Phase 1 - Design decisions

In this document the design and implementation of Agent-Smith, a distributed storage network, based on the pithos architecture, will be discussed.

Phase goal

The goal of phase-1, was to design and implement the bare-bones system, with the following components/features:

- Dependency management
- Skeleton Structure
- Externalized configuration
- Bootstrapping of nodes
- Local storage
- Distributed hash table (DHT) / Overlay storage

Component/Feature implementations

The following sections will discuss each component/feature in more detail

Dependency management

For dependency management `Maven` will be used.

Pro's of `maven` for dependency management:

- Automatically retrieves required libraries from a central repository
- Trusted dependency source
- Retrieves source code on request
- Updating/changing dependency versions
- etc ... stack overflow post with examples

Skeleton Structure

`Spring-boot` (webflux) was used to reduce development, testing and integration time. Spring-boot provides most of the boilerplate code and tedious project configurations which allows one to focus on the task at hand, instead of spending time manually configuring a project. Spring also provides an embedded tcp server, netty which will be used in phase-2. Another big advantage of Spring is that it easily integrates with Maven.

Externalised configuration

The power of `Spring` was used to enable externalized configuration properties and automatic loading and binding of configuration classes. This saves time by not having to write boilerplate code.

Bootstrapping of nodes

`Pithos` requires a bootstrap mechanism for new peers that wish to join the network.

In order to enable new peers to join the `Smith network` a bootstrap mechanism is required that will have knowledge of which groups have peer-positions available and what their super-peer addresses are.

To enable bootstrapping of nodes, a few options and approaches were considered.

1. A static, region specific bootstrap server, that handles http requests of peers that wish to join a group and allows super-peers to broadcast their availability.
2. `Apache ZooKeeper`, a hierarchal key-value store, that will have a complete overview of all groups - peers, super-peers, group addresses (super-peer IP).

After brief investigation and a proof-of-concept. `Apache ZooKeeper` was chosen as a bootstrap mechanism. This allows the bootstrap logic to be implemented client side and diminishes the need for a separate bootstrap server code base. `ZooKeeper` a mature project that is highly scalable and easy to configure, making it a favorable.

A `ZooKeeper` instance is still required to run alongside the `Agent-smith` application, which is provided in a `docker-compose` file

```
ZooKeeper is a centralized service for maintaining configuration information,
naming, providing distributed synchronization, and providing group services. All
of these kinds of services are used in some form or another by distributed
applications. Each time they are implemented there is a lot of work that goes into
fixing the bugs and race conditions that are inevitable. Because of the difficulty
of implementing these kinds of services, applications initially usually skimp on
them, which make them brittle in the presence of change and difficult to manage.
Even when done correctly, different implementations of these services lead to
management complexity when the applications are deployed.
```

Additional POC

- Leader election

Local storage

`Pithos` requires a local storage mechanism to enable local object storage. This provides a highly responsive object `state-management` mechanism.

To achieve this goal within `Agent-Smith`, we require a local storage component. For the purpose of allowing the project to be as extendable as possible two local storage implementations were created.

1. `H2 in-memory database` - is a relational database management system written in Java. It can be embedded in Java applications or run in client-server mode.
2. `MongoDB` - MongoDB stores data in JSON-like documents that can vary in structure, offering a dynamic, flexible schema. MongoDB was also designed for high availability and scalability, with built-in replication and auto-sharding.

The `H2 database`, is an in-memory storage implementation, which will allow the application to function as normal without the need for an external database. This is however aimed to be used for development purposes only.

The `MongoDB` is a client-server storage implementation, which gives the client the possibility to save entire POJOs as objects within the DB, without configuring the database manually.

A `MongoDB` instance is still required to run alongside the `Agent-smith` application, which is provided in a `docker-compose` file

Distributed hash table (DHT) / Overlay storage

`Pithos` requires a DHT storage mechanism, to act as an overlay storage. This provides reliable object `state-persistency` in the storage network.

To achieve this goal within `Agent-Smith`, A high performance key-value pair storage library is required, a few options that were considered,

1. `Kademlia` - a distributed hash table for decentralized peer-to-peer computer networks
2. `FreePastry` - an open-source implementation of Pastry intended for deployment in the Internet
3. TomP2P - a java-based, P2P-based high performance key-value pair storage library

After some investigation and a proof-of-concept, `TomP2P` was chosen.
`TomP2P` is the only DHT implementation that has a well maintained library, with active, responsive developers. The library is documented and code example of various implementations are available. This made `TomP2P` the only truly viable choice at the time of implementation.