vision Service Architecture

Jason A. Grafft

# *vision* Service Architecture

## Table of Contents

1. Global Service Architecture
   1. Global Communication
   2. External Communication
   3. Internal Communication
2. Microservices
   1. Master
   2. LogManager
   3. FileSystem
   4. Database
   5. AuthManager
   6. Capture
   7. Streaming
3. Traits
   1. Credentials
   2. Location

## Global Service Architecture

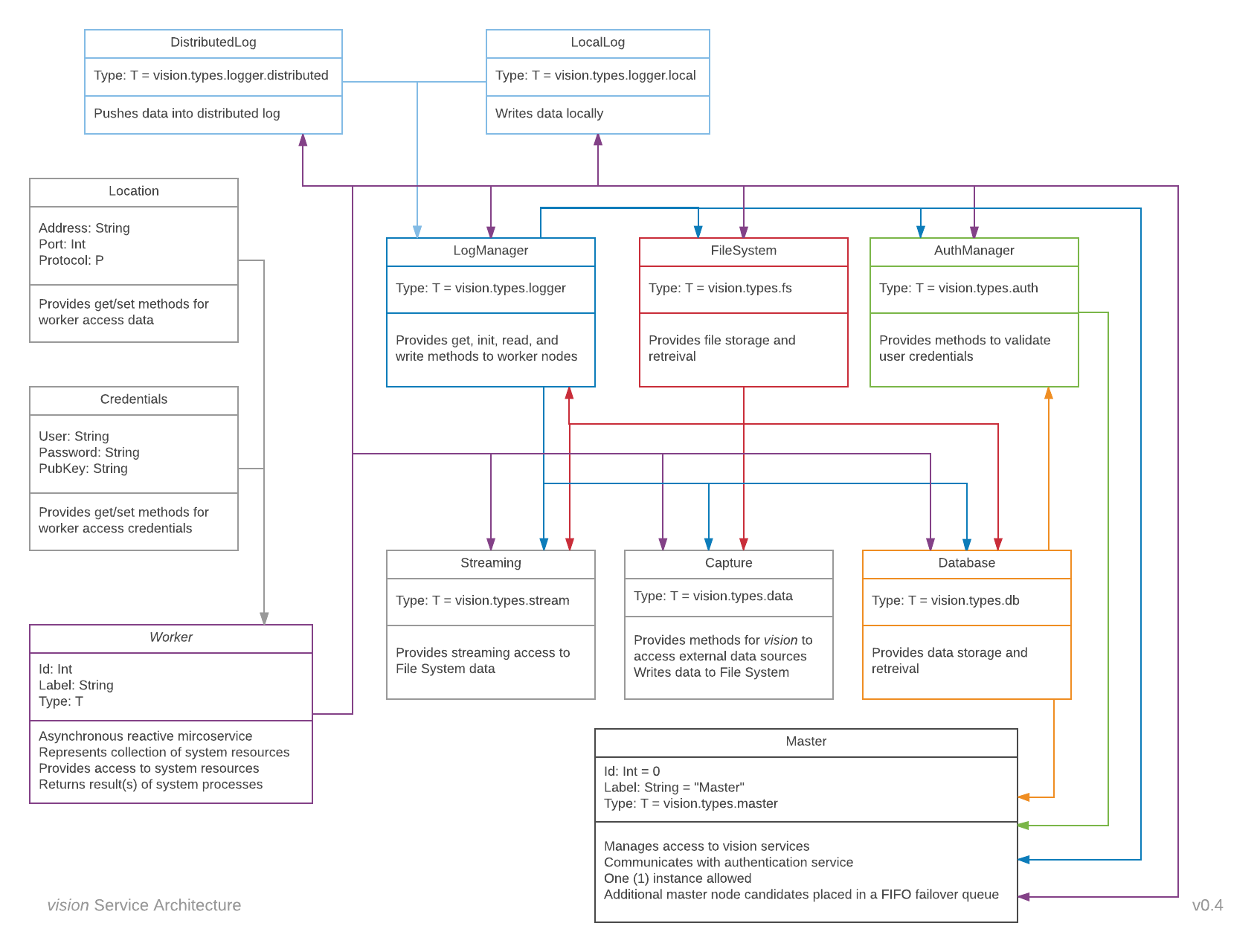


Figure 1: vision Global Service Architecture

The *vision* data service is an asynchronous, distributed system composed of independent, reactive microservices called Workers, who collaborate to deliver a service to end-users. Users interact with a Master worker, responsible for activation, coordination, and oversight of all *vision* workers. Communication between the Master and other workers occurs at a level inaccessible to users, who are limited to making requests of the Master worker. A controlled, central access point helps normalize state across the distributed system, reducing the potential for bugs and errant behavior.

Each worker must be associated with a **local** LogManager (except other LogManagers), responsible for writing events to disk, providing read-only access for log requests, and pushing events to a distributed, eventual consistency data service.

### Preferred Methods and Patterns

*vision* utilizes the actor model implemented by Akka,[[1]](#footnote-26) and, in general, follows Akka’s recommended design patterns. Wherever possible, *vision* microservices communicate asynchronously via non-blocking *methods* **without** the use of futures, a *pattern* which introduce significant complexity and performance concerns.

From an outside perspective, microservices are immutable. This is the preferred design pattern for preserving scalability and the predictability of state in a distributed system. With a compelling reason, mutabilitity may be used within an individual microservice, but this design decision is **not** accessible or viewable by an external process.

A **local** LogManager worker must be associated with each and every *vision* microservice, and the *vision* service must implement one (1) global LogManager microservice. Thus all event data is stored local to each microservice and pushed to an eventual consistency distributed log. Individual microservices utilize their local LogManager, which is resposible for communicating with the global LogManager as needed.

### External Communication

Users interact with *vision* through a CSS, HTML, and JavaScript web browser interface. This interface passes user requests to the *vision* Master worker as JSON packets, which are received by Akka HTTP[[2]](#footnote-29), serialized with Spray JSON[[3]](#footnote-30), then parsed by event-handling logic. Essential record-keeping information is forwarded to the connected LogManager, and the request to the appropriate *vision* Worker. Once these comunications are made, the Master worker is absolved of responsibility, and it is up to individual *vision* workers to communicate with other services and pass back results, as appropriate, to the end-user.

### Internal Communication

Communication between *vision* workers is carried out by Akka Actors[[4]](#footnote-33) over TCP. Since all *vision* workers inherent data models from a common class, it is generally safe for *vision* Workers to assume noncorrupted outgoing communications to other *vision* Workers can be evaluated correctly. This facilitates wider use of the non-blocking Tell (fire-forget) method, which offers better concurrency and drastically reduces the complexity of the communication loop.

## Microservices

*vision* microservices are independent, asynchronous services that provide methods to access the collection of system resources they represent. They are intended to be highly specialized, easily replicatable units which are literate in the communication protocols used by *vision*, and will often serve as a bridge between *vision* users and third-party services such as computing frameworks, databases, file formats, system processes, et cetera.

### Master

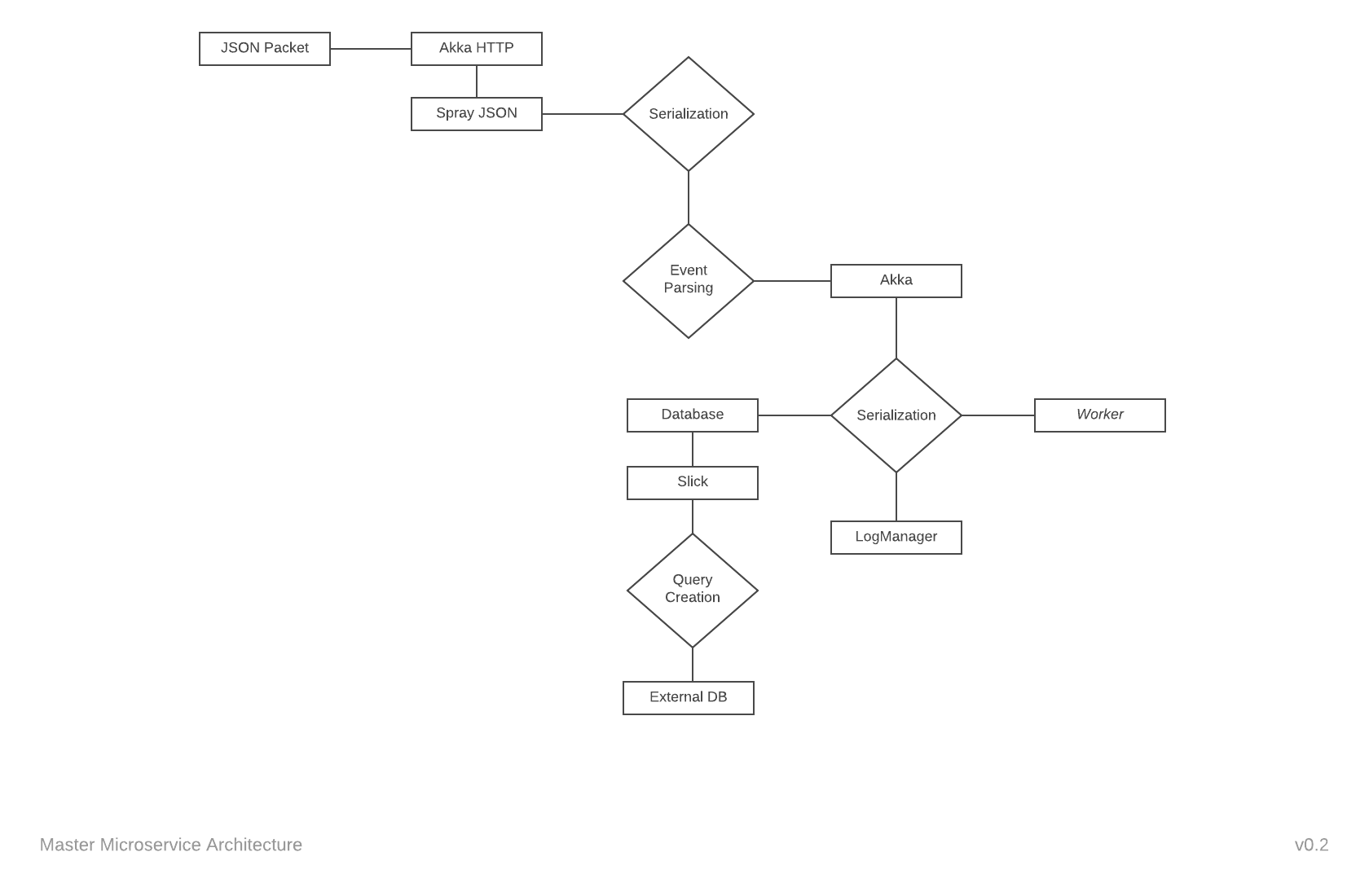


Figure 2: Master Microservice Architecture

*Associated Workers:* Database, LogManager

The Master microservice coordinates the effort of all *vision* workers, and serves as the sole point-of-contact for end users. All user requests are received and disseminated to *vision* workers at the discretion of the Master worker. It is allowed to hand off communication tasks to other workers, thus providing implicit permission for the designated worker(s) to communicate with external entities; in these cases, the worker is required to be literate in the necessary communication protocols.

The Master microservice is allocated a dedicated, local Database worker for storage and retrieval of worker access information. Only the minimum information set required to communicate with an individual worker should be committed to the Master’s database, all other needed credentials should be passed to or retreived by the activated worker.

### LogManager

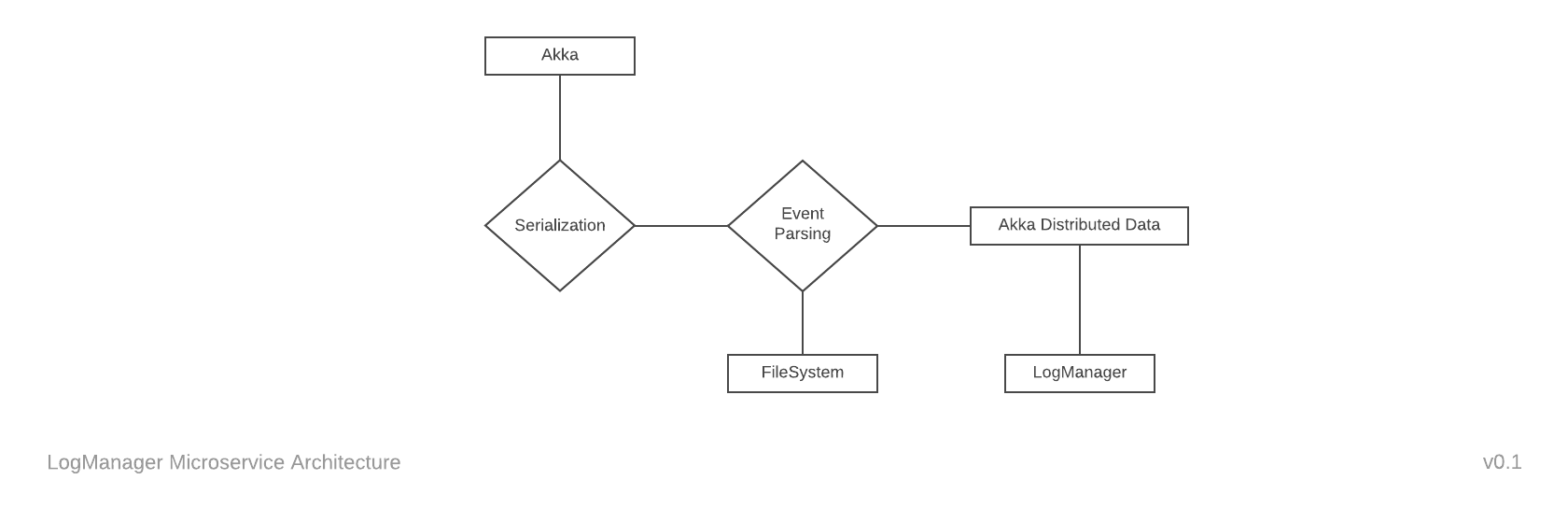


Figure 3: LogManager Microservice Architecture

*Associated Workers:* FileSystem, LogManager (global)

The LogManager microservice creates, maintains, and provides access methods for append-only log files of the events passing through connected workers. It also pushes a copy of these event records to a global LogManager via Akka Distributed Data,[[5]](#footnote-39) an eventual consistency service based on Conflict-free Replicated Data Types.[[6]](#footnote-40) Information appended to these logs is used by *vision* for dynamic allocation of service resources, and thus essential to the stability of *vision* itself.

Redundancy is employed as a failsafe and to ensure robust data capture. The increase in complexity from implementation of an eventual consistency pattern is acceptable given the value of the information.

### FileSystem

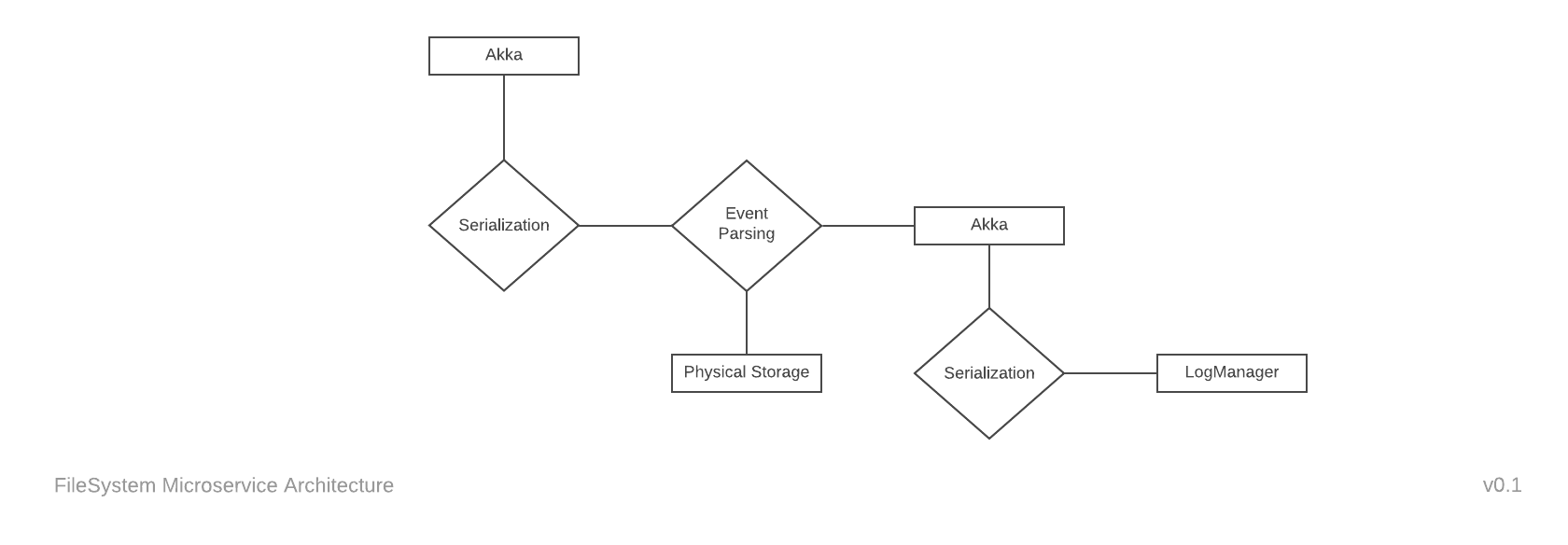


Figure 4: FileSystem Microservice Architecture

*Associated Workers:* LogManager

The FileSystem microservice reads, writes, and provides access to physical file systems. The primary purpose of FileSystem workers is to ensure separation of concerns within *vision*. They are expected to be literate in a single file format and provide methods for the essential operations of their connected *vision* microservices.

### Database

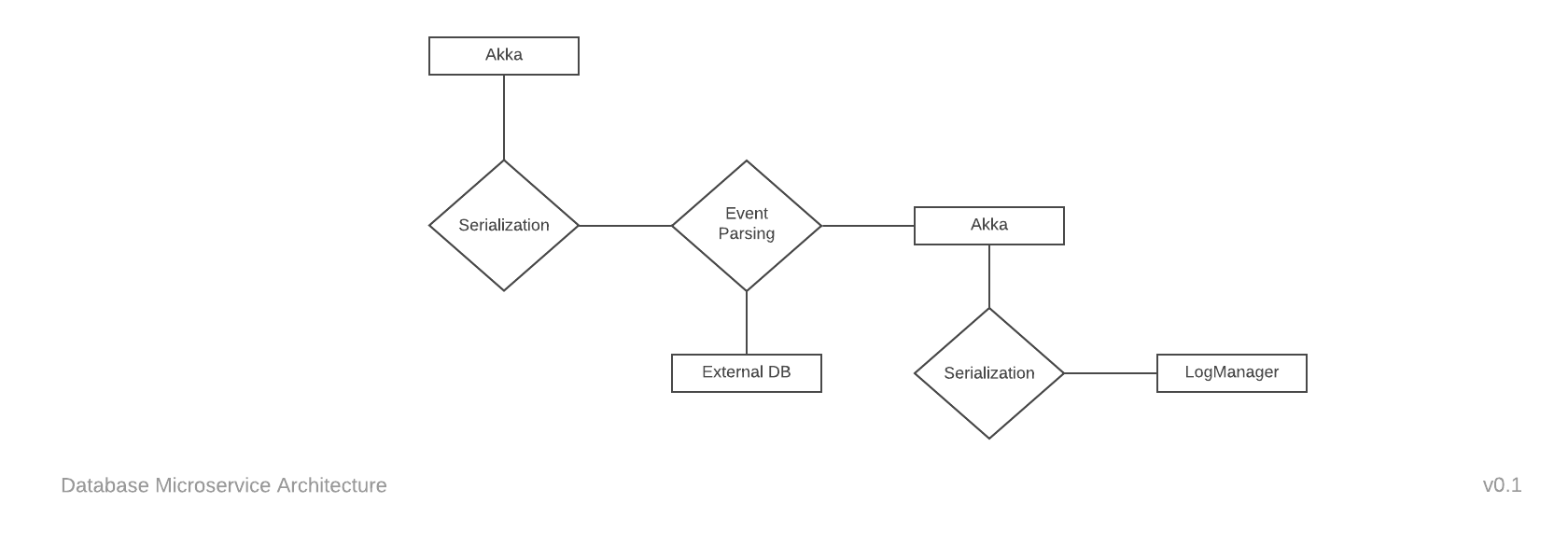


Figure 5: Database Microservice Architecture

*Associated Workers:* LogManager

The Database microservice provides methods for *vision* microservices to interact with third-party database services. The primary purpose of Database workers is to ensure separation of concerns within *vision*. They are expected to be literate in a single database protocol and provide methods for the essential operations of their connected *vision* microservices.

### AuthManager

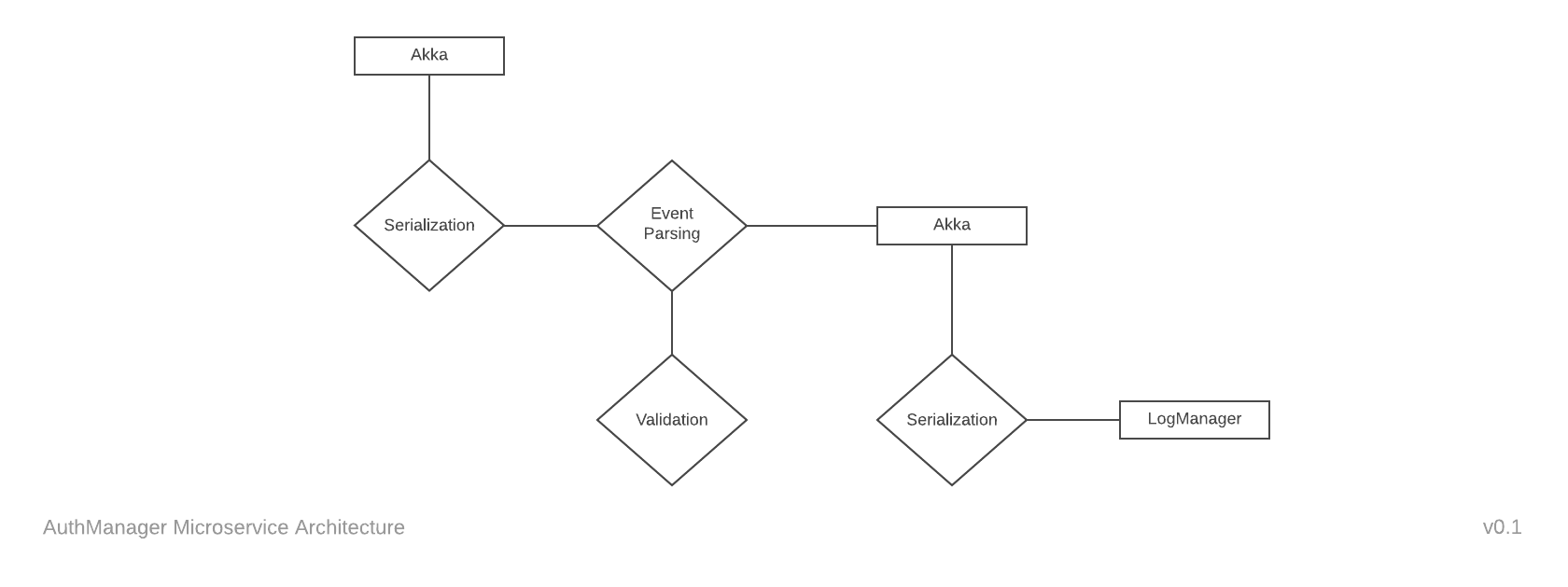


Figure 6: AuthManager Microservice Architecture

*Associated Workers:* LogManager

The AuthManager microservice provides methods for *vision* microservice to validate credentials passed in by associated workers. They are expected to be literate in a single authentication method, and return the minimum information set required to demonstrate validity or invalidity.

### Capture

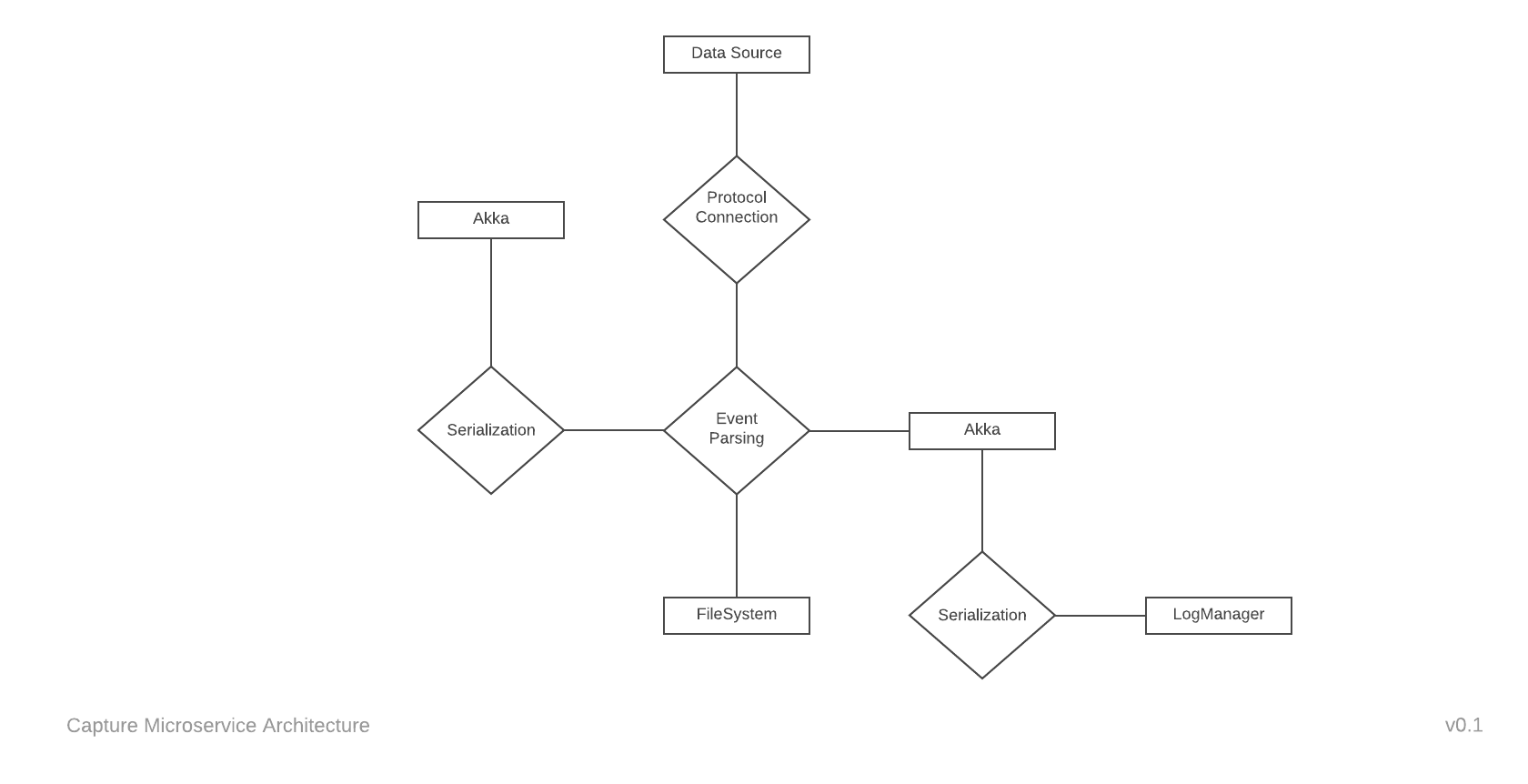


Figure 7: Capture Microservice Architecture

*Associated Workers:* FileSytem, LogManager

The Capture microservice provides an interface for the **intake** of data **from** sources **external** to *vision*. They are expected to be literate in a single communication protocol and a single strategy of handling the data to be written to disk. Confirmation, validation, and the like are optional and should only be implemented if a need can be demonstrated. Capture workers are **write-only**.

### Streaming

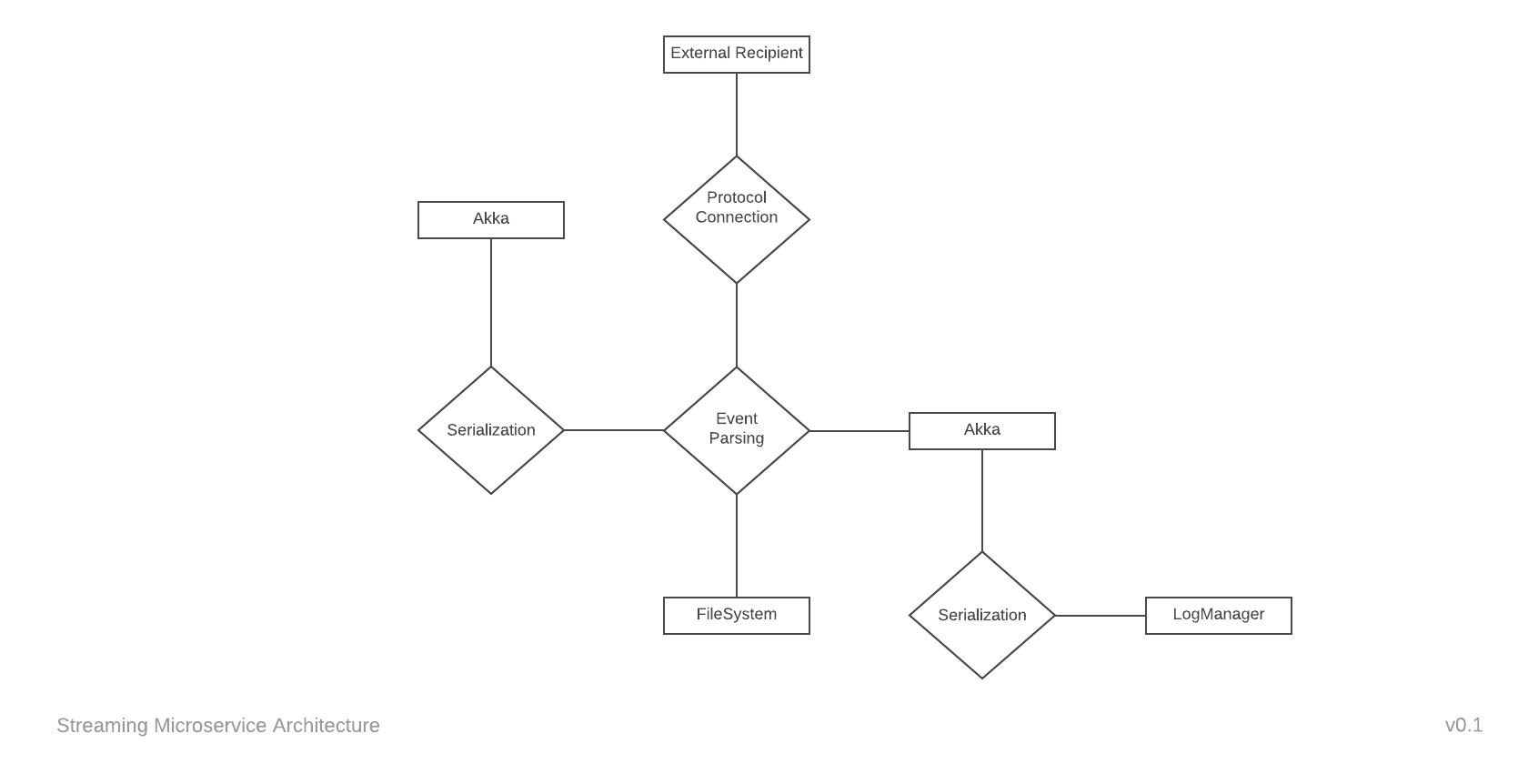


Figure 8: Streaming Microservice Architecture

*Associated Workers:* FileSytem, LogManager

The Streaming microservice provides and interface for the **output** of data **to** sources **external** to *vision*. They are expected to be literate in a single communication protocol and a single strategy of handling the data to be read from a disk. Confirmation, validation, and the like are optional and should only be implemented if a need can be demonstrated. Streaming workers are **read-only**.

## Traits

Traits represent information *vision* workers require for interacting with associated workers and third-party services. They are optional, and should only be used when necessary.

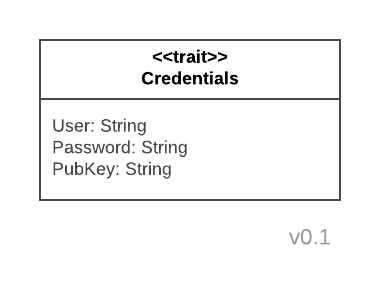


Figure 9: Credential Trait

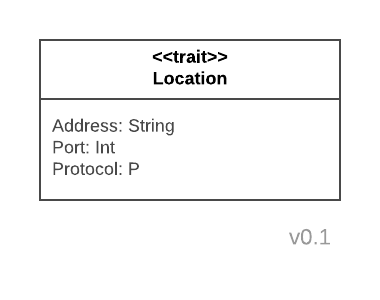


Figure 10: Location Trait

## References

Lightbend, Inc. “Akka Modules Documentation,” 2017. <https://akka.io/docs/>.

———. “spray-json,” 2017. <https://github.com/spray/spray-json>.

Shapiro, Mark, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. “Conflict-Free Replicated Data Types.” *SOMETHING IN FRENCH*, August 25, 2011. <URL>.

1. Lightbend, Inc, “Akka Modules Documentation,” 2017, <https://akka.io/docs/>. [↑](#footnote-ref-26)
2. Lightbend, Inc. [↑](#footnote-ref-29)
3. Lightbend, Inc, “spray-json,” 2017, <https://github.com/spray/spray-json>. [↑](#footnote-ref-30)
4. Lightbend, Inc, “Akka Modules Documentation.” [↑](#footnote-ref-33)
5. Lightbend, Inc. [↑](#footnote-ref-39)
6. Mark Shapiro et al., “Conflict-Free Replicated Data Types,” *SOMETHING IN FRENCH*, August 25, 2011, <URL>. [↑](#footnote-ref-40)