Offline Analysis Tool

Tool Chain description and technologies

Agenda

1. OAT Overview
2. Front-end
3. REST API
4. NoSQL
5. Dispatcher

Offline Analysis Tool

Why ?	Increase product quality
How ?	Ease project's access to testing solutions focused on endurance and big data analysis.
What ?	Implement tools capable of executing checks on big sets of previously recorded data with native reuse support.

Requirements

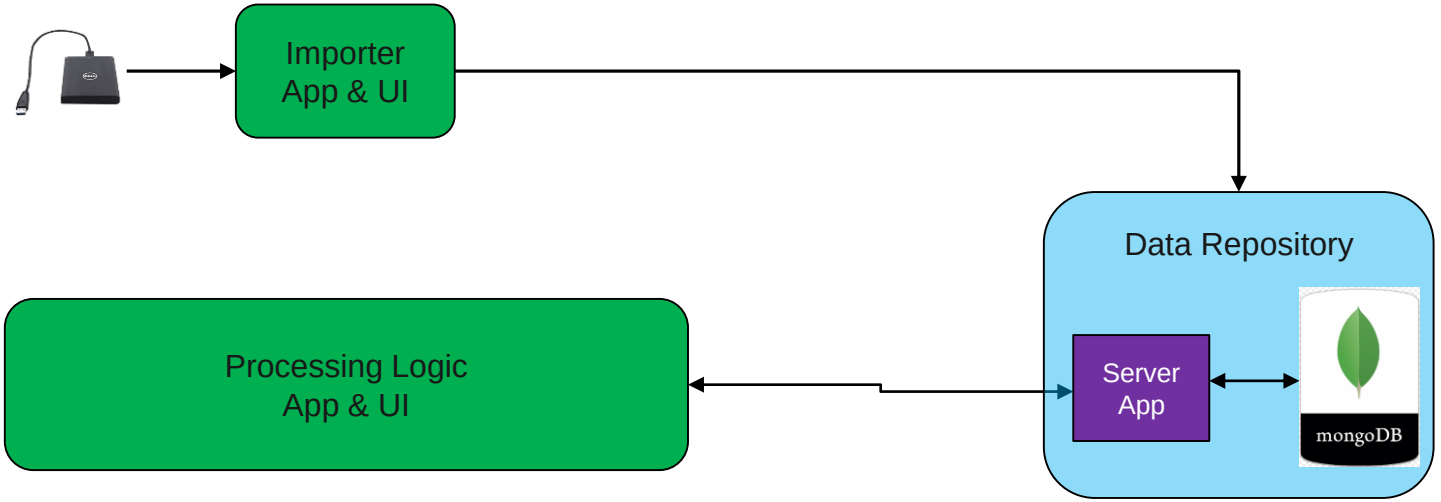
- Recorded data contain bus traffic from the vehicle or test systems.
- Users are able to implement analysis algorithms (plugins) or reuse existing ones.
- Analysis should run offline on a specialized system.
- The user should be able to check the analysis progress and download results when done.

Offline Analysis Tool Case

Typical Use

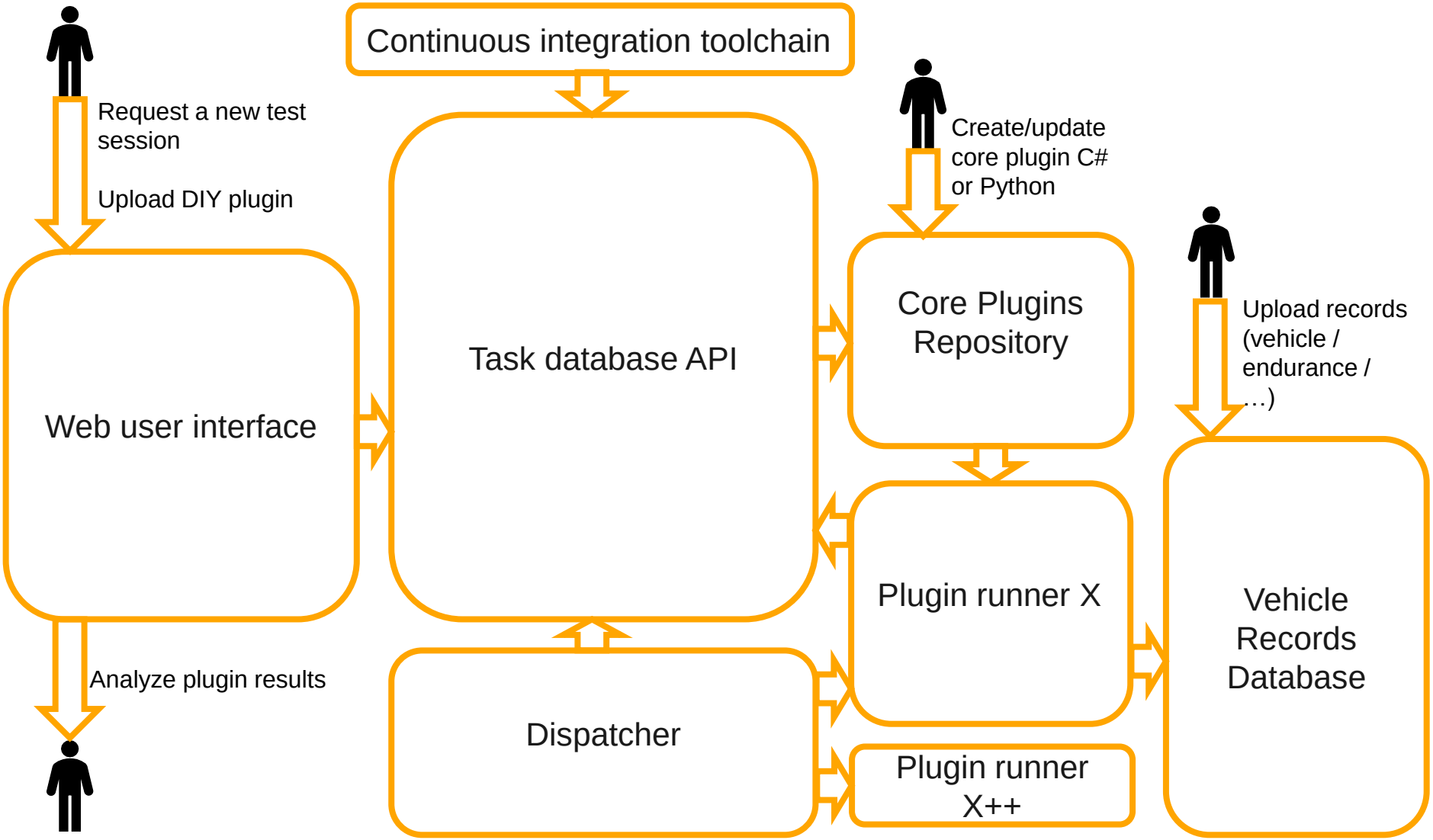
1.

2.



3.

Architecture

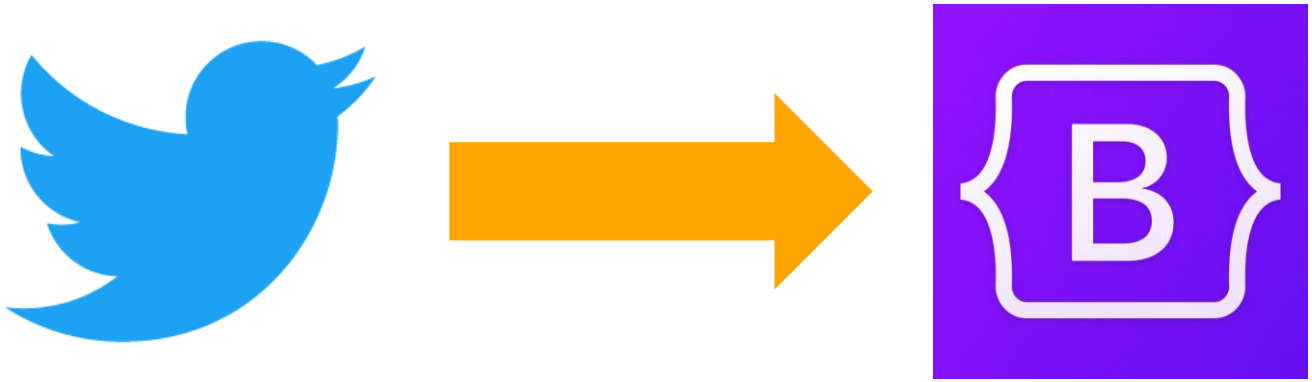


Agenda

1. OAT Overview
2. Front-end
3. REST API
4. NoSQL
5. Dispatcher

Bootstrap

Bootstrap is an open-source front-end framework, used for web application development, made up of HTML, CSS and Java Script, developed by Twitter.



Bootstrap is used to make the website responsive. Responsive means that if you open a website in PC, laptop, tablet or mobile, the screen size of the website will automatically adjust, and the website content shows well.

Advantages of Bootstrap

Time saving

Easy to use

Responsive Design

Cross Browser Compatible

Good documentation and community support

Code

```
<div class="row">

        <div class="my-5">

            <button type="button" class="btn btn-secondary" data-
toggle="modal" data-target="#uploadConfig">Add a new configuration
file</button>

        </div>

</div>
```

```
<div class="modal" id="uploadConfig" role="form">
        <div class="modal-dialog modal-dialog-centered">
            <div class="modal-content">
                <div class="modal-header orange-background darkblue-color">
                    Configuration file information
                    <button type="button" class="close" data-
dismiss="modal">&times;</button>
                </div>
```

```
<div class="modal-body">
    <div class="error-message"></div>
        @using (Html.BeginForm("AddConfigFile", "ConfigFileManager", FormMethod.Post, new { enctype = "multipart/form-data" })){
                <div class="row my-3 mx-2">
                    <div class="custom-file" id="browse">
                        <input type="file" class="custom-file-input" id="customFileConfig" name="file" />
                        <label class="custom-file-label" for="customFile" id="fileLabel">Choose a config file</label>
                    </div>
                </div>
                <div class="row my-3 mx-2">
                    <div class="input-group">
                        <div class="input-group-prepend">
                            <span class="input-group-text styled">Version</span>
                        </div>
                        <input type="text" class="form-control" name="version" required />
                    </div>
                </div>
```

```
                    <div class="row my-3 mx-2">
                        <div class="input-group">
                            <div class="input-group-prepend">
                                <span class="input-group-text styled">Description</span>
                            </div>
                            <input type="text" class="form-control" name="description" required />
                        </div>
                    </div>
                    <div class=“form-btn-container my-3" tabindex="0" data-toggle="tooltip" title="Browse a
file">
                        <button type="submit" class="form-btn" id="uploadConfigBtn" style="cursor:not-
allowed" disabled>Upload</button>
                    </div>
                }
                </div>
            </div>
        </div>
</div>
```

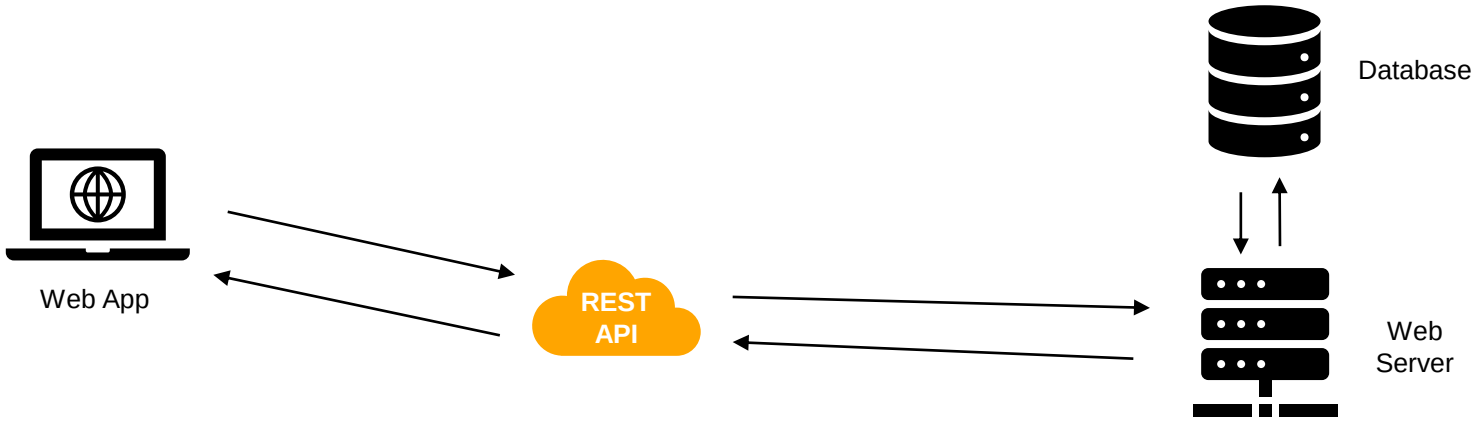
Agenda

1. OAT Overview
2. Front-end
3. REST API
4. NoSQL
5. Dispatcher

REST API

REST stands for **Re**presentational **S**tate **T**ransfer and is simply a set of **architectural standards** or guidelines that structure how you communicate data between your application and the rest of the world, or between different components of your application.

RESTful APIs rely on **HTTP** to transfer information - the same protocol that web communication is based on! So, when you see the **http** at the beginning of a URL, your browser is using HTTP to request that website from a server. REST works in the same way!



REST API

= separation between client and server.

A **client** is a **person** or **software** that uses the API. Meanwhile, a **server** is a remote computer able to grab data from the database and hand it over to the API.

Typically, there's a separation between these two app components. This enables the client to only deal with getting and displaying the information, while the server can focus on storing and manipulating the data.

REST APIs provide a **standardized way of communicating between client and server.** In other words, it doesn't matter how the server is put together or how the client is coded up, as long as they both structure their *communications* according to REST architecture guidelines, using HTTP.

REST API

= the server does *not* save any of the previous requests or responses.

Statelessness makes each request and response very **purposeful** and **understandable**. So, if you're a developer and you see someone else's API request in existing code, you will be able to understand what the request is for without any other context.

REST API

= client cannot tell whether it is connected directly to the end server or an intermediary along the way.

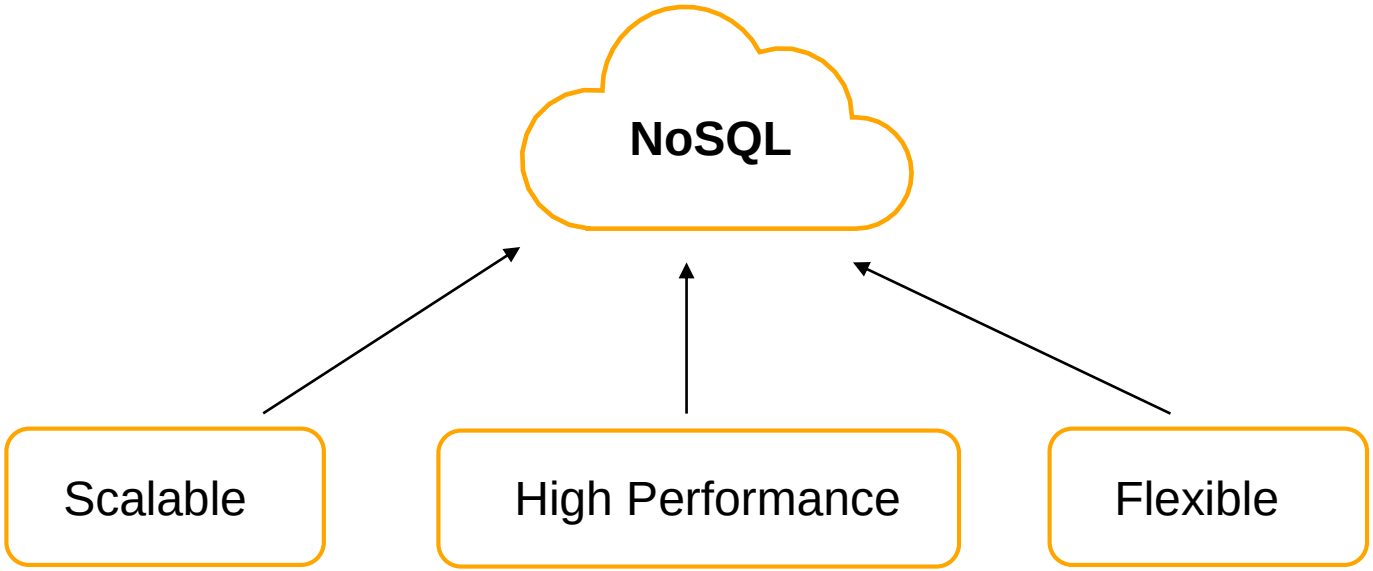
Every component that uses REST **does not have access** to components beyond the specific one it is interacting with. That means a client that connects to an intermediate component has no idea what that component is interacting with afterward. This encourages developers to create independent components, making each one easier to replace or update.

Agenda

1. OAT Overview
2. Front-end
3. REST API
4. NoSQL
5. Dispatcher

NoSQL

- **NoSQL** (“non SQL” or “not only SQL”) databases are non tabular, and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.



NoSQL

NoSQL databases were developed in the late 2000s with a focus on scaling, fast queries, allowing for frequent application changes, and making programming simpler for developers.

Relational databases accessed with SQL (Structured Query Language) were developed in the 1970s with a focus on reducing data duplication as storage was much more costly than developer time. SQL databases tend to have rigid, complex, tabular schemas and typically require expensive vertical scaling.

Why NoSQL?

Handles Large Volumes of Data at High Speed with a Scale-Out Architecture

- SQL databases are most often implemented in a scale-up architecture, which is based on using ever-larger computers with more CPUs and more memory to improve performance.
- NoSQL databases were created in Internet and cloud computing eras that made it possible to more easily implement a scale-out architecture. In a scale-out architecture, scalability is achieved by spreading the storage of data and the work to process the data over a large cluster of computers. To increase capacity, more computers are added to the cluster.
- The scale-out architecture of NoSQL systems provides a clear path to scalability when data volume or traffic grows. Achieving the same type of scalability with SQL databases can be expensive, require lots of engineering, or may not be feasible.

Why NoSQL?

Stores Unstructured, Semi-Structured, or Structured Data

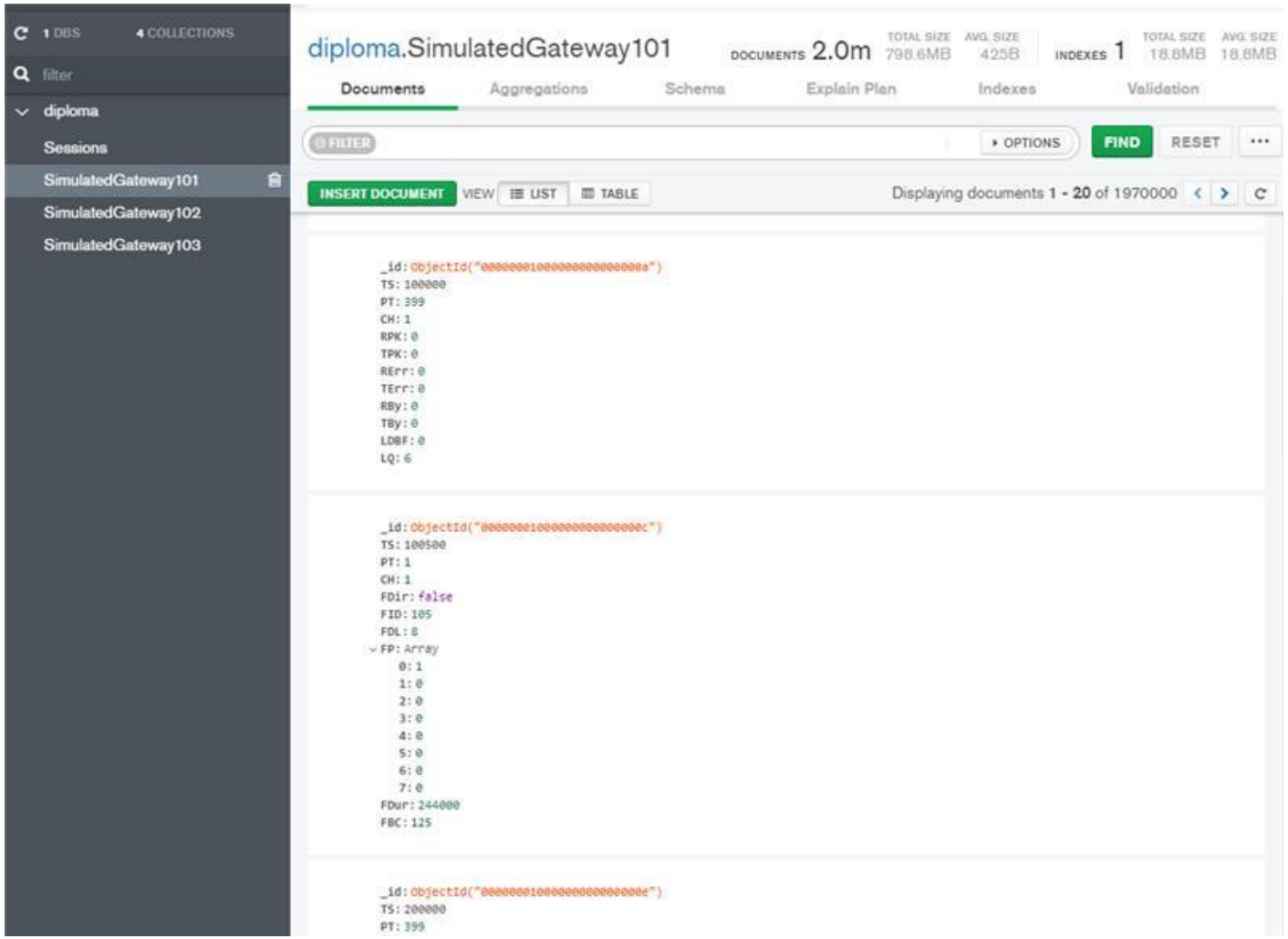
- Relational databases store data in structured tables that have a predefined schema. To use relational databases, a data model must be designed and then the data is transformed and loaded into the database.
- NoSQL databases have proven popular because they allow the data to be stored in ways that are easier to understand or closer to the way the data is used by applications. Fewer transformations are required when the data is stored or retrieved for use. Many different types of data, whether structured, unstructured, or semi-structured, can be stored and retrieved more easily.

Why NoSQL?

Enables Easy Updates to Schema and Fields

- NoSQL databases have become popular because they store data in simple straightforward forms that can be easier to understand than the type of data models used in SQL databases.
- Also, NoSQL databases often allow developers to directly change the structure of the data.
- Document databases don’t have a set data structure to start with, so a new document type can be stored just as easily as what is currently being stored.
- With key-value and column-oriented stores, new values and new columns can be added without disrupting the current structure.
- In response to new kinds of data, graph database developers add nodes with new properties and arcs with new meanings.

NoSQL Database Example



NoSQL Database Example

```
1   _id: ObjectId("0000000100000000000000c")
2   TS : 100500
3   PT : 1
4   CH : 1
5   FDir : false
6   FID : 105
7   FDL : 8
8   FP : Array
9      0 : 1
10     1 : 0
11     2 : 0
12     3 : 0
13     4 : 0
14     5 : 0
15     6 : 0
16     7 : 0
17  FDur : 244000
18  FBC : 125
```

```
ObjectId
Timestamp
Int32
Int32
Boolean
Int64
Int32
Array
Int32
Int32
Int32
Int32
Int32
Int32
Int32
Int32
Int64
Int32
```


```
{
    "_id":"0000000100000000000000c",
    "TS":"100500",
    "PT":1,
    "CH":1,
    "FDir":false,
    "FID":"105",
    "FDL":8,
    "FP":[1,0,0,0,0,0,0,0],
    "FDur":"244000",
    "FBC":125
}
```

Why NoSQL?

Developer-Friendly

- Adoption of NoSQL databases has primarily been driven by uptake from developers who find it easier to create various types of applications compared to using relational databases.
- Document databases such as MongoDB use JSON as a way to turn data into something much more like code. This allows the structure of the data to be under the control of the developer.
- Most NoSQL databases have a strong community of developers surrounding them. This means that there is an ecosystem of tools available and a community of other developers with which to connect.

SQL vs. NoSQL

Relational Data Model

Pros:
- Easy to use and setup
- Universal. Compatible with many tools
- Good at high-performance workloads
- Good at structure data

Cons:
- Time consuming to understand and design the structure of the database
- Can be difficult to scale

Document Data Model

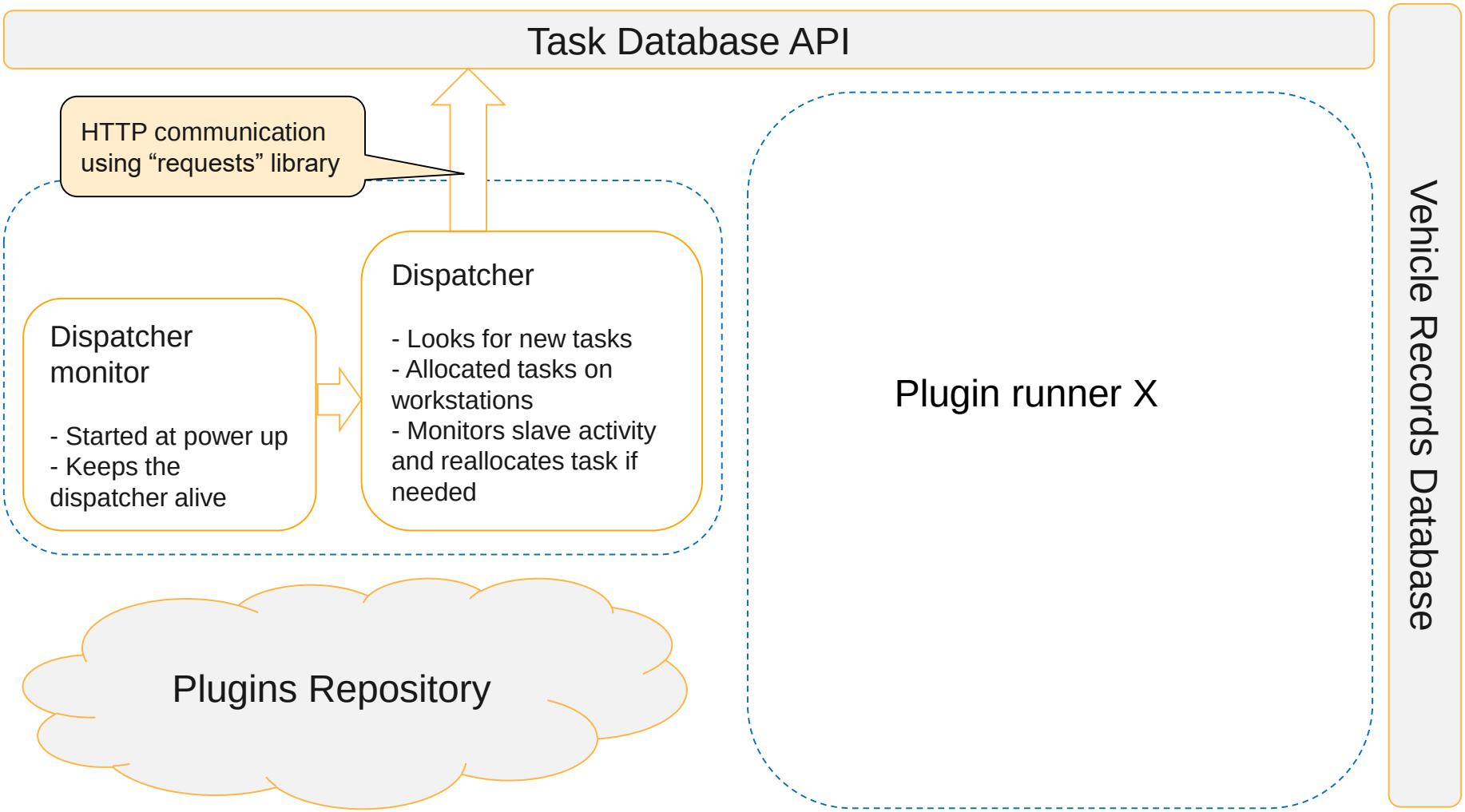
Pros:
- No investment in the design model.
- Rapid development cycles
- In general, faster than SQL
- Runs well on the cloud

Cons:
- Unsuited for interconnected data
- Technology still maturing
- Can have a slower response time

Agenda

1. OAT Overview
2. Front-end
3. REST API
4. NoSQL
5. Dispatcher

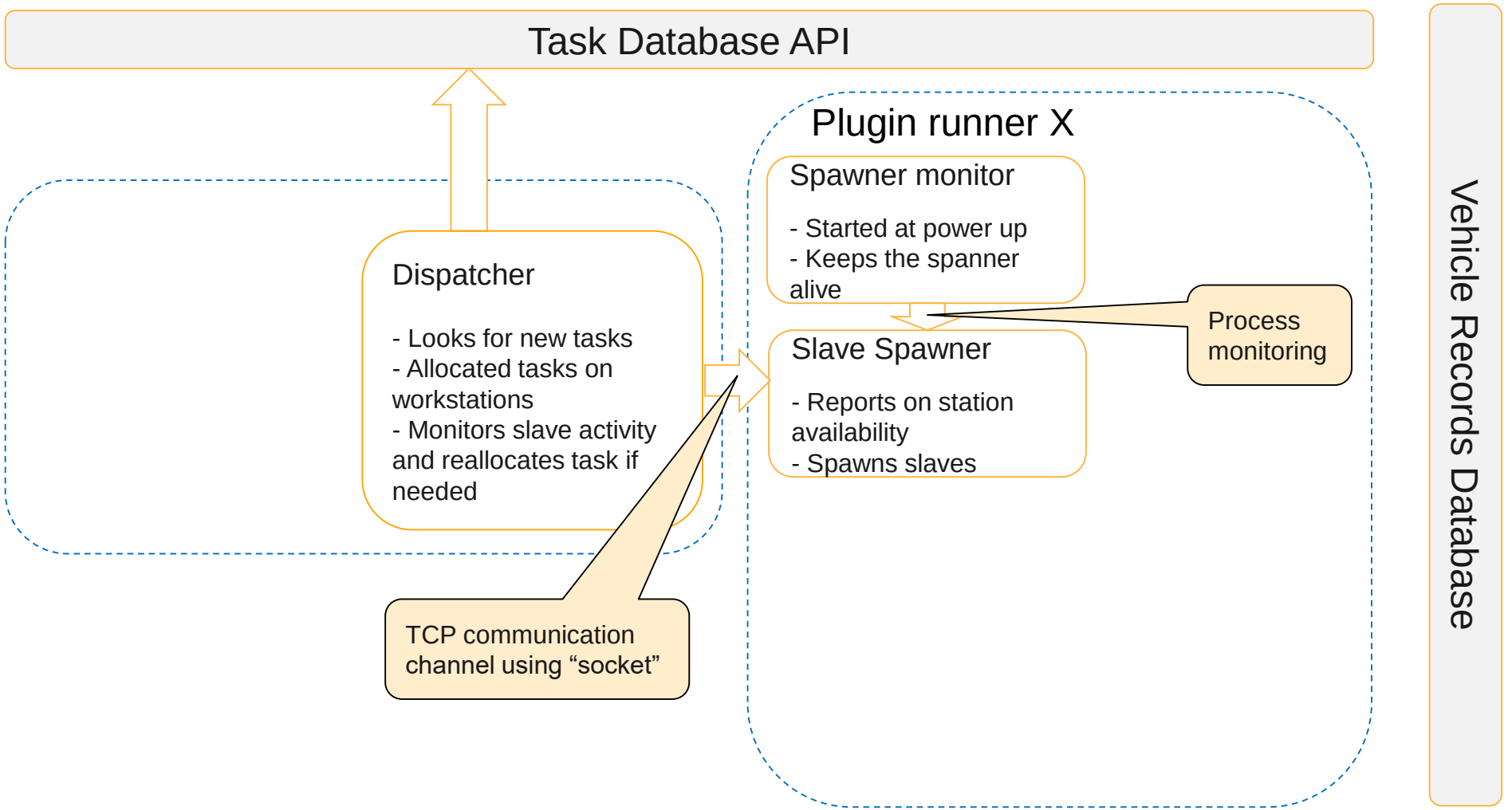
Dispatcher


Task Database API
HTTP communication using “requests” library
Dispatcher monitor
- Started at power up
- Keeps the dispatcher alive
Dispatcher
- Looks for new tasks
- Allocated tasks on workstations
- Monitors slave activity and reallocates task if needed
Plugin runner X
Vehicle Records Database
Plugins Repository

Dispatcher

```python
import requests

def isPositiveResponse(code):
    if 200 <= code < 400:
        return True
    return False

def GET(address):
    try:
        result = requests.get(address, timeout=10)
    except Exception as ex:
        return None, str(ex.args)

    if not isPositiveResponse(result.status_code):
        return result, 'Negative server response: ' + str(result.status_code) + ' for get request on address: ' + address

    return result, None
```

Handling multiple execution systems

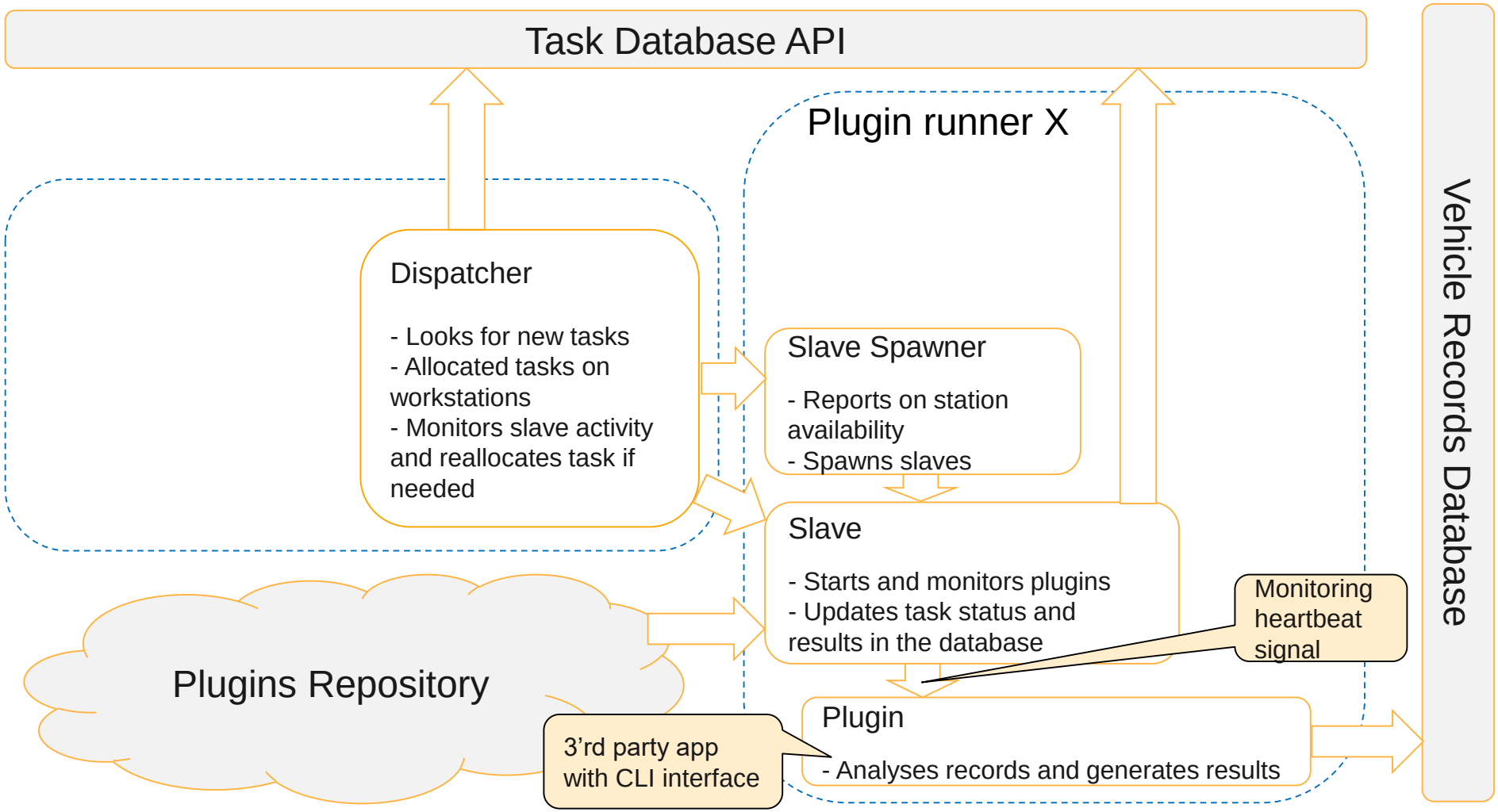


Handling multiple execution systems

```
class TCPClientSocket:

    addr = None
    socket = None

    def __init__(self, clientSocket, addr):
        self.socket = clientSocket
        self.addr = addr
        self.listening = True

    def send(self, data):
        if self.listening:
            self.socket.send(data)
```

```
def startParallelProcess():
    spawner = Process(target=Main().mainLoop)
    spawner.start()
    return spawner


if __name__ == '__main__':
    spawner = startParallelProcess()
    while True:
        if not spawner.is_alive():
            spawner = startParallelProcess()
        sleep(LAUNCHER_CYCLE_TIME)
```

Running 3’rd party code



Running 3’rd party code

```
def monitorLoop(self):
    self.log.info('Starting plugin monitoring loop.')
    while self.pluginIsRunning():
        self.lasAliveTimeStamp = time()
        sleep(self.loopCycleInSeconds)
    self.log.info("Plugin has finished execution.")
```

```
def pluginIsRunning(self, heartBeatAvailabilityTimeout = 60):
    lines = self.readDataFromPluginHeartbeat(heartBeatAvailabilityTimeout)
    if lines is None:
        return False

    plugin = PluginHeartbeat()
    error = plugin.fromString(lines[-1])
    if error is not None:
        self.log.error("Could not interpret plugin heartbeat data")
        self.log.error(error)
        return False

    missingAttributeError = False
```

Plugin control