Architecture Decision Record: Configuration

Context

Arachne has a number of goals.

1. It needs to be *modular*. Different software packages, written by different developers, should be usable and swappable in the same application with a minimum of effort.
2. Arachne applications need to be *transparent* and *introspectable*. It should always be as clear as possible what is going on at any given moment, and why the application is behaving in the way it does.
3. As a general-purpose web framework, it needs to provide a strong set of default settings which are also highly overridable, and *configurable* to suit the unique needs of users.

Also, it is a good development practice (particularly in Clojure) to code to a specific information model (that is, data) rather than to particular functions or APIs. Along with other benefits, this helps separate (avoids "complecting") the intended operation and its implementation.

Documenting the full rationale for this "data first" philosophy is beyond the scope of this document, but some resources that explain it (among other things) are:

- Simple Made Easy - Rich Hickey
- Narcissistic Design - Stuart Halloway
- Data Beats Functions - Malcolm Sparks
- Always Be Composing - Zach Tellman
- Data > Functions > Macros - Eric Normand

Finally, one weakness of many existing Clojure libraries, especially web development libraries, is the way in which they overload the Clojure runtime (particularly vars and reified namespaces) to store information about the webapp. Because both the Clojure runtime and many web application entities (e.g servers) are stateful, this causes a variety of issues, particularly with reloading namespaces. Therefore, as much as possible, we would like to avoid entangling information about an Arachne application with the Clojure runtime itself.

Decision

Arachne will take the "everything is data" philosophy to its logical extreme, and encode as much information about the application as possible in a single, highly general data structure. This will include not just data that is normally thought of as "config" data, but the structure and definition of the application itself. Everything that does not have to be arbitrary executable code will be reflected in the application config value.

Some concrete examples include (but are not limited to):

- Dependency injection components
- Runtime entities (servers, caches, connections, pools, etc)
- HTTP routes and middleware
- Persistence schemas and migrations
- Locations of static and dynamic assets

This configuration value will have a *schema* that defines what types of entities can exist in the configuration, and what their expected properties are.

Each distinct module will have the ability to contribute to the schema and define entity types specific to its own domain. Modules may interact by referencing entity types and properties defined in other modules.

Although it has much in common with a fully general in-memory database, the configuration value will be a single immutable value, not a stateful data store. This will avoid many of the complexities of state and change, and will eliminate the temptation to use the configuration itself as dynamic storage for runtime data.

Status

Proposed

Consequences

- Applications will be defined comprehensively and declaratively by a rich data structure, before the application even starts.
- The config schema provides an explicit, reliable contract and set of extension points, which can be used by other modules to modify entities or behaviors.
- It will be easy to understand and inspect an application by inspecting or querying its configuration. It will be possible to write tools to make exploring and visualizing applications even easier.
- Developers will need to carefully decide what types of things are appropriate to encode statically in the configuration, and what must be dynamic at runtime.