RFC: [descriptive title]

Owner: [your name]
Status: [proposed | accepted | canceled]
Comment by [date] (if still in proposed state)

Foreword

> This template is intended to help solicit feedback on proposals and designs. It presents an outline to help organize your ideas, and provides guidelines and examples of what should go in each section. **The outline should be considered a starting point and not a strict format that must be followed**. Guidance on the RFC process can be found here.

Identify your audience before you start writing. These are the people who are impacted by what you're proposing, subject matter experts, or people who can make the proposal happen/not happen. It may be Product, a particular team, tech-at-large, or some other group. Address that audience at the beginning of your doc.

In one sentence, summarize what's being proposed. Then briefly describe how the document will make a case for the proposal. This section should be no more than a couple sentences.

A meta-example is provided in the grey box above. We'll use real-world examples from here on out:

> The purpose of this document is to socialize engineering decisions involving investment in the Fadoodle Service. We present the current state of the Service, and the problems it causes for our API. We propose a solution, along with metrics for success.

Motivation

Why is this change being proposed? You first need to **state the problem and make a case that the problem is worth solving**.

- Focus on the impact of the problem - monetary expense, wasted time, unnecessary risk, customer unhappiness, etc.
- Quantify the impact. Metrics are great, estimates are ok too.
- State the immediate cause of the problem. Some potential causes could be scaling issues, missing functionality, or broken process.
- Use terminology that's relevant to your audience.

Don’t write too much. This section should be less than one page - one or two paragraphs or a couple bullet points. Do not go in depth about the cause of the problem. If the problem is unhappy customers because of scaling-induced slowness, it’s ok to simply say “because of scaling issues”.

Don’t describe your solution in this section. It’s ok to allude to a better way or suggest we could achieve a net-positive gain by fixing the problem, but don’t spend more than one or two sentences talking about solutions. You’ll go into more detail later.

> […] to support this usage, the cassandra-rhubarb cluster has been scaled to 12 c3.2xlarge nodes [ed: this doc’s audience would recognize that as expensive]. Despite this large expenditure and attempts to configure the cluster differently, the latencies for the RPC calls regularly exceed several hundred milliseconds and Cassandra often experiences load alerts.

> After deprecating a service, there is always difficulty with knowing when to actually terminate the service completely. It commonly continues to be run in a hobbled state until it either breaks beyond what anybody is willing to repair or it costs too much to maintain. These services can also end up being huge security holes because they’re unmaintained and nobody wants to touch them. PiedPiper has many internal services that have been deprecated, but continue to run because there hasn’t been enough communication between teams to finally kill the service.

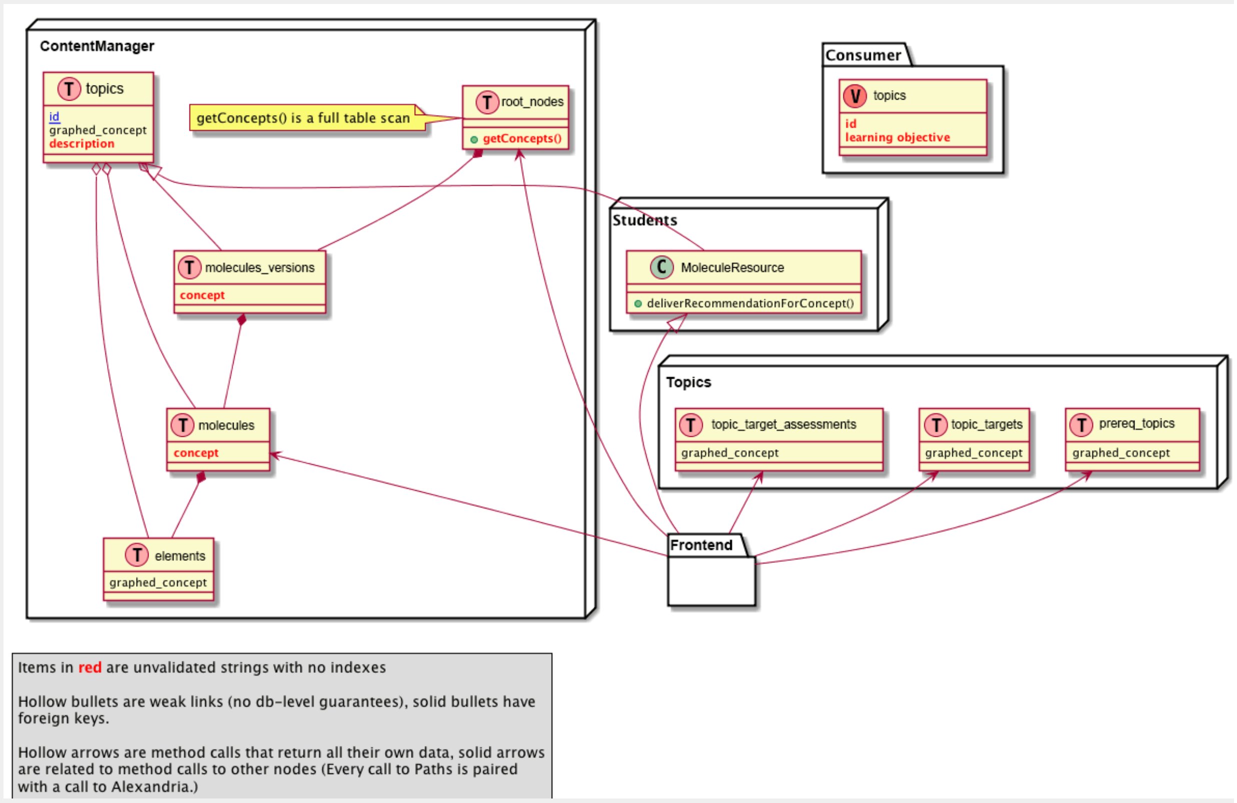
> 1. Graph retrieval times for certain graphs are far outside the acceptable range. The Zowzy service stores graphs as a series of commits, and graphs are retrieved by fetching and applying each commit in sequence. For a small set of graphs the number and size of commits is large enough that this strategy breaks down.
> 2. Services that use BigCustomer graphs are reliant on an on-disk Berkeley cache to keep retrieval times within acceptable ranges. This prevents these services from making effective use of continuous deployment.
> 3. The CassandraZowzy cluster costs around $11,000 per month across envs.
> 4. Zowzy is very complex and is only well understood by a couple of people, which makes maintaining and performance tuning challenging.

Background

The purpose of this section is to **provide context for the reader to understand what you’re proposing**. Describe the current state of the world to illustrate the scope and nature of the problem - go into detail about the immediate causes mentioned in the Motivation section. Some things to talk about could be data schemas, access patterns, race conditions, unnecessary dependencies, or duplicated logic (this is not an exhaustive list).

People are lazy and don’t read, so you’ll need to prioritize information. Don’t bury important details alongside less important information. Pictures are good too.

The level of detail in the section is highly dependent on your target audience and the issues you’re describing. If the motivation is missing functionality, you might skip this section entirely. **When in doubt err on the side of being too concise and put deeper background info in the appendix.**



Items in red are unvalidated strings with no indexes

Hollow bullets are weak links (no db-level guarantees), solid bullets have foreign keys.

Hollow arrows are method calls that return all their own data, solid arrows are related to method calls to other nodes (Every call to Paths is paired with a call to Alexandria.)

Creating and updating tokens requires that we fetch the latest non-disabled access or refresh token for the account in question. This is an unfortunate consequence of the fact that we allow accounts to possess more than one refresh token (active or otherwise). The resulting query includes an aggregate over refresh_token table entries with three index conditions:

```
SELECT id as dbIdentifier, foo_id as fooId, expiration,
account_id as accountId, token as refreshToken FROM tokens
WHERE account_id = :accountId and expiration (select
max(expiration) from refresh_token where expiration > :timeNow
and account_id = :accountId and disabled = false)
```

Depending on the exact dataset, the query optimizer tends to either (1) use only one index, usually disabled or (2) perform an index merge against disabled and account_id. As we have seen, it is often the case that neither approach is efficient.

Requirements

AKA success criteria. Briefly (one or two sentences) state your solution, and then talk about all the good stuff stakeholders will get once the proposal is implemented - **impact of the solution**. This section is a counterpart to the Motivation section and the same guidelines apply.

Don't go in-depth on your solution in this section - you want to present goals and implementation separately.

We propose to move BigCustomer graphs into the new Britannic graph store. In this model, all graph access and modification flows through the Britannic service. Britannic will store both graph deltas and fully constructed versions of BigCustomer graphs. Course deltas and content will also be stored in Britannic, and will be distinct from book graphs. Zowzy client's complex commit-based caching and graph construction logic will be replaced with simple queries for the latest version of a given book graph and course data, and a basic key-value graph id ➜ graph cache.

Tumtum Service will be simplified to support just one Token pair per Account, improving query performance dramatically (Initiative One). Tumtum architecture will be further simplified by centralizing Tumtum requests to Proxy, at the edge of the API, rather than spreading them across Gyre and Gimble farther along (Initiative 2). Finally, we will evaluate the addition of in-memory and/or Redis Access Token caches to further improve token validation performance, if necessary to reach our goals (Milestone 3).

The requirements should make it very clear when the your goal has been achieved.

Bad: "We'll make the /do-something endpoint much faster"
Good: "We'll achieve a < 100ms response time (99th) for the /do-something endpoint"

Talk about the things your proposal won't do. This will prevent scope creep.

Note that Standard Central Config (SCC) does not intend to replace Standard Service Config (SSC). They can and should be used side by side in order to reduce migration time - SCC will be bindable during the Config injection phase in two phase injection at the same time as

ServiceConfig classes.

[...] SCC will not provide a standard way of ensuring that a configuration value stays constant for the lifetime of a client service's request.

Implementation

Describe scope and nature of **your solution**. Start by describing the final state of the world once your proposal has been implemented. **Focus on things that are difficult to change** - typically public interfaces, data schemas, frameworks/other tech choices, or human processes. (Humans are notoriously difficult to refactor.)

Again, people are lazy and don't read, so prioritize information and use pictures when possible. If you are proposing a new schema, provide a schema diagram that highlights the changes. If you're proposing a new interface, provide the method signature or REST documentation.

```
// get contexts with given ids in batch. Returned map will contain an
// entry for all of the input context ids that have associated contexts.
map<commons.id, Context> getContexts(1: required list<commons.id>
contextIds, bool fooEnabled)
```

Primary key lookups on both ContextDefinition and ScopedContextDefinitions followed by realizing contexts. Note that the realized context format is the same for both of these entities.

Then explain step-by-step how you'll get from the current state to the final state. This is sometimes broken out into a separate section titled "Migration Plan". Call out dependencies and teams or services that will be impacted. Readers should get a rough sense of how much work is involved, or the cost of the solution.

Avoid introducing ideas that aren't directly related to the problem you're solving - at best they will invite bikeshedding that distracts from the important issues, at worst they will cause an otherwise good proposal to stall. Make use of existing conventions where possible.

Be opinionated! Don't propose multiple solutions and ask commenters to pick one. The best way to solicit opinions is to make a statement (whether you're confident in it or not) and wait for people to tell you you're wrong. If you considered other solutions, note that and put details about alternate solutions in the appendix.

Signoff & Comments

Prefill the names of anyone whose signoff is required.

Name	+1/-1	Comments
Alfred P. Knewton		

Appendix

This is where you put supporting information that might be relevant some readers but not all. Alternative but rejected implementations and dissenting opinions should also go here.