Executive Summary

By JD Margulici :: linkedin.com/in/jdmargulici/ :: April 12th, 2022

Abstract

This document summarizes the scope of the Anaximander framework and what motivates its development. The framework is a Python library that targets data-intensive enterprise applications in industrial domains. Its primary goal is to enhance developers' productivity by abstracting data engineering from structures and transformations modeling. Anaximander relies on rich model semantics and metaprogramming to automate infrastructure provisioning and data access in selected cloud environments.

Motivation

As described in the Anaximander Statement of Purpose, my motivation for this work originates from my experience as an application developer in different domains: transportation operations and planning (e.g. airport terminals, highways...), traveler information (e.g. travel time estimates in an urban transportation network), manufacturing (e.g. production and assets monitoring), as well as electronic commerce and marketing. The insights gleaned from these projects can be summed up as follows:

- Cloud computing has considerably lowered the barrier to building robust applications that are scalable, secure, and reliable. However, implementing a cloud-based architecture still requires a large code base, specialized data engineering skills, and numerous service interfaces. This work pulls resources away from solving business logic, data science and machine learning issues that create informational value.
- There exist patterns of data structure and transformation that transcend application domains yet are not explicitly captured in programming tools. This is particularly true of spatiotemporal data, despite its universal character -think for instance of the dual equivalence between segment-based state representations and location-based sampling for linear infrastructure such as roadways or pipelines; an even more straightforward example is the need to perform analytics by joining data streams against intricate time schedules, which is common across industrial domains.
- Infrastructure-as-code makes it possible to provision and deploy cloud services programmatically. By recognizing the fact that data-intensive applications employ a largely predictable stack and by leveraging universal patterns as described above, we can generate infrastructure code automatically based on business and science-driven model and transformation declarations.

The Anaximander framework proposes a modeling approach that merges object-oriented programming and the data science toolkit to enable concise declarations of data, metadata, and ultimately transformation pipelines. The overarching goals are to boost developer productivity and shift efforts from systems engineering to application engineering, as well as to facilitate data integration across domains with unifying semantic primitives expressed as archetypes. Anaximander is distributed freely under the permissive MPL2 open-source license for maximum impact.

Use Cases and Scope

Anaximander takes the form of a Python library that provides building blocks for so-called data-intensive application backends. These may be defined as enterprise software applications that perform ongoing analytical processing with substantial data volumes, as opposed to transaction-oriented applications. To be more specific, here are some key assumptions made by the framework about the type of applications that it targets:

- The application continually integrates sources of data (either as streams or periodic batches), and similarly, it continually outputs data streams or periodic reports. It is always on and hence we would like it to operate in a fault-tolerant manner -meaning servers or disks can fail without impacting system uptime or data integrity. Along the same lines, the system cleanly separates the application code, which contains the business logic, from the computing infrastructure on which it is running.
- The application's logic involves complex event processing and data transformations -think, how do we turn streams of individual GPS logs into reliable travel time estimates? Further, the nature of the data is such that time, and optionally space, are first-order concepts that permeate though most if not all data sets involved.
- The application's output is made available to users through interactive dashboards that provide various summary metrics with drill-down and aggregation features in graphical form, or to other downstream applications through a web-based Application Protocol Interface -with, again, both streaming and batch options.

Typical use cases include Internet-of-Things backends, AR/VR analytics, and more generally enterprise software that runs on data streams, cutting across industrial domains such as utilities, transportation, environmental monitoring, epidemiology, agriculture, manufacturing, or supply chain management -to sum up, digital twins of physical world environments.

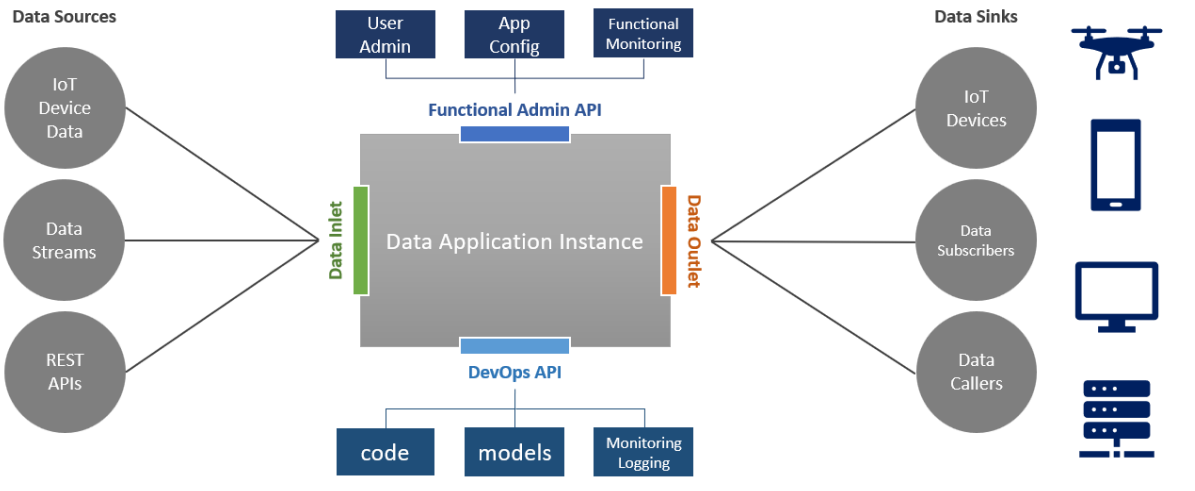

Figure 1 - Operational concept of a typical data application instance

Such applications implement a predictable access pattern, as depicted in Figure 1. At the highest level, a data-intensive applications' interfaces can be broken down into four groups:

1. A data inlet that feeds incoming data to the system, either by fetching it or through a publish/subscribe mechanism.
2. A data outlet that works in reverse, i.e., feeding outgoing data to clients, again either through a query engine or by pushing events.
3. A set of developer interfaces: these comprise code and model updates, as well as system logs and monitoring functions.
4. An administration interface for operators, allowing changes in configuration and functional monitoring -typically provided to power users or business managers in the form of a graphical user interface.

Behind these interfaces, the system also implements a predictable array of technologies and processes, as indicated in Figure 2. These comprise storage and computing services, following increasingly more established patterns: relational entities that describe the operating environment, events that are processed as streams or batches and get persisted in a sliding fashion from hot storage to cold storage, and machine learning capabilities that enable complex inferences based on models calibrated and updated from historical data.

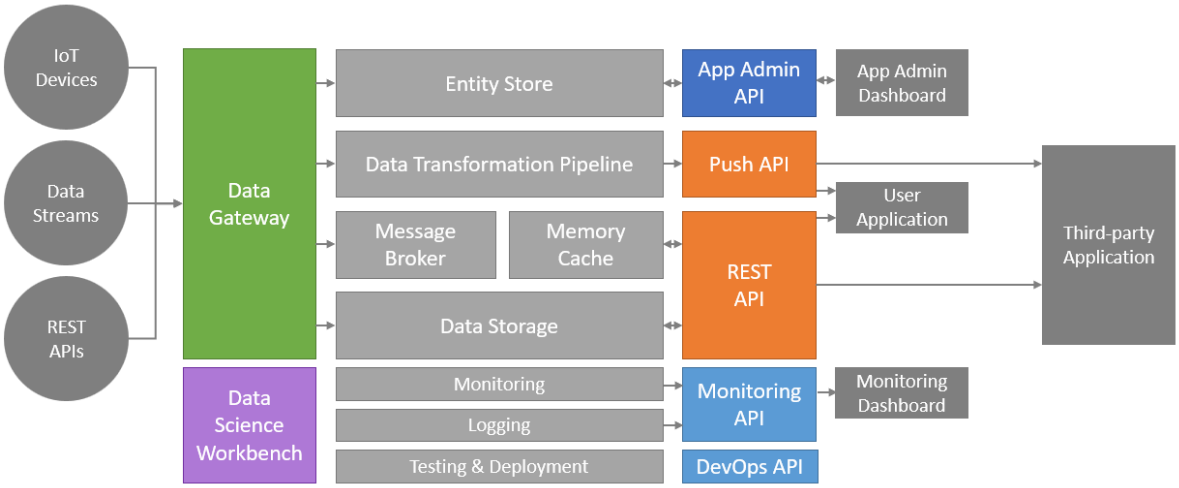

Figure 2 - Anatomy of a data-intensive application

There exists an abundance of enabling technologies and serverless cloud services to implement these patterns. Anaximander deeply integrates with selected technologies and services to abstract the infrastructure away from data models and transformations. In concrete terms, a developer may declare a data model for, say, location fixes sourced from a fleet of connected vehicles, and decide on three tiers of storage based on recency: hot storage in a memory cache, medium storage in a columnar database, and cold storage in a data lake. Anaximander would programmatically generate read and write queries to those three storage backends, along with basic administrative commands such as table creation, based on the model declaration, and expose them as object methods. Hence one can think of the framework as a generalization of object-relational mapping technology to a polyglot system comprised of multiple databases. Yet this is only one facet of the integration scope, which also extends to programmatically generating data APIs, automating data validation and basic ETL patterns, and out-of-the-box data representation and dashboard integration.

The enabling concept for all this automation is what the framework refers to as data archetypes. Every data model declaration extends one of the archetypes, and these archetypes govern data access, data transformation and data representation. Models can augment the basic archetypes with traits, which confer them with additional properties. For instance, the vehicle location fixes used in the previous example follow the Sample archetype (i.e., pseudo-regularly polled event messages), and implement the GeoLocation trait (which gives the model spatial properties, such that collections of fixes would implement a bounding box method). With this level of semantics, the modalities in which the data is used and transformed become quite predictable, with makes automation possible. Of course, as is the case with any templating system, automation only covers so much, but developers have access to the system's underlying technologies to add use cases -for instance, if the outbound data API needs to expose a complex query, this would still be programmed traditionally using a web or API framework. Anaximander's goal is to provide a core architecture around which to build (that alone can be a significant simplification), and to aim for the 80/20 rule when it comes to automation -that is, automate the predictable use cases, which presumably represent 80% of the effort, and leave the remaining 20% to programming outside the framework.

Status

As of this writing, the Anaximander framework has been under development for several months and an alpha release is planned by the end of April 2022. So far, the effort has been concentrated on the archetype system and its implementation in the Python programming language. Hence the alpha release will enable a proof of concept and demonstrate the creation and manipulation of data objects. The next phase will consist in integrating key backend technologies to enable cloud data storage and access -most likely in the Google Cloud Platform. Once this is complete, Anaximander will play its initial role as a richly semantic object-storage mapper, linking various databases to an object-oriented interface, including metadata-enabled Dataframes that feature automated validation and entity relationships.

From that point on, besides incorporating additional backends as needed, the work will progress along three axes:

- Functional semantics: incorporating physical units, levels of measurement, geospatial support, stateful entities support, specialized plotting -and even more down the road.
- Administrative semantics: model versioning, data lineage, and fined-grained data authorization are the initial priorities.
- Data transformations: this is still prospective, particularly when it comes to integration with streaming frameworks -a much needed simplification in my view, but a possibly complex endeavor. As a first step, the framework will add summarization methods to data archetypes, providing basic analytics in an object-oriented fashion.

The project is actively looking for early adopters, contributors, advisors, and domain experts, as well as seeking use cases on which to build proof-of-concepts and applications. If interested in learning more, please contact JD Margulici: jdm@novavia.us.