Anwendungsdokumentation

Server

Ein PaenkoDb Cluster besteht aus mehreren Servern, die alle miteinander verbunden sind. Ein Server wird deshalb auch Peer genannt.

Konfigurationsdatei

Die Konfigurationsdatei wird im *toml* Format konfiguriert. Zuerst wird der eigene Server konfiguriert. Dies geschieht im `[server]`-Block.

Jeder Peer benötigt eine eindeutige `node_id`, mit dieser kann der Peer identifiziert werden. Außerdem braucht jeder Peer eine `node_address`. Über diese Adresse kann wird mit den anderen Peers kommuniziert. Der `community_string` ist Security-Feature, das gewährleistet, dass sich nicht unbekannte Peers dem Cluster beitreten. Dieser String muss auf allen Peers gleich sein, damit sie sich gegeneinander verbinden können. Die letzte Einstellung des Servers ist die `binding_addr`. Diese Adresse ist für den Webserver.

```
[server]
node_id = 1
node_address= "0.0.0.0:9000"
community_string="test"
binding_addr="0.0.0.0:3000"
```

Zurzeit unterstützt PaenkoDb nur einen festkonfigurierten Benutzer. Dieser wird über den `security`-Block konfiguriert. Dieser Block besteht aus `username` und `password`. Wobei das Passwort in Sha256 gehashed sein muss.

```
[security]
username = "kper"
password = "a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3"
```

Die Peer-Konfiguration erfolgt über `[[peers]]`. Die doppelten eckigen Klammern werden dann zu einem Array zusammengefasst. Ein Peer besteht aus `node_id` und `node_address`.

```
[[peers]]
node_id = 2
node_address = "0.0.0.0:9001"
```

PaenkoDb kann mehrere Logs parallel betreiben. Ein Log lässt sich über einen `[[logs]]`-Block konfigurieren. Ein Log-Eintrag besteht aus `path` und `lid`. `Path` ist der Pfad zum Verzeichnis, wo die Daten gespeichert werden. `lid` ist eine eindeutige Identifikationsnummer, welche dem `Uuid`-Version 4 Standard entsprechen muss. Diese wird auch LogId genannt und muss auf allen Nodes gleich sein!

```
[[logs]]
path = "node0_0"
lid = "3d30aa56-98b2-4891-aec5-847cee6e1703"

[[logs]]
path ="node0_1"
lid ="2275a90d-5884-4498-9ba5-91122e2fe875"
```

Möchte man nun einen Server zur Laufzeit hinzufügen, muss statt die Peers über `[[peers]]` konfigurieren, einen `[dynamic_peer]` in der Konfiguration konfigurieren. Dieser Dynamic Peer ist der Ansprechpartner um dem Cluster beizutreten und dieser gibt seine Peers an den Anwärter weiter. Der `dynamic_peer`-Block besteht aus `node_id` und `node_address`. Wenn man einen `[dynamic_peer]`-Block in seiner Konfiguration hat, benötigt man keine `[[peers]]` mehr, da diese vom DynamicPeer übernommen werden.

```
# Aus Sicht von node_id = 3 der dritte Peer
[dynamic_peer]
node_id= 1
node_address = "leader:9000"
```

Um dann den Server zu starten:

```
document server config.toml
```

`document` entspricht dabei die Binary und `config.toml` ist der Pfad zur Konfigurationsdatei.

Docker

Neben dem gewöhnlichen Server-Start lässt sich PaenkoDb auch über Docker starten. Dazu muss lediglich nachdem das GitHub-Repository heruntergeladen wurde und in den `docker`-Ordner gewechselt werden. Durch den Befehl `docker-compose up` werden automatisch zwei Nodes gestartet. Die Raft-Kommunikation findet dabei auf den Ports `9000` und `9001` statt und die Server sind über die Ports `3000` und `3001` per Webbrowser oder Rest-API erreichbar.

```
git clone https://github.com/paenko/paenkodb
cd paenkodb/docker
docker-compose up
```

REST

Parameter für die Restrouten:

:lid	LogId des Logs	Uuid Version 4
:id	Id des Dokumentes	Uuid Version 4

:session	Id der Transaktion	Uuid Version 4

Login

GET	/auth/login	Zeigt den derzeit eingeloggten Nutzer an
POST	/auth/login	Loggt Nutzer ein

Payload

```
{
    "username": String,
    "password": String (Plaintext)
}
```

POST	/auth/logout	Loggt Nutzer aus

Dokumente

GET	/document/:lid/:id	Abfrage eines Dokumentes
POST	/document/:lid	Fügt ein neues Dokument ein

Payload

```
{
    "payload": BASE64, "version": Number
    "id": Uuid
}
```

Anmerkung: `id` wird nicht übernommen.

POST	/document/:lid/transaction/:session	Fügt neues Dokument ein (Transaktion)

Payload

```
{
    "payload": BASE64,
    "version": Number
    "id": Uuid
}
```

Anmerkung: `id` wird nicht übernommen.

DELETE	/document/:lid/:id	Löscht Dokument
DELETE	/document/:lid/:id/transaction/:session	Löscht Dokument (Transaktion)
PUT	/document/:lid/document/:id	Manipuliert Dokument

Payload

```
{
    "payload": BASE64
}
```

PUT	/document/:lid/transaction/:session/:id	Manipuliert Dokument (Transaktion)

Payload

```
{
    "payload": BASE64
}
```

Transaktionen

POST	/transaction/begin/:lid	Startet eine neue Transaktion
POST	/transaction/commit/:lid/:session	Committed eine Transaktion
POST	/transaction/rollback/:lid/:session	Rollt eine Transaktion zurück

Meta

GET	/meta/log/:lid/documents	Zeigt alle Ids in einem Log an
GET	/meta/:lid/state/leader	Zeigt die Sicht des Leaderstates
GET	/meta/:lid/state/candidate	Zeit die Sicht des Candidatestates
GET	/meta/:lid/state/follower	Zeigt die Sicht des Followerstates
GET	/meta/peers	Zeigt alle Peers an

Cli - Client

doc-id	Id des Dokuments
lid	Id des Logs
node-address	Adresse des Servers
username	Username

password	Passwort (gehashed in Sha256)
filepath	Pfad zur Datei, die entweder hochgeladen oder geupdated werden soll
transid	Id der Transaktion

```
document get <doc-id> <lid> <node-address> <username> <password>
document put <doc-id> <lid> <node-address> <filepath> <username> <password>
document post <lid> <node-address> <filepath> <username> <password>
document remove <doc-id> <lid> <node-address> <username> <password>
document server  <config-path>
document begintrans <lid> <node-address> <username> <password>
document commit <lid> <node-address> <username> <password> <transid>
document rollback <lid> <node-address> <username> <password> <transid>
document transpost <lid> <node-address> <filepath> <username> <password> <transid>
document transremove <lid> <node-address> <doc-id> <username> <password> <transid>
document transput <lid> <node-address> <doc-id> <filepath> <username> <password> <transid>
```

C-Sharp Library

UuidManager

Uuids können in der C-Sharp Library mithilfe des UuidManagers verwaltet werden. Der UuidManager stellt folgende Funktionen zur Verfügung:

SaveIds(string file)	Speichert alle Uuids die hinzugefügt wurden
Load(string file)	Lädt gespeicherte Uuid Objekte
Add(string id, string description, UuidObject.UuidType type)	Fügt eine Uuid mit Beschreibung und Typ hinzu
LookForId(string description)	Gibt eine Uuid mit entsprechender Beschreibung zurück
LookForDescription(string id)	Gibt eine Beschreibung mit entsprechender Uuid zurück
LookAll(UuidObject.UuidType type = UuidObject.UuidType.All)	Gibt alle hinzugefügten Uuids mit entsprechenden Typen zurück

Als Typ kann entweder Key oder Log angegeben werden. Ein Key ist die Uuid eines Files und ein Log ist eine Logid.

Node

Die C-Sharp Library kann wie im folgendem Beispiel verwendet werden, das Beispiel ist ein durchgängiges Programm und wird in einzelnen Abschnitten erklärt:

```
Node node = new Node(IPAddress.Parse("207.154.216.94"), 3000);
node.Login(new Authentication("ich", "pw"));
UuidManager.Load("ids.json");
```

Ein Node Objekt symbolisiert einen Datenbankserver. Über den Konstruktor des Node Objekts wird eine Verbindung zum jeweiligen Server hergestellt. Mit der Methode Login der Node Klasse wird eine Session mit dem Datenbankserver eröffnet. Diese Session wird benötigt um die Authentifikationsdaten zu behalten, sie wird am ende des Beispiels mit Logout wieder beendet. Hier werden über den UuidManager alle Uuids aus dem File ids.json geladen.

```
var logs = node.GetLogs();

if (UuidManager.LookForId("HauptLog") == null)
{
UuidManager.Add(logs[0], "HauptLog", UuidObject.UuidType.Log);
}
```

Die GetLogs Methode der Node Klasse gibt alle Logs am Server zurück. Mit UuidManager.LookForId wird überprüft, ob eine Uuid mit der Beschreibung HauptLog vorhanden ist, falls dies nicht der Fall ist, wird der erste Log des Servers als HauptLog im UuidManager hinzugefügt.

```
Bankdaten tk = new Bankdaten();
Document x = Document.FromObject<Bankdaten>(tk);

if (UuidManager.LookForId("Bankdaten") == null)
{
node.PostDocument(x, UuidManager.LookForId("HauptLog"), "Bankdaten");
}
```

Bankdaten kann eine beliebige Klasse sein, in diesem Fall hat sie zwei Properties (Konto1 und Konto2). Es wird ein Objekt dieser Klasse erstellt und der statischen Methode Document.FromObject übergeben, diese Methode erstellt ein Dokument das an einen Datenbankserver geschickt werden kann. Die Payload dieses Dokuments entspricht einer Serialisierung des übergebenen Objekts. Es wird mithilfe des UuidManagers überprüft ob bereits eine Uuid für die Beschreibung Bankdaten existiert, falls nicht, werden die Bankdaten an den Datenbankserver gepostet. Es wird Dokument x an den Log mit der Beschreibung Hauptlog gesendet. Hierbei ist zu beachten das während des postens ein neuer Eintrag im UuidManager erstellt wird.

```
var response = node.
```

```
    GetDocument(
        UuidManager.LookForId("HauptLog"),
        UuidManager.LookForId("Bankdaten"));

var bankdaten = response.ToObject<Bankdaten>();
Console.WriteLine($"a: {bankdaten.Konto1} b: {bankdaten.Konto2}");
UuidManager.SaveIds("ids.json");
node.Logout()
```

Um ein Dokument vom Datenbankserver herunterzuladen wird GetDocument verwendet. Diese Methode benötigt einen Parameter für das Dokument das erhalten werden möchte und den Log in dem sich dieses Dokument befindet. Es wird hier das Dokument Bankdaten vom Log HauptLog angefordert. Dieses Document wird mit der Methode ToObject der Klasse Document deserialisiert und kann nun wieder als Objekt vom Typ Bankdaten verwendet werden. Am ende speichern wir alle Uuids im Manager ab und beenden die Session.

Alle Methoden die CRUD Operationen durchführen können synchron oder asynchron aufgerufen werden.

Document

Library Methoden die eine document Route ansprechen geben ein Objekt mit dem Typ Document zurück oder nehmen ein solches Objekt entgegen. Ein Document enthält die payload eines Dokuments, die Document Klasse besitzt folgende Methoden:

FromStream(Stream docstream)	Erstellt ein Dokument mit dem Inhalt eines Streams als payload
FromObject(T docobj)	Erstellt ein Dokument mit dem Inhalt eines Objekts als payload
ToObject()	Erst ein Objekt mit dem Inhalt der payload eines Documents

C-Sharp Manager

Auf Youtube und der Paenko Facebook Seite existiert ein Video welches die Verwendung des GUI Progamms erklärt.