Our Digital Twin

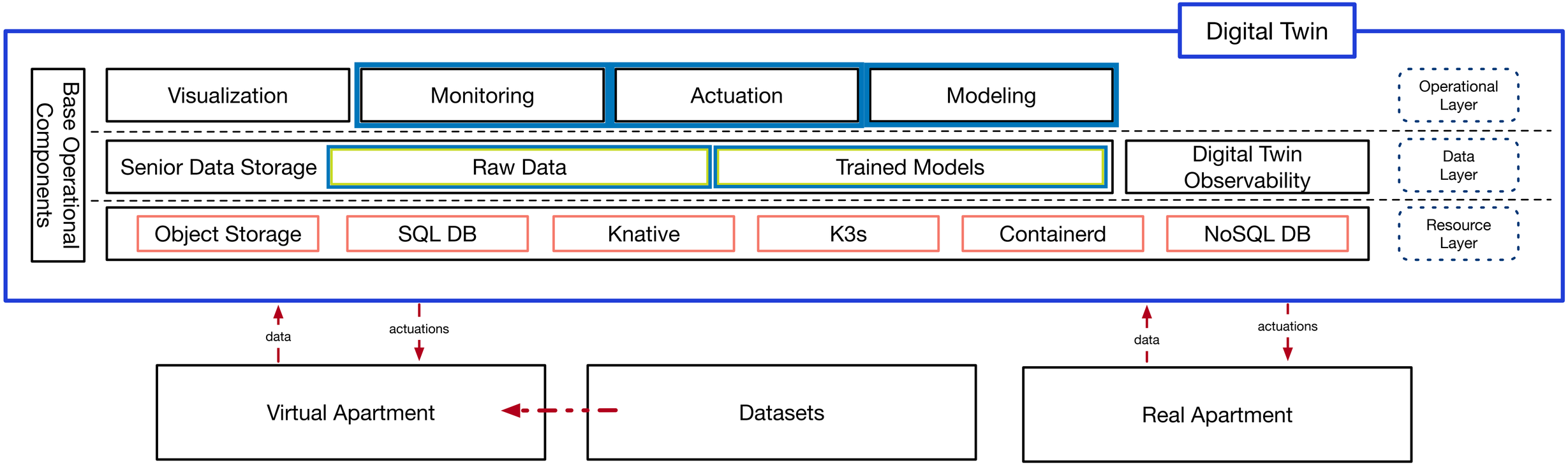
Isaac Nunez
Technische Universität München

Digital Twins

- What is a Digital Twins (DT)?
  - It is a virtual representation of a physical entity
  - What is our physical entity? **You**!
  - What is the virtual representation? **You will have to build it**!
- Where do we start?
  - What do we need to build a virtual representation?
  - What have you been collecting during the last weeks?
  - The difference to industry DTs is that they usually **model** parts via mathematical equations and data
  - Why do we want to model it?
    - Understand reactivity
    - Understand behavior to change
    - Predict change
    - Why are they important?

The progress so far

- What do we have?
- What do we want?



Your Starting Point

- We want some reactivity
  - The purpose? Event handling
- The tool?
  - A base implementation of an **event-driven** scheduler
  - It is a Python implementation that keeps track of functions and events
  - You will use it to communicate between components
  - How do you make your components work with the scheduler?
- The base implementation of your components
  - It is a base Python implementation that registers functions and expected events in the scheduler.
  - It also provides functionality to create **triggers**
    - A trigger creates the **event** and its **necessary data**
    - The trigger sends it to the **scheduler**
    - The scheduler ensures that the function's events have all arrived. Once they all have arrived, the **invocation** is forwarded for execution (**dispatched**), the remote resource is contacted, and the necessary data is passed.

How to fetch data?

- InfluxDB v2 has a Python client called `influxdb-client-python`. You will use it to fetch data from your InfluxDB container
- Keep in mind the following considerations:
  - Your components must **not enter error states** upon reboots
  - Your components must persist the data, **preferably not on disk**
    - Use the **MinIO** instance that is already present in the cluster
    - The credentials are in the ConfigMap
    - You must fetch them from there via files or the K8s Dashboard
    - MinIO also has a Python client

How to Use the Client?

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(
  url=INFLUX_TOKEN,
  org=INFLUX_ORG,
  username=INFLUX_USER,
  password=INFLUX_PASS,
  verify_ssl=False) as client:
    bucket_api = client.buckets_api()
      res = bucket_api.find_bucket_by_name(bucket)
      if res:
          logger.info(f"Bucket {bucket} found...")
      else:
          bucket_api.create_bucket(bucket_name=bucket, org=INFLUX_ORG)
          logger.info(f"Bucket {bucket} has been created...")
```

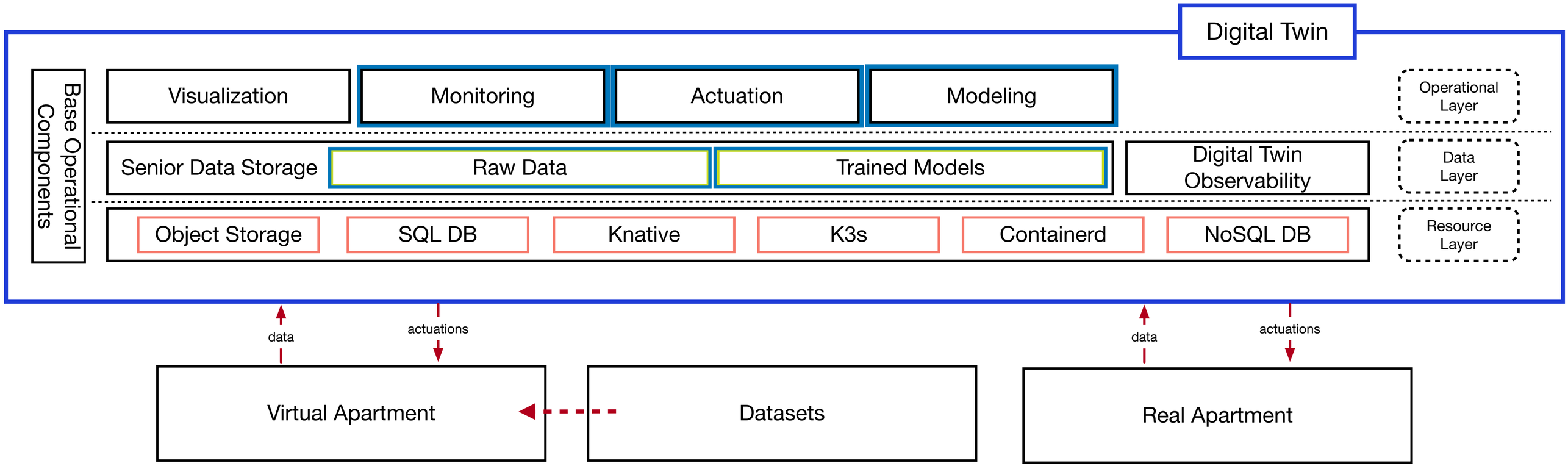
How do you fetch data?

```
from influxdb_client import InfluxDBClient

with InfluxDBClient(
  url=INFLUX_TOKEN,
  org=INFLUX_ORG,
  username=INFLUX_USER,
  password=INFLUX_PASS,
  verify_ssl=False) as client:
        p = {
            "_start": timedelta(days=-7),
        }

        query_api = client.query_api()
        tables = query_api.query(f'''
                                 from(bucket: "{bucket}") |> range(start: _start)
                                 |> filter(fn: (r) => r["_measurement"] == "{measurement}")
                                 |> filter(fn: (r) => r["_type"] == "{tag}")
                                 ''', params=p)
        for table in tables:
          for record in table.records:
            logger.info(record)
```

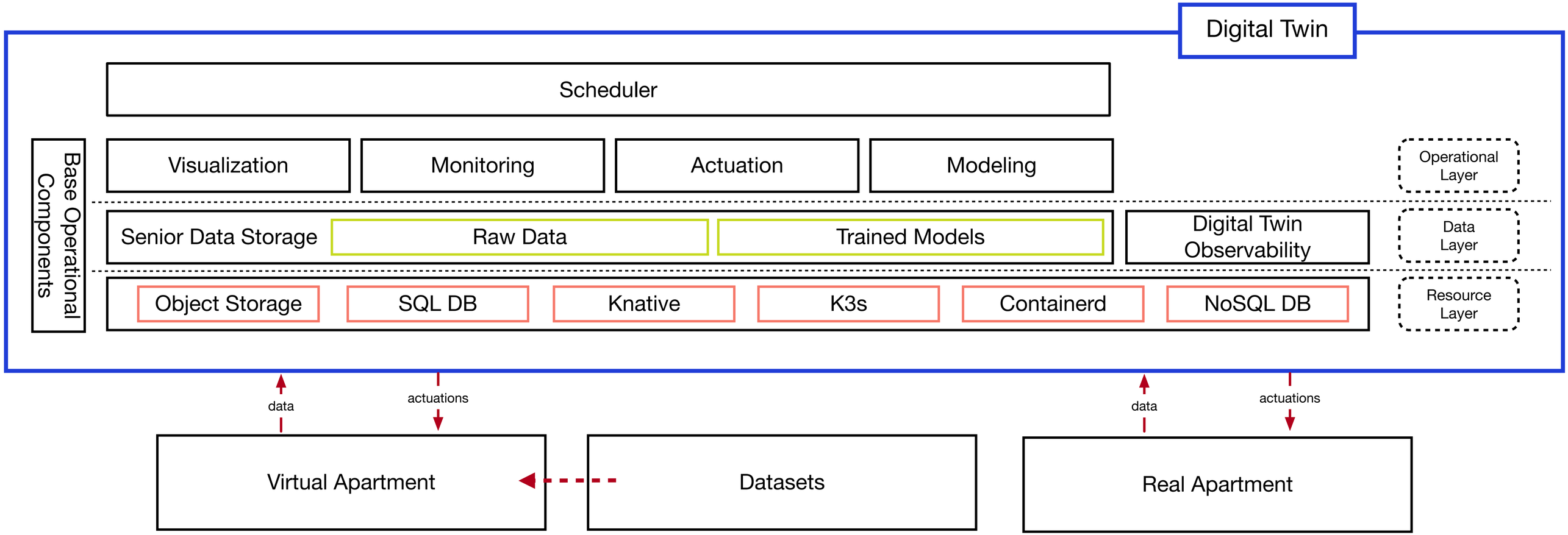
How can you write data?

```
from influxdb_client import InfluxDBClient, Point, WritePrecision

with InfluxDBClient(
  url=INFLUX_TOKEN,
  org=INFLUX_ORG,
  username=INFLUX_USER,
  password=INFLUX_PASS,
  verify_ssl=False) as client:
    write_api = client.write_api(write_options=SYNCHRONOUS)
    data = Point("<measurement>")\
            .tag("<tag>", "<val>")\
            .field("<name>", <val>)\
            .time("<timestamp>",
                  write_precision=WritePrecision.MS)
    write_api.write(bucket=bucket, record=data)

    logger.info("Point has been successfully recorded")
```

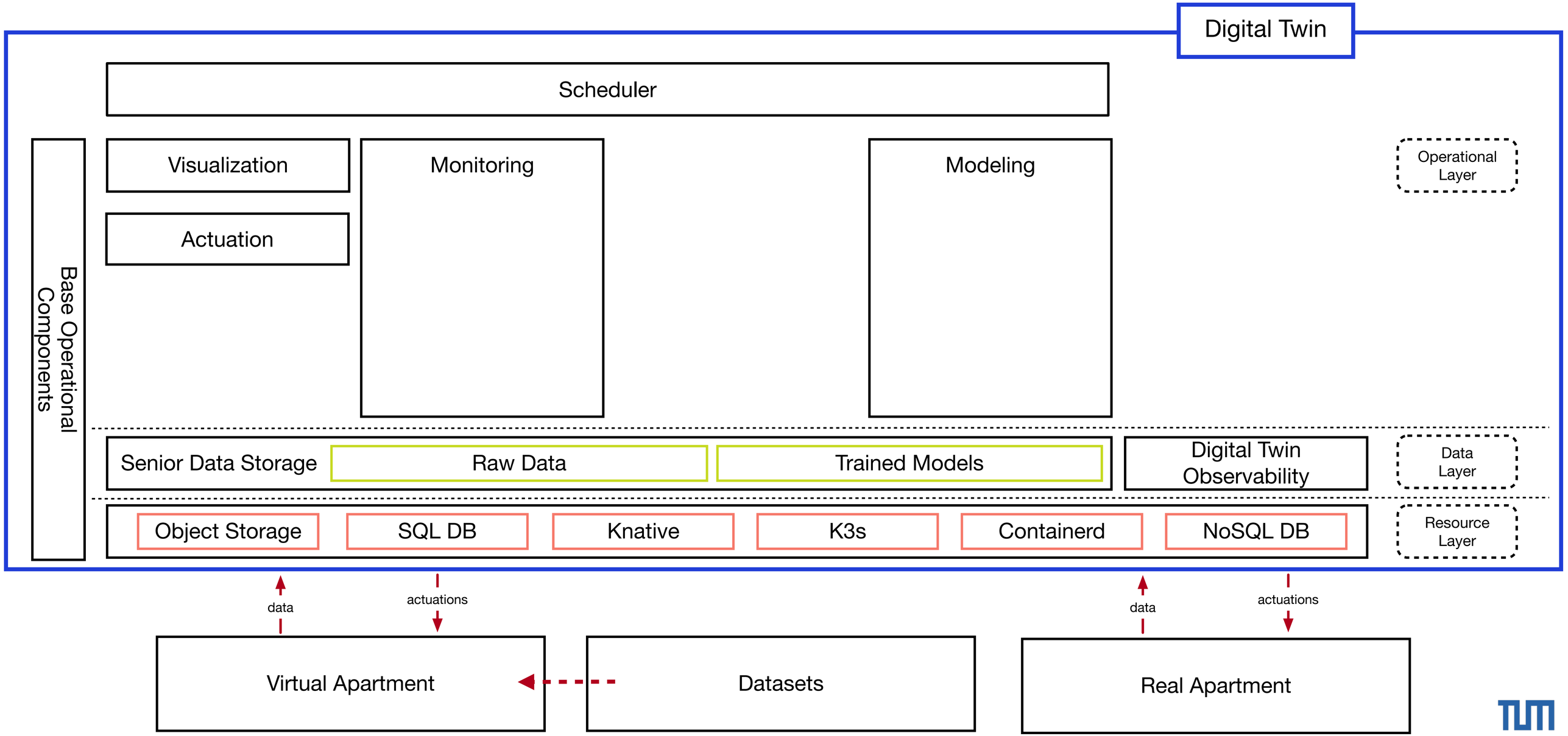
Our playground



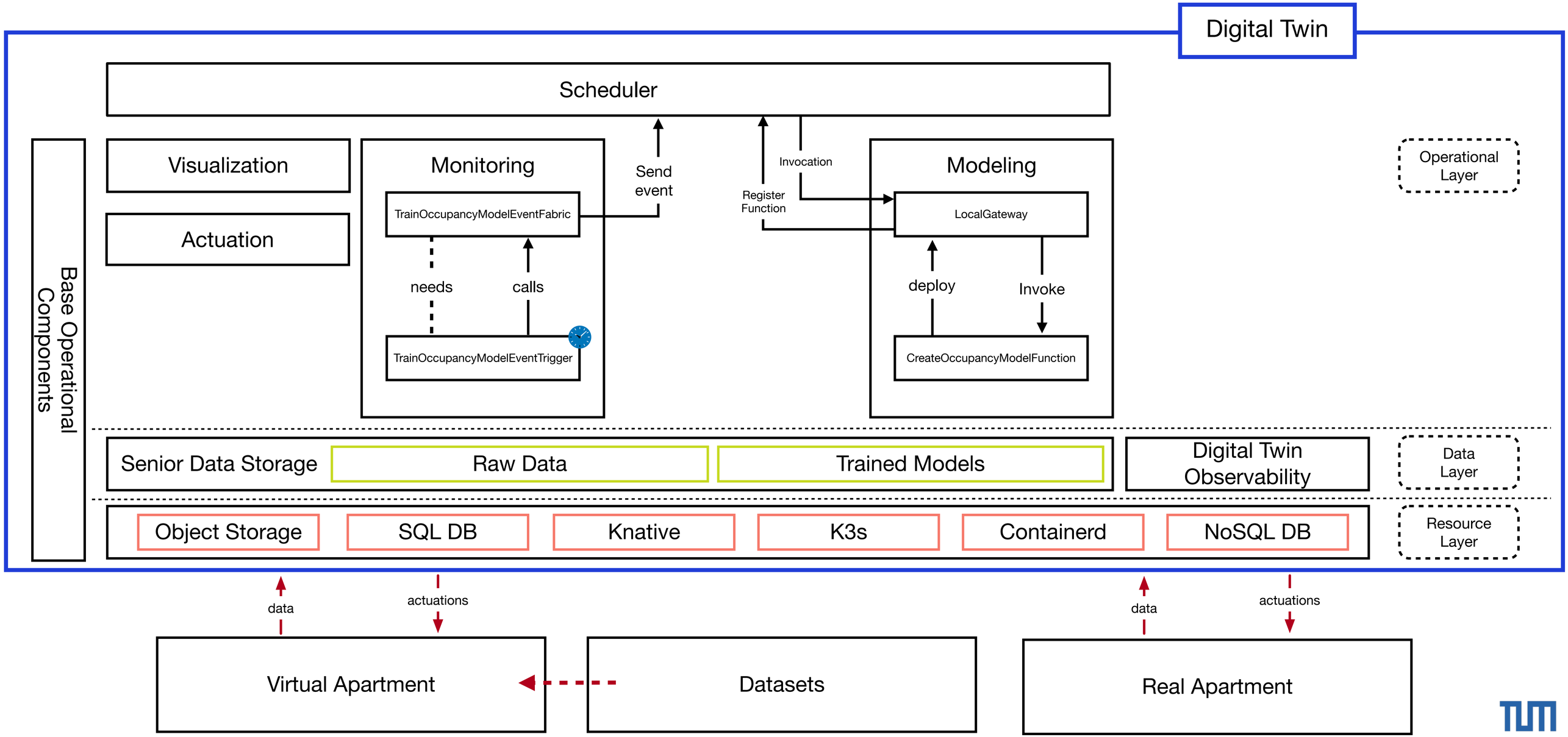
Our playground



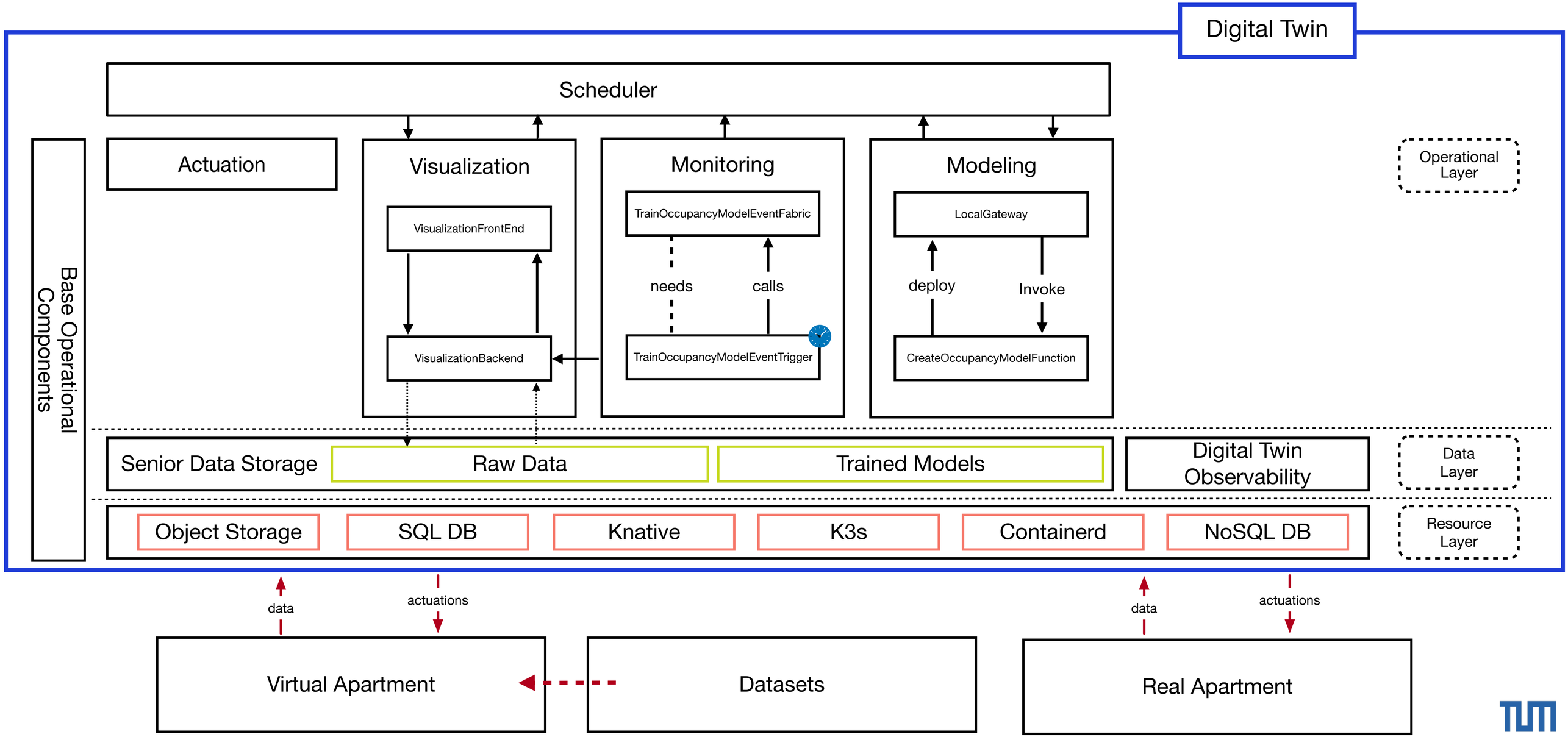
Our playground



Our playground



Our playground



How to create an event?

```
class BaseEventFabric(ABC):
    def __init__(self):
        self.scheduler = os.environ.get("SCH_SERVICE_NAME", "localhost:8080")
        print(f"Relying on the scheduler at {self.scheduler}")
        super(BaseEventFabric, self).__init__()

    @abstractmethod
    def call(self, *args, **kwargs) -> Tuple[str, Any]:
        raise NotImplementedError("Implement the 'call' method in your class")

    def __call__(self, *args, **kwargs):
        evt_name, data = self.call(*args, **kwargs)
        http = urllib3.PoolManager()
        res = http.request('POST', self.scheduler,
                           json=dict(name=evt_name, data=data), retries=urllib3.Retry(5))
        if res.status >= 300:
            print(
                f"Failure to send EventRequest to the scheduler because {res.reason}")
```

Triggers

```
class Trigger(ABC):
    def __init__(self, evt_cb: BaseEventFabric,
                 duration: str = "1s",
                 one_shot: bool = False,
                 wait_time: str = None):

        super(Trigger, self).__init__()

        ...

        self.thr = Thread(target=self.run)
        self.thr.start()

    def run(self):
        while True:
            if self.wt is not None:
                time.sleep(self.wt.total_seconds())

            self.scheduler.run_pending()
            pending_jobs =
self.scheduler.get_jobs()
            if len(pending_jobs) == 0:
                return

            time.sleep(1)
```

Triggers

```
class OneShotTrigger(Trigger):

    def __init__(self, evt_cb: BaseEventFabric, wait_time: str = None):
        super(OneShotTrigger, self).__init__(
            evt_cb, one_shot=True, wait_time=wait_time)


class PeriodicTrigger(Trigger):
    def __init__(self, evt_cb: BaseEventFabric, duration: str, wait_time: str = None):
        super(PeriodicTrigger, self).__init__(
            evt_cb, duration, wait_time=wait_time)
```

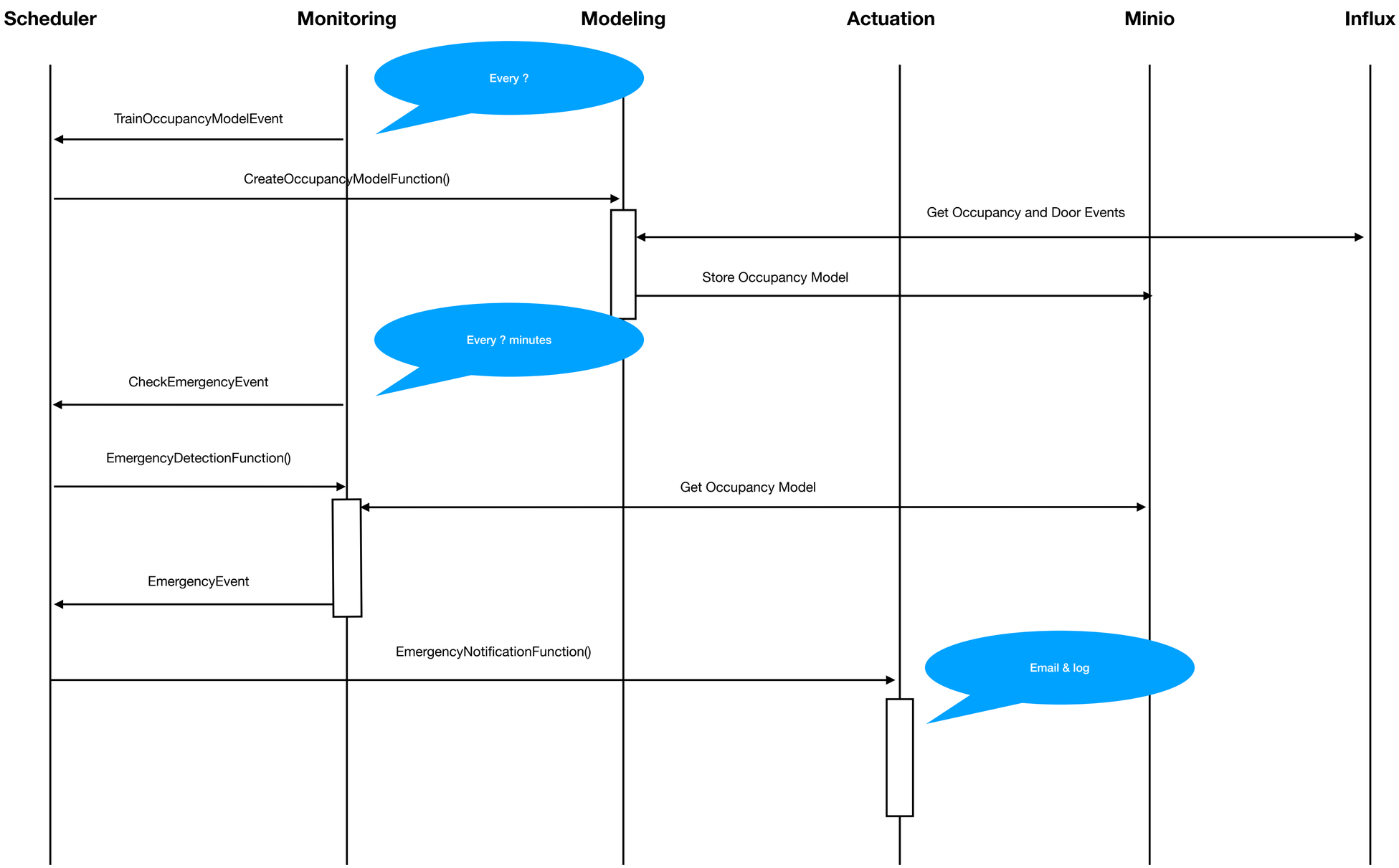
The LocalGateway

```
class LocalGateway(FastAPI):
    ...
    def deploy(self,
               cb: Callable[..., Any],
               name: str,
               evts: List[str] | str,
               method: str = "GET",
               path: str = None):
        ...
        res = http.request('POST', url, json=dict(
                name=name,
                url=endpoint,
                subs=evts,
                method=method.upper()),
                retries=urllib3.Retry(5))
    ...
```

How to create it?

- Requirement is: **Python 3.12**
- Use as a starting point the ***base-sif*** folder. This is the base for all your components.
  - The *base-sif* folder has a „base“ folder with examples on how the component is built and how you can build your triggers.
  - The *requirements.txt* includes all requirements to create your components. The installed package is called ***sifec_base*** and it includes all necessary methods and classes.
- You must also deploy the scheduler ***sif_edge***. You need it to execute your functions.
  - To configure the *LocalGateway*, you must define the **SCH_SERVICE_NAME** as an environment variable with the Service’s URL of the *sif_edge*.
- The respective ***k8s*** folder within *sif_edge* and *base-sif* have all YAML files to deploy it onto your Raspberry Pi.
- Please keep in mind, every component you deploy must be named as stated in these slides: monitoring, modeling, etc.
  - You must replace the component name onto the respective k8s’ YAML file.

What do we want?



Modeling & Decision Details

- Calculate the average duration in each room.
- Build a model that determines the number of visits and average time spent for a visit in a given room.
  - Be aware that the average depends on the day of the week and time of the day (seasonality).
  - **Attention**: The model **MUST** be stored in MinIO with proper metadata, such as creation timestamps and the period of the training data. You **CANNOT** delete old models (**retention policy: forever**).
- You decide what parameters define an emergency.
  - You must justify your decision on the final report.
  - Be aware of taking into account the **variability** and **magnitude of the average** in your data.
- After an emergency is detected, the monitoring **triggers** an alarm for the caregiver via e-mail. The **actuation component** provides the **function** for sending emails.

How to backup your data

rsync -avP --rsync-path="sudo rsync" iot-user@<rpi-ip>:/data/wise2025/<folder>/ <local-folder>/

Possible values for <folder> are influxdb, minio, and sif-edge, mariadb, core