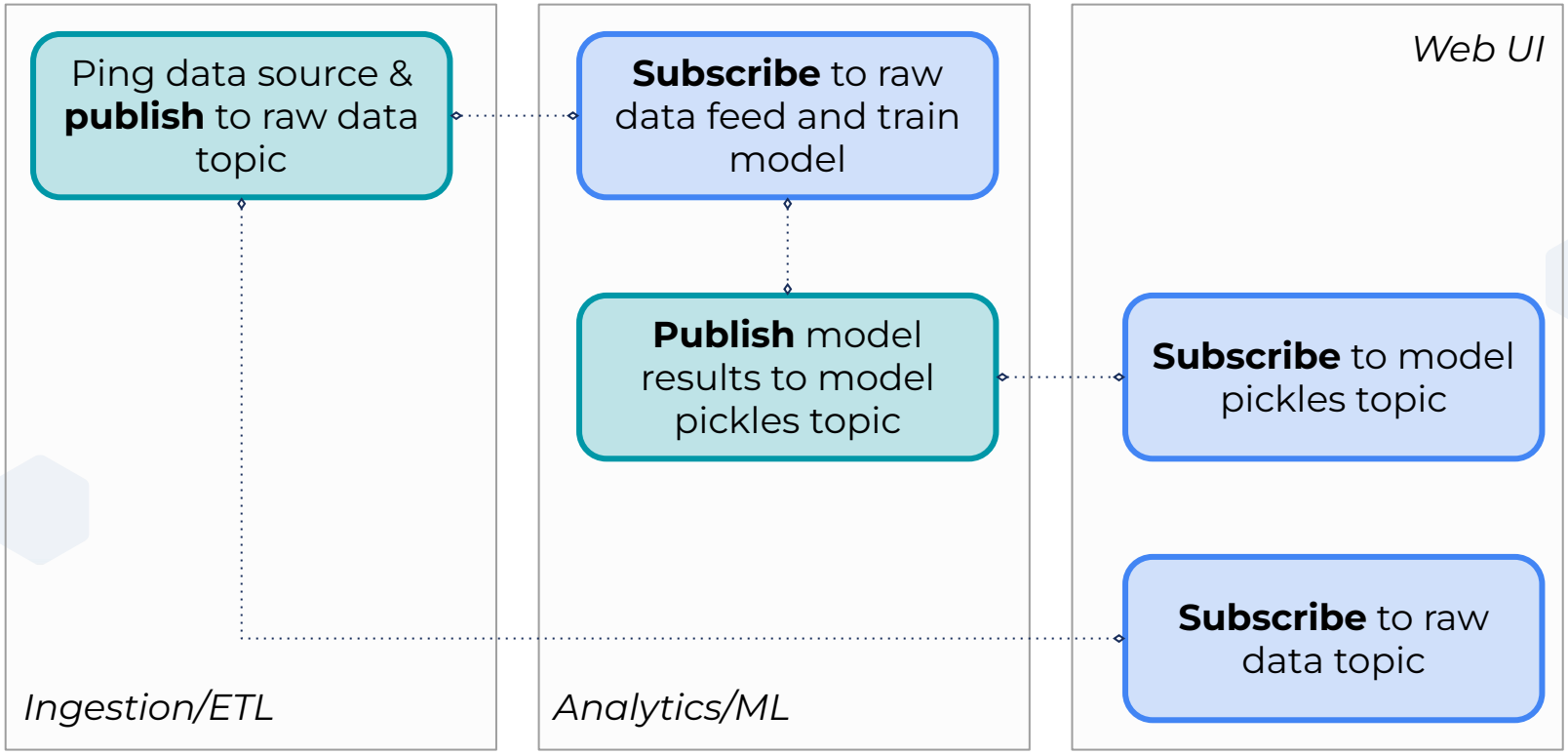


The event log

ROTATIONAL
Navigating data across space and time.

Asynchronous data science flow

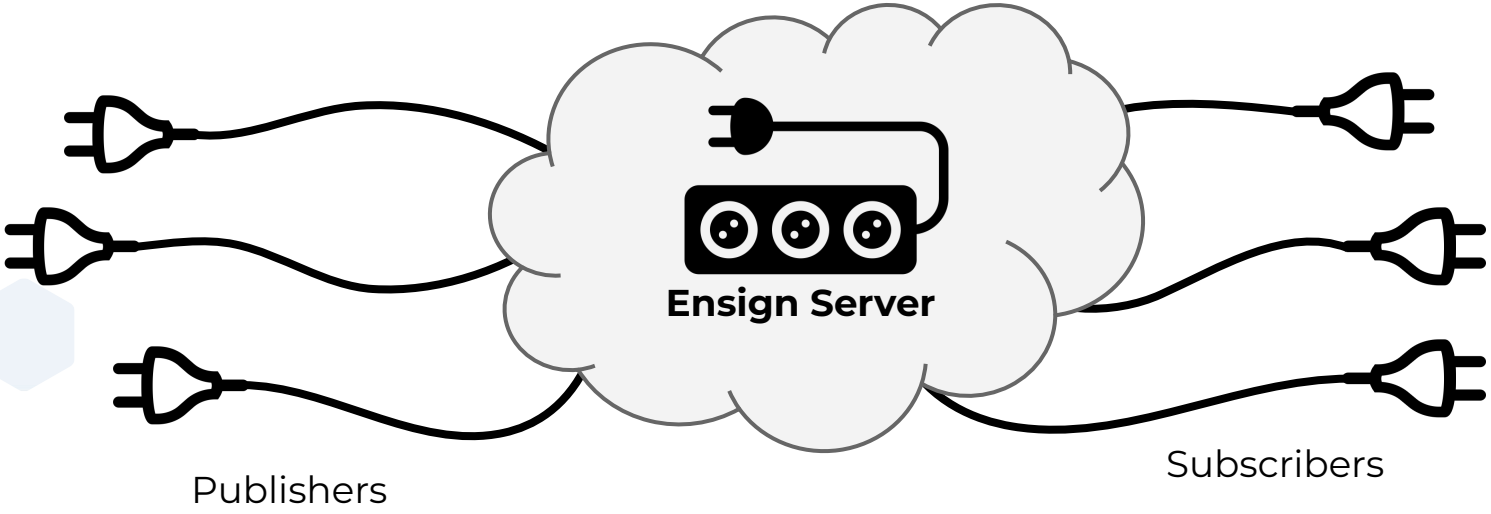
Producers and **consumers** route data between the layers of your application via the topics.



About PyEnsign

PyEnsign allows you to create low-profile data science apps that **publish** and **subscribe** to topic streams, without leaving the comfort of Python.

The PyEnsign API is asynchronous, meaning that **publish** and **subscribe** don't block.



main data-playground / earthquakes / python / Top

README.md

Accessing Real-time Earthquake Data with Python

Connect directly to the USGS API means having realtime access to important and life-saving data about natural disasters.

Getting Started

Ready to save lives? Let's do it!

Install the Dependencies

```
git clone git@github.com:rotationalio/data-playground.git
cd data-playground/earthquakes/python
pip install -r requirements.txt
```

Navigate to the **python example** and **clone it** locally to your machine

Repo link here

```python
class EarthquakePublisher:
    def __init__(self, topic="earthquakes-json", interval=900, user=ME):
        # details omitted for brevity

    def compose_query(self):
        # details omitted

    def unpack_usgs_response(self, message):
        geo_events = message.get("features", None)
        # details omitted

        for geo_event in geo_events:
            details = geo_event.get("properties", None)
            if details is None:
                raise Exception("unable to parse usgs response, no event details found)"
            data = {...} # details omitted
            yield Event(json.dumps(data).encode("utf-8"), mimetype=self.datatype)

    async def recv_and_publish(self):
        await self.ensign.ensure_topic_exists(self.topic)
        while True:
            query = self.compose_query()
            response = requests.get(query).json()
            events = self.unpack_usgs_response(response)
            for event in events:
                await self.ensign.publish(
                    self.topic,
                    event,
                    on_ack=self.print_ack,
                    on_nack=self.print_nack,
                )
            await asyncio.sleep(self.interval)

    def run(self):
        asyncio.run(self.recv_and_publish())
```

```python
class EarthquakeSubscriber:
    def __init__(self, topic="earthquakes-json"):
        self.topic = topic
        self.ensign = Ensign()

    def run(self):
        """
        Run the subscriber forever.
        """
        asyncio.run(self.subscribe())

    async def handle_event(self, event):
        try:
            data = json.loads(event.data)
        except json.JSONDecodeError:
            print("Received invalid JSON in event payload:", event.data)
            await event.nack(nack.UnknownType)
            return

        print("New earthquake report received:", data)
        await event.ack()

    async def subscribe(self):
        """
        Subscribe to the earthquake topic and parse the events.
        """
        id = await self.ensign.topic_id(self.topic)
        async for event in self.ensign.subscribe(id):
            await self.handle_event(event)
```

ROTATIONAL

```python
class EarthquakeSubscriber:
    def __init__(self, topic="earthquakes-json", threshold=4.2):
        self.topic = topic
        self.ensign = Ensign()
        self.threshold = threshold

    async def handle_event(self, event):
        emergency = false
        try:
            data = json.loads(event.data)
            emergency = self.detect_emergencies(data)
        except json.JSONDecodeError:
            print("Received invalid JSON in event payload:", event.data)
            await event.nack(nack.UnknownType)
            return

        if emergency:
            print("Severe disturbance detected, engage emergency measures:," data)
        else:
            print("Low-magnitude earthquake detected:", data)
        await event.ack()

    def detect_emergencies(self, geo_event):
        magnitude = geo_event.get("magnitude", None)
        if magnitude is not None and magnitude >= self.threshold:
            return true
        else:
            return false
```

You can embed real-time analytics into a Subscriber to interpret events without delay.

ROTATIONAL

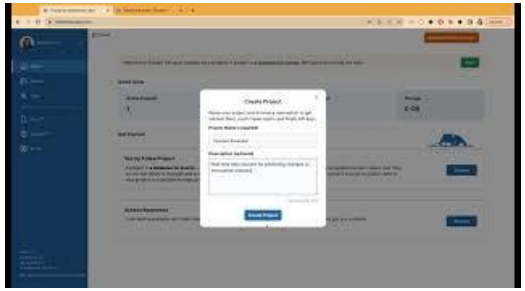
```python
class EarthquakeSubscriber:
    def __init__(self, topic="earthquakes-json", threshold=4.2):
        self.topic = topic
        self.ensign = Ensign()
        self.threshold = threshold

    async def handle_event(self, event):
        emergency = false
        try:
            data = json.loads(event.data)
            emergency = self.detect_emergencies(data)
        except json.JSONDecodeError:
            print("Received invalid JSON in event payload:", event.data)
            await event.nack(nack.UnknownType)
            return

        if emergency:
            print("Severe disturbance detected, engage emergency measures:", data)
        else:
            print("Low-magnitude earthquake detected:", data)
        await event.ack()

    def model_emergencies(self, geo_event):
        magnitude = geo_event.get("magnitude", None)
        if magnitude is not None:
            self.update_model(geo_event)
            print("Predictive model has been updated!")
```

You can use the same trick to do real-time modeling and inference, triggering a model update or prediction immediately.

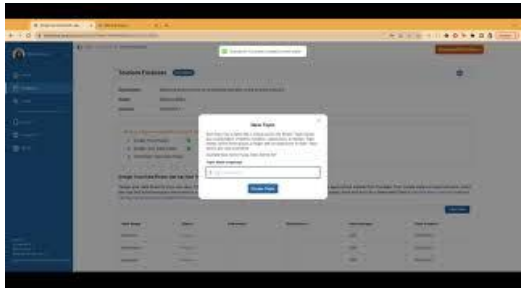
```python
class EarthquakeAnalyzer:
    def __init__(self, topic="earthquakes-json"):
        self.topic = topic
        keys = self._load_keys()
        self.ensign = Ensign(
            client_id=keys["ClientID"],
            client_secret=keys["ClientSecret"]
        )

    async def replay(self, slice=False, sample_size=100):
        """
        Replay all events, from the beginning. Use optional `slice` and
        `sample_size` params to extract a slice of data

        Parameters
        ----------
        slice : boolean, default=False
            Specify True if you want a small slice of data. Else leave false to get
            all historical data.

        sample_size : int, default=100
            If slice is True, the number of events to slice from the beginning of the
            event stream. If slice is False, this parameter is ignored.
        """
        if slice:
            q = f"SELECT * FROM {self.topic} LIMIT {str(sample_size)}"
        else:
            q = f"SELECT * FROM {self.topic}"

        # Get the cursor first!!
        cursor = await self.ensign.query(q)

        # The cursor is the async generator, so asynchronously process the events
        async for event in cursor:
            yield event
```

Replay historic events using EnSQL

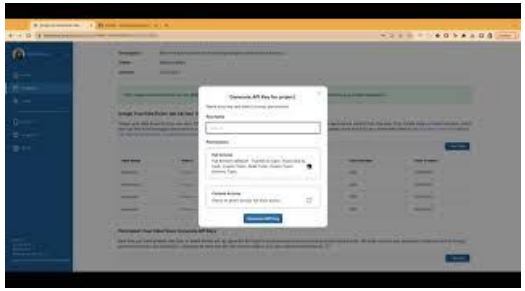
Getting Started Videos: Youtube Playlist



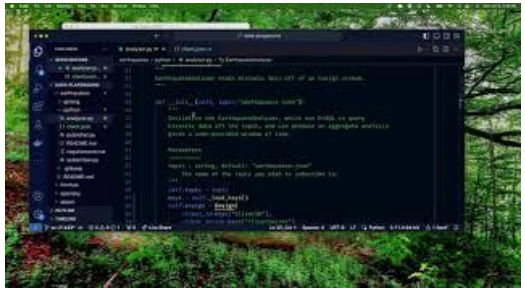
Set up a project



Create a topic



Download API keys

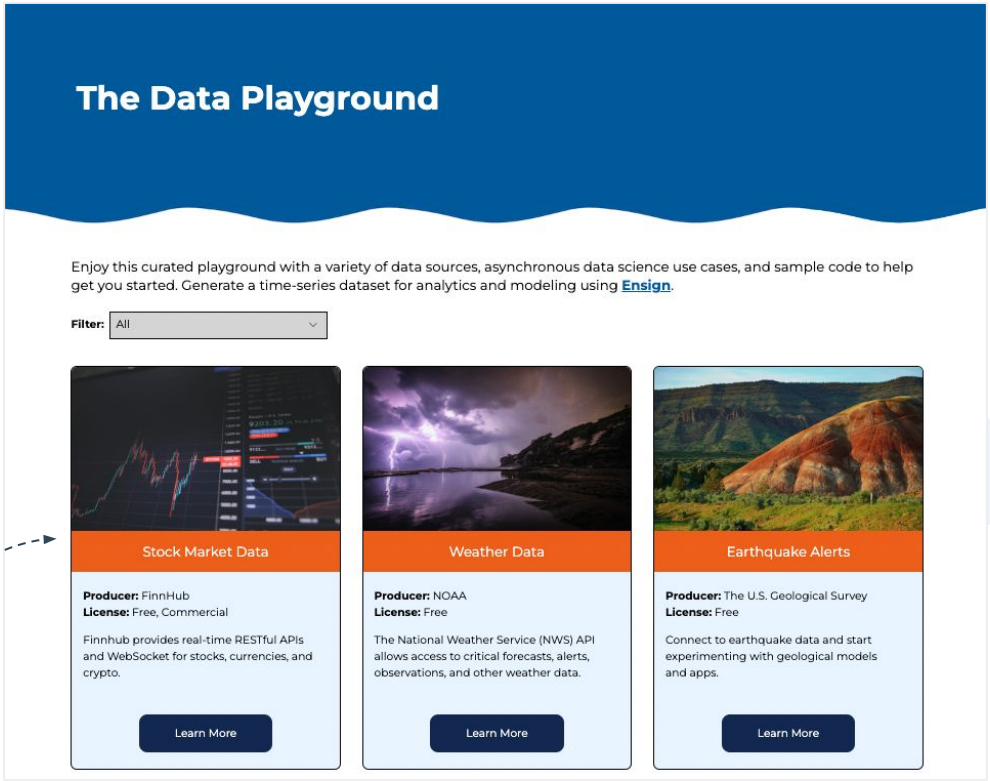


Use API keys in code

A Playground for Streaming Datasets

Check out **more data sources** you can start using now: earthquakes, stock market, weather, flights, etc.

https://rotational.io/data-playground



ROTATIONAL

Good luck & see you at office hours!