Architecture Proposal

for

Team Decided - Raft Consensus Library

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

2. Architectural Goals and Philosophy

The most important goal of this project is increasing the reliability of a User Application Service; hence all the high priority non-functional requirements link directly to this and it's also the primary architectural design goal. By directly designing for reliability we incorporate it foundationally into our final product; part of this reliability focused design is a large amount of time initially invested into accurately and exhaustively planning out the project to meet all the functional and non-functional requirements.

An underlying requirement of reliability at the code level is rigorous full suite testing and a focus of strong adherence to code quality/style guidelines. This reduces errors, as well as improves maintainability.

Key to the implementation of reliability is usability, therefore there is also a focus on the design of the usability of the library. To reinforce usability for the product, there will also be complete documentation available to ease integration.

The final library needs to maintain functionality in a system with up to a minority of its nodes failed, and will be put into potentially hostile environments in terms up frequently failing nodes.

Joshua and Sean 07-04-2018

3. Assumptions and Dependencies

Assumptions:

- Market need for consensus library, specifically in .NET
- Performance and latency overhead of consensus is manageable without negatively impacting customer experience
- Requirement for security and the impact of that overhead is acceptable
- The limitation of compatibility in cross-platform implementation for mobile is limited to Xamarin is acceptable
- Dynamic cluster membership, including increase and decreases cluster size/scalability is an important feature to the developers
- Designing this library as free and open source software will not detrimentally inhibit company adoption and that auditability is important
- That developers would utilise the debug logging features of the library to diagnose their issues
- That developers would report back any bugs or issues with library back so they may be resolved

Dependencies:

- Developer implementing the programming required to read/write User Application Service state information into the consensus log
- Developer is able to start a User Application Service off of solely the information contained within the consensus log
- Developer implementing or having load balancing logic in any User Application Client accessing the active User Application Service. This may simply include failover at the IP level between the Cluster of Application Servers.
- Developer will either manually specify node networking addresses or enable a lobby system in their own User Application Service to supply them as required to the library for the starting of the cluster

4. Architecturally Significant Requirements

- The system must be able to communicate over a network
- The system must be able to maintain consensus between distributed network nodes
- The system must be able to maintain a consistent replicated log
- The system must be able to survive failures up to a minority of it’s nodes
- The system must be as easy as possible to implement into existing projects
- The system must be as reliable as possible

5. Decisions, Constraints, and Justifications

Decision	Justification
C# language using .NET framework	C# is the second most popular language of 2017, so using it will make our library available to a wide range of developers
C# over Java	The more mature library integration framework of C# (Nuget) over Java outweighs the 5 percentage points of difference in their popularity
.NET Standard	.NET Standard is a subset of functions in the .NET framework which is. Although this makes development more difficult due to the reduction of prebuilt libraries
Bitbucket	Developer familiarity and the ability to have repos private during development
Full suite unit testing	As reliability is the most important functional requirement, we want to ensure not only that the code is working as intended, but also that changes to the code impact functionality
Style guide	Part of releasing open source code while easing development and maintenance of it by disparate developers is ensuring strict adherence to coding standards
Code review	With reliability such an important non-functional requirement, systematic independent party examination of code is put in place to find bugs as early as possible, as well as maintain adherence to style guide.
Documentation	To enhance usability, thorough cleary and consistent documentation
Implementing own network library	The existing functionality did not exist in an available library, or existing libraries were excessively bloated. With networking foundation to the node communication, controlling it was a practical decision.
Test driven development	As we've focused on exhaughtive design for reliability reasons, TDD is a complementary fit due to its ability to ensure that all of the designed requirements are implemented
Visual Studio IDE	Industry standard .NET IDE, developer familiarity, ease of unit testing
Security	Always-on security was decided due to the nodes being designed to cluster across the public internet, as well as avoiding common issue of leakage or exploits such as the recent etcd and memcached.

Constraint	Justification
Consistent public API	Maintain ease of implementation for developers, regardless of underlying implementation changes
Minimalistic approach	When deciding on additional features or functionality, careful consideration must be taken to ensure a minimalist design is adhered to such as that the implementation of the software maintains being as simple as possible. This is due to reliability reasons, by aiming to avoid potential issues caused during implementation by over complicated or extensive options for developers.
Usability first approach	When deciding on approaches which the developers interacts with (such as the API), the most important constraint is usability and must be considered over unessential functionality. This usability first approach increases reliability, which has been justified multiple times above.
UDP	Although the guaranteed ordering and acknowledgment of packets in TCP would be ideal for a consensus algorithm, the additional round trip times required for this increase latency overhead unreasonability, and as the consensus algorithm takes cases of consistency and packet loss, UDP was chosen as the network protocol so we did not need to incur this overhead.
Universally standard data structure for distributed log	A key-value store (also known as “dictionary” in .NET) was chosen for it’s speed, reliability and non-complex implementation for developers.

6. Architectural Mechanisms

Architectural Mechanism 1 - Distributed Consistent Log

This is the component used by the User Application Service (UAS) to commit their running service's data into a distributed log amongst the consensus nodes. This is the foundational feature which allows other UASs to start up in the event of a running UAS failure.

Architectural Mechanism 2 - Fault Tolerance

This is the features which allows an increase in availability of a given service, it does this though enabling the failing over of a User Application Service to another available node in the cluster.

Architectural Mechanism 3 - Network Communication

This is the functionality which allows the distributed consensus nodes to communicate with each other. It will based on the "fire-and-forget"/"connectionless" UDP protocol to reduce latency, while leaving the overhead of handling packet loss to the consensus algorithm.

Architectural Mechanism 4 - Security

This feature provides identification through public key cryptography, confidentiality through encryption, and integrity through HMAC, for all network communications.

Architectural Mechanism 5 - API Based Integration

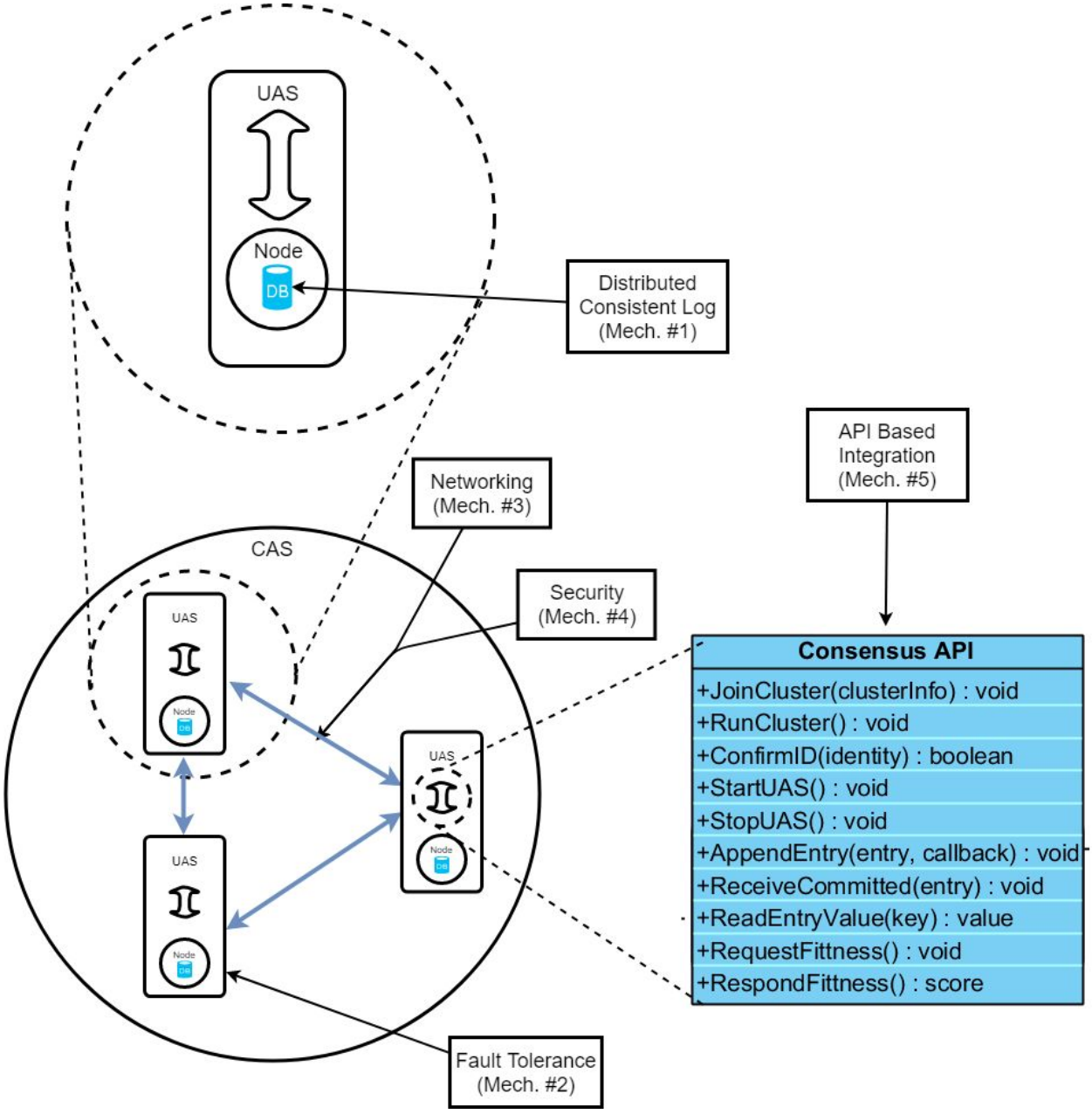
The User Application Server (UAS) communicates with it's consensus node through the use of a .NET class library. This single interface focused on usability is how the UAS communicates to the consensus algorithm.

Architectural Mechanism 6 - High Quality Code

There will be a dedicated and focused effort on ensuring the highest possible quality of code as part of this project. As this code is to be ideally used in ensuring UAS high availability, it's focus on quality must be paramount.

7. Layers or Architectural Framework

From the below diagram, it's shown where each Architectural Mechanism exists in the design. These architectural mechanisms in the diagram are numbered to reflect the assigned numbers previously in Architectural Mechanisms above.



Where each NFRs is addressed in the architectural framework

NFR	**Related Architectural Framework Mechanism**
Reliability	Fault tolerance
Usability	API Integration
Documentation	API Integration, however handled externally to the system
Quality	High Quality Code
Performance	N/A, this is handled at the algorithm and code level
Compatibility	N/A, this is handled by choice of code language
Availability	Networking
Scalability	Distributed Consistent Log, handled in consensus algorithm
Security	Networking
Privacy	Networking
Testability	High Quality Code
Extendability	High Quality Code
Auditability	High Quality Code
Troubleshooting	High Quality Code

8. Architectural Views

Use case view

Please see Use Case Diagram

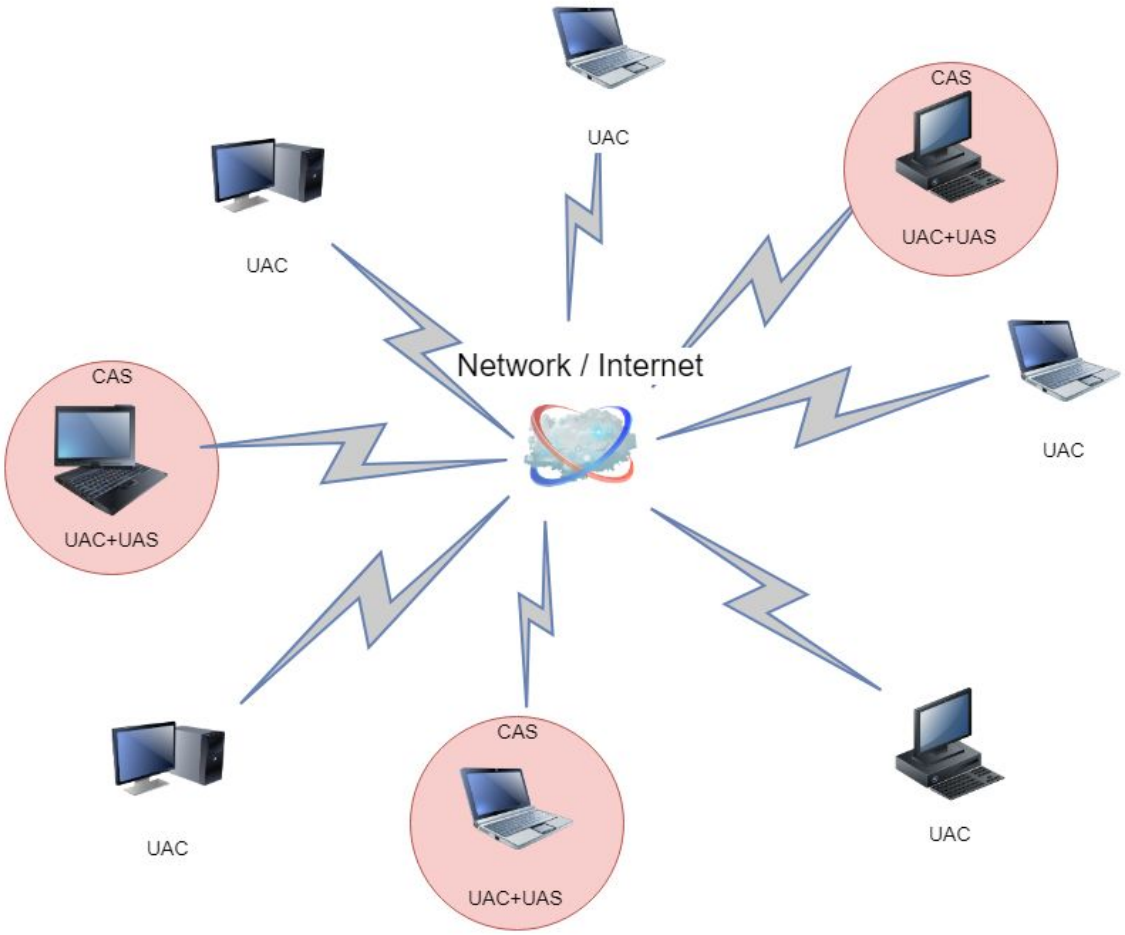
Logical view

Please see Domain Model

Physical view

The consensus library will be integrated into the User Application Service, and will be used to logically tie multiple UASs together into a Cluster of Application Services (CAS).

In the example case of a distributed network where UASs are offloaded from centralised company servers onto UACs there is only a logical difference between UAC and UAS, not physical.



In the example case of dedicated linked servers which provide services for clients, there are multiple distinct servers which run the UASs forming a Cluster of Application Services (CAS). The User Application Clients communicate with the CAS through simple IP failover style.