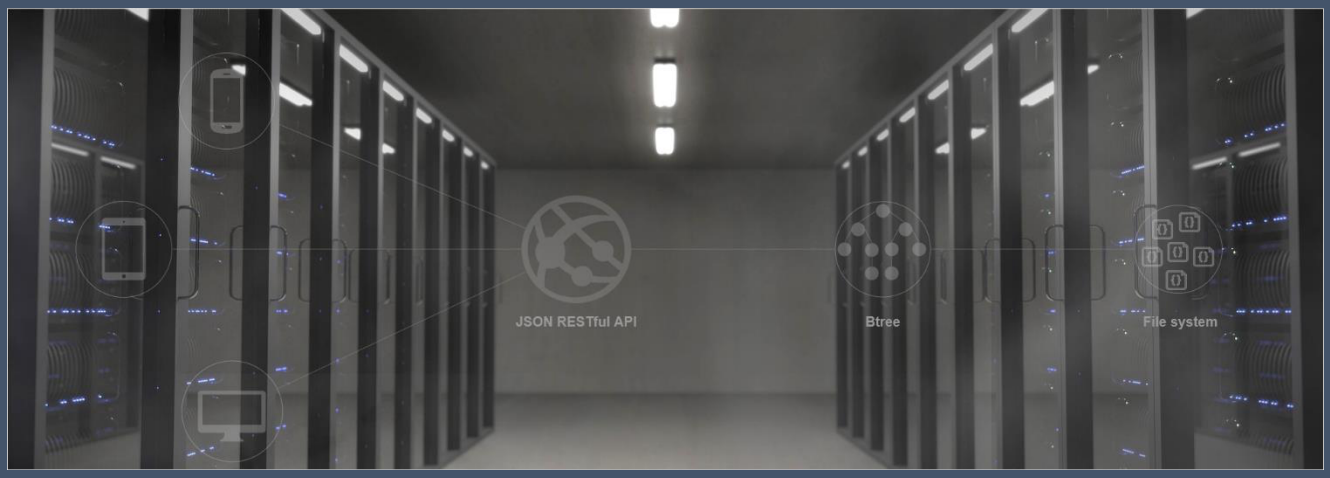
HBU
Höhere Berufsbildung Uster
Höhere Fachschule Uster

Diplomarbeit

KVM Store Server

Key - Value-Map Store Server



Eine Diplomarbeit von Theodor Bogdanovic
für die Höhere Fachschule Uster

HF Informatik NDS

5. März 2018

11 Anleitungen

In den nachfolgenden beiden Unterabschnitten sind sämtliche Informationen enthalten, um den kvmstoreserver zu installieren und zu betreiben.

11.1 Installationsanleitung

Diese Anleitung geht von einem bereits installierten Ubuntu Linux 16.04.x Server aus. Sollte dies nicht der Fall sein, holen Sie sich bitte die notwendige technische Unterstützung, um die Installation durchzuführen. Alternativ zu Testzwecken, kann auch eine Desktopversion genutzt werden. Hierzu können Sie einen der beiden nachfolgenden Links nutzen:

- Deutsch: https://wiki.ubuntuusers.de/Ubuntu_Installation/
- English: https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop

11.1.1 Installation

Die Installation wird durch ein Shell Script realisiert, welches *whiptail* (Fensterdarstellung in der Konsole) als Frontend nutzt. Die Installation wurde so einfach wie möglich gehalten und benötigt lediglich drei User Eingaben, um *kvmstoreserver* zu installieren. An dieser Stelle sollen nur die Schritte beschrieben werden, die notwendig sind, um den Installer anzustossen.

Schritt 1/5:

Kopieren Sie das gezippte TAR Archiv *kvmss-installer.tar.gz* in ein Verzeichnis Ihrer Wahl innerhalb Ihres Linux Accounts.

Schritt 2/5:

Starten Sie eine Konsole. Wechseln Sie in das Verzeichnis, in welchem sich das TAR Archiv befindet. Entpacken Sie den Installer mit nachfolgender Konsolenanweisung:

```
tar -xvf kvmss-installer.tar.gz
```

Schritt 3/5:

Beim Entpacken wurde das Verzeichnis *kvmstoreserver_installer* erzeugt. Wechseln Sie nun in dieses Verzeichnis mit der Konsolenanweisung:

```
cd kvmstoreserver_installer
```

Schritt 4/5:

Führen Sie nachfolgende Konsolenanweisung aus. Diese startet den Installer. Sie benötigen ROOT Rechte für die Ausführung des Installers «**sudo**».

```
sudo ./kvmss-installer.sh
```

Schritt 5/5:

Folgen Sie den Anweisungen des grafischen Installers. Nach seiner Beendigung haben Sie eine bereits laufende Instanz des Key – Value-Map Store Servers.

Wenn Sie möchten, können Sie das Verzeichnis *kvmstoreserver_installer* und das TAR Archiv kvmss-*installer.tar.gz* löschen. Es wird empfohlen, es beizubehalten, da im Verzeichnis der Deinstaller integriert ist.

11.1.2 Deinstallieren

Haben Sie das Verzeichnis und das TAR Archiv bereits gelöscht, führen Sie bitte alle Schritte wie unter Punkt 11.1.1 beschrieben, nochmals aus. Ansonsten können Sie bei Schritt 3/5 einsteigen.

Die Deinstallation funktioniert gleich wie die Installation, da der Deinstaller im Installer integriert ist. Somit werden Sie wieder zum grafischen Frontend geleitet. Folgen Sie auch hier den Anweisungen des Tools. Sie müssen hier hauptsächlich nur noch bestätigen.

Der Deinstaller entfernt ALLE Komponenten und **Daten** von der Harddisk. Er übernimmt auch die Deregistrierung des Service beim Systemd Init Daemon.

11.2 Benutzeranleitung / API Referenz

Dieser Abschnitt unterteilt sich in zwei Unterabschnitte. Im ersten wird die Nutzung der RESTful JSON Schnittstelle behandelt und im zweiten die Konfigurationsmöglichkeiten des *kvmstoreserver*.

11.2.1 API Referenz

In diesem Abschnitt werden wir die Möglichkeiten betrachten, wie wir Daten in den *kvmstoreserver* hochladen, auf sie wieder zugreifen und sie bei Bedarf wieder entfernen können. Hierzu soll nachfolgendes JSON Dokument als Basisbeispiel für sämtliche Demonstrationen dienen.

```
{
    "system_name": "Workflow-Server-1",
    "system_number": 1,
    "system_version": 1.2,
    "system_active": true,
    "system_null": null
}
```

Grundsätzlich wurde der kvmstoreserver so entwickelt, dass jedes valide JSON Dokument gespeichert werden kann. Hätte das Beispiel oben eine verschachtelte Innenstruktur, welche unter einem eigenen Namen hinterlegt wäre, so wäre es für den kvmstoreserver weiterhin valid. Man könnte aber nur den Innenstrukturnamen adressieren und hätte als Antwort ein ganzes JSON Dokument. Dies ist nicht der primäre Einsatzzweck dieses Produktes. Er soll lediglich eine Mappe an Werten an einen Key binden können. Das Beispiel von oben ist hierfür ideal, da es sämtliche validen Datentypen abdeckt, die in einem JSON Dokument, aber auch im kvmstoreserver, zulässig sind.

Für die nachfolgenden Beispiele benötigen Sie einen Rest Client. Sie können hier den Client Ihrer Wahl verwenden. Achten Sie lediglich darauf, dass sein Request Header auf **Content-Type : application/json** gestellt ist. Sollte Ihr Client nicht das automatische Erkennen von Response Types unterstützen, so setzen Sie die Einstellungen für den Response Header auf **Accept : application/json**.

11.2.1.1 Eintrag erzeugen

Wir wollen das obige JSON Beispiel unter dem Key Name **work-srv-1.conf** hinterlegen. Kopieren Sie das Beispiel und fügen Sie es in Ihren Client als Content hinzu. Geben Sie als URL folgendes ein:

```
http://<Domönen Namen oder IP>:3344/add/work-srv.conf
```

Dies kopiert Ihr JSON Dokument in den kvmstoreserver unter der Revisionsnummer 0. Als Antwort erhalten Sie ein JSON Dokument, welches den Response Status 201 (Creadet) als Feld enthält, eine kurze Operationsbeschreibung sowie einen Text, welcher auf die aktuelle Revisionsnummer 0 verweist. Sollte etwas schiefgelaufen sein, wiederholen Sie die Schritte. Ursache des Fehlers könnte ein Schreibfehler sein.

Möchten Sie eine Korrektur nachspeichern, wie bspw. das JSON Feld **system_active** auf **false** setzen, dann ändern Sie jetzt den Value im JSON Dokument und rufen Sie nochmals den */add/* Controller, wie oben beschrieben, auf. Nun haben Sie die erste Version als Revision 0 und die Korrektur als Revision 1 gespeichert.

11.2.1.2 Einträge abrufen

Möchten Sie das soeben eingefügte Dokument komplett einsehen, bzw. die aktuellste Revisionsnummer 1, dann geben Sie nachfolgendes in Ihre Client URL-Adressleiste ein:

```
http://<Domönen Namen order IP>:3344/get/work-srv.conf
```

Dies liefert Ihnen das gesamte Dokument zurück, so wie Sie es hochgeladen haben.

Möchten Sie die erste Version des gesamten Dokuments, welche unter der Revisionsnummer 0 hinterlegt ist, einsehen, müssen Sie folgenden Aufruf durchführen:

```
http://<Domönen Namen order IP>:3344/get/work-srv.conf?rev=0
```

Nun erhalten Sie die erste Version des Dokuments, in welcher das Feld *system_avtive* noch auf *true* steht.

Möchten Sie nur den Value des JSON Feldes *system_active* zurückgeliefert bekommen, müssen Sie den *search* Parameter benutzen:

```
http://<Domönen Namen order IP>:3344/get/work-srv.conf?search=system_active
```

Nun erhalten Sie einen Response zurück, in welchem der Body nur den JSON Type *false* enthält. Dieser Case ist der am häufigsten verwendete in der Praxis, denn in der Regel möchte man lediglich einen Konfigurationsparameter für ein Endsystem abrufen und nicht erst mit viel Aufwand den Response Body parsen.

Nun möchten wir wissen, welchen Wert der Konfigurationsparameter *system_active* vor der Korrektur hatte. Dieser liegt in der Revision 0.

```
http://<Domönen Namen order IP>:3344/get/work-srv.conf?rev=0&search=system_active
```

Nun sollten wir als Ergebnis einen JSON Type *true* zurückerhalten, denn diesen Wert hatte das Feld bei der ersten Erstellung.

Die Reihenfolge der Parameter ist übrigens egal. *Search* hätte auch als erster Query Parameter deklariert werden können. Lediglich die Parameter – Initialisierung «?» und die Deklaration der Nachfolgeparameter «&» ist von der Reihenfolge her relevant.

11.2.1.3 Eintrag entfernen

Nehmen wir an, dass unsere Konfiguration *work-srv.conf* nicht mehr benötigt wird. In diesem Fall können wir sie vom kvmstoreserver löschen. Dazu muss erstens einmal der */del/* Controller angesprochen werden und zweitens ein spezifisches JSON Dokument für den Löschauftrag erstellt und mitgegeben werden.

Eine solche Löschanforderung könnte dabei wie folgt aussehen:

```
http://<Domönen Namen order IP>:3344/del/work-srv.conf
```

Dieser JSON Body muss noch mitgegeben werden:

```
{
    "key_name": "work-srv.conf",
    "confirm_delete": true
}
```

Grundsätzlich muss jede Löschanweisung den URL Key Namen im JSON Feld *key_name* tragen. Dies dient der Bestätigung seitens User oder Applikation, ob der Key grundsätzlich der richtige ist. Als Sicherheitsmassnahme ist auch jede Löschanfrage mit *confirm_delete* zu bestätigen.

11.2.1.4 Allgemeine Informationen

Grundsätzlich sind auch Key Namen mit Leerschlag erlaubt. Auch ein breites Set an Sonderzeichen wird unterstützt. Dennoch wird empfohlen keine solchen Elemente im Key Namen zu verwenden. Als allgemeine Richtlinie könnte gelten, was nicht in einem Windows Dateinamen erlaubt ist, sollte auch nicht im Key Namen vorkommen.

11.2.1.5 Den Binärbaum einsehen

Möchten Sie eine vom Binärbaum in einen Slice gemappte Ansicht des Baumes sehen, so rufen Sie folgendes auf:

```
http://<Domönen Namen order IP>:3344/tree
```

11.2.1.6 Versionsinformationen

Für allgemeine Informationen über den kvmstoreserver, können Sie nachfolgendes eingeben. Als kleiner Tipp, wenn Sie die Verfügbarkeit des kvmstoreserver monitoren möchten, bspw. mit Zabbix, so können Sie diesen Aufruf für periodisches Polling nutzen:

```
http://<Domönen Namen order IP>:3344/version
```

11.2.2 Kvmstoreserver Konfiguration

Der *kvmstoreserver* kann grundsätzlich nur per Konfigurationsdatei konfiguriert werden. Diese liegt nach der Installation im User HOME Verzeichnis des Users *kvmstoreserver*, unter */home/kvmstoreserver/kvmstoreserver/etc/kvmstoreserver.conf*. Möchten Sie hier Änderungen vornehmen, dann loggen Sie sich als User kvmstoreserver mit dem Passwort, welches Sie bei der Installation vergeben haben, ein und wechseln Sie ins oben angegebene Verzeichnis. Hier können Sie mit einem Editor Ihrer Wahl die Konfigurationsdatei öffnen. Sie sehen dabei folgende Parameter, die konfigurierbar sind:

11.2.2.1 storage_target

Hier wird der Pfad angegeben, in welchem der kvmstoreserver die JSON Files abspeichern kann.

11.2.2.2 serde_driver

Hier können Sie entscheiden, welcher Typ Speicher Backend verwendet werden soll. In dieser Version steht nur der Filesystem Driver *filesystem* zur Wahl. Backend Driver für MSSQL und Postgres können in einer nachfolgenden Produktivversion folgen.

11.2.2.3 defualt_log_file

Der Pfad und Standard Name des Logfiles.

11.2.2.4 Log File Parameter

- **Log_max_size_mb**: Definiert wie gross das Log File in Megabyte werden darf.
- **Log_max_age_days**: Definiert wie alt das älteste Log File sein darf, bevor es gelöscht wird.
- **Log_write_to_console**: Schreibt die Applikationslogs parallel ins File und auf die Konsole. Dieser Parameter steht per Default auf false und sollte nur zu Debugging – Zwecken aktiviert werden. Das Schrieben auf die Konsole bedeutet bei einem Systemd Init System, dass die Logs auch nach /var/log/syslog geschrieben werden. Dies kann syslog mit der Zeit überfüllen.

11.2.2.5 http Parameter

- **http_hostname**: Geben Sie hier den Domänennamen oder die IP ein, auf welche der kvmstoreserver hören soll. Die Eingabe des Installers wurde hier hingeschrieben.
- **http_port**: Der kvmstoreserver hört standardmässig auf Port 3344. Wünschen Sie einen Portwechsel, können Sie hier einen neuen eingeben. Achtung: Ports unter 1024 können nur mit Root – Rechten genutzt werden!
- **read_header_timeout**: Definiert wie lange auf einen hängenden Header (korrupter Header) gewartet werden soll bzw. auf sein Ende. Angabe in Sekunden. Der Default Wert ist optimal und sollte nicht geändert werden.
- **read_timeout**: Definiert wie lange ein Request Read dauern darf. Sollten extrem grosse JSON Dokumente zum Einsatz kommen oder die Netzwerkverbindung zu langsam sein, sollte dieser Wert (in Sekunden) zwingend erhöht werden.
- **write_timeout**: Definiert wie lange der kvmstoreserver in einen Response schreiben darf. Auch hier gilt: Sind die JSON Dokumente sehr gross oder ist die Verbindung sehr träge, zwingend den Wert (in Sekunden) erhöhen.

- **tcp_idle_timeout_millsec**: Dieser Wert definiert, wie lange der kvmstoreserver bzw. der OS Kernel Network Stack, eine geschlossene TCP Verbindung als Socket Tupel aufrecht erhalten soll. Haben Sie eine Umgebung, welche aus vielen Clients besteht, welche in langen Intervallen auf den kvmstoreserver zugreifen, sollte dieser Wert klein gehalten werden (100 Millisekunden gelten als Optimum). Habe Sie hingen wenig Clients, die in kurzen Intervallen wie bspw. alle 20 Sekunden, Zugriffe ausführen, so ist ein Wert von 20 Sekunden bzw. 20'000 Millisekunden optimal.

11.2.2.6 Route Parameter

- **ctrl_add_path**: (Default: /add/); Hier kann der URL Pfad zum Hinzufügen von Dokumenten definiert werden.
- **ctrl_get_path**: (Default: /get/); Hier kann der URL Pfad zum Abrufen von Dokumenten definiert werden.
- **ctrl_del_pat**h: (Default: /del/); Hier kann der URL Pfad zum Löschen von Dokumenten definiert werden.
- **ctrl_tree_path**: (Default: /tree); Hier kann definiert werden, wie der Pfad zum Btree-zu-Slice Mapper lauten soll.
- **ctrl_version_path**: (Default: /version); Hier kann definiert werden, wie der Pfad zur Versionsinfo lauten soll.

Hinweis: Die Pfadlänge der *add*, *get* und *del* Controller ist auf 1 beschränkt. Konfigurationen wie bspw. */controller/add* sind nicht zugelassen, werden aber nicht als Fehler angesehen. Die Funktionsweise des *kvmstoreserver* ist aber nicht mehr gewährleistet. Auf eine Einschränkung wurde bewusst verzichtet, da in einer zukünftigen Produktivversion ein anderer Ansatz verfolgt werden soll. Daher sollen an dieser Stelle keine Code spezifischen Restriktionen eingebaut werden, welche nachträglich wieder entfernt werden müssen. Zusätzlich MUSS jeder der genannten Controller mit einem Backslash («/») enden. Der Grund ist der gleiche wie bei der Pfadlänge.

11.2.2.7 STATIC und DYNAMIC

Diese Version ist mit der *configPoller* Erweiterung kompiliert worden. Dies bedeutet, dass alle Änderungen an der Konfigurationsdatei in einem Intervall von 60 Sekunden (Hard Coded) erkannt werden. Die erkannten Änderungen werden dann zur Laufzeit des kvmstoreserver geladen und sind sofort aktiv. Dies gilt aber nicht für alle Parameter. In der Konfigurationsdatei sind Kommentar – Tags, welche zeigen, ob der Parameter vom Typ STATIC oder Typ DYNAMIC ist. Ist ein Parameter vom Typ STATIC, so werden die Änderungen erst nach einem Neustart der Applikation wirksam. Ist er hingegen vom Typ DYNAMIC, werden sie wirksam, sobald der configPoller einen Prüfintervall anstösst und die Änderung wahrnimmt.

ACHTUNG: Sollte der Parameter fehlerhaft sein, also bspw. ein Buchstabe in einem Integer Wert vorkommen, wird das Package *configMap*, welches für das Einlesen und die Zurverfügungstellung der Konfiguration verantwortlich ist, in eine PANIC laufen. Der kvmstoreserver wird CRASHEN!!!

15 Anhang

Nachfolgend befinden sich die zur Arbeit gehöhrenden Komponenten, Beschreibungen und Informationen, welche aber nicht expliziter Bestandteil der eigentlichen Problemlösung sind.

15.1 inject_client

Diese Komponente wurde im Verlauf der Arbeit entwickelt, um Connection – Belastungstests durchführen zu können. Die Hauptaufgabe besteht hier explizit im Ausführen von Stresstests und nicht im netzwerkseitigem Abfüllen des kvmstoreserver mit Daten. Der inject_clinet ist als Konsolenapplikation entwickelt worden und liegt dem beiliegenden Datenträger bei. Er wurde für Windows, Linux (Cross-Compiling, ungetestet) und MacOS X (Cross-Compiling, ungetestet) kompiliert.

Inject_client wurde hauptsächlich auf der Plattform Windows entwickelt und eingesetzt. Es wird empfohlen, ihn auch auf dieser Plattform einzusetzen. Linux hat hier Schwierigkeiten bei der Erzeugung der hohen Anzahl an Socket – Files (Restriktion auf 1024 Files) und muss dementsprechend zuerst um konfiguriert werden, damit der Client funktioniert. Die Nutzung auf MacOS ist gänzlich ungetestet und die Funktionalität unbekannt.

Inject_clinet versucht, bei jedem Aufruf per Konsole 6000 gleichzeitige Connections zum Target aufzubauen und, beginnend ab Parametereingabewert, JSON Dokumente nach einem statischen Pattern in den kvmstoreserver abzulegen. Die Felder des Dokumentes beinhalten dabei die Datentypen String, Integer (Go intern Int64), Float64, Boolean und JSON NULL (Go nil). Die Werte basieren alle auf der Laufnummer (Integer), welche inkrementell ab Startpunkt erhöht wird.

Die Restriktion auf 6000 Connections bzw. parallel ausgeführte goroutinen, resultiert aus der Tatsache, dass ein Windows System ca. 16000 (+ ein kleiner Überschuss) TCP Sockets öffnen kann (mehr ist möglich nach Manipulation der Registry). Go bietet keine Möglichkeit, um den http Default Client (Package) anzuweisen, die Connections zu schliessen. Windows hält die geöffneten Sockets im State *wait_close* ca. 4 Minuten lang, was bedeutet, dass in dieser Zeit keine weiteren Connections erlaubt sind, wenn der Range aufgebraucht ist. Aus diesem Grund werden lediglich 6000 Connections pro Aufruf getätigt. So können zwei Durchläufe nacheinander getätigt werden, um eine Durchschnittszeit zu erhalten.

Der inject_client liefert nach jedem erfolgreichen Durchlauf seine Gesamtlaufzeit sowie einen Print des letzten Eintrages zwecks Überprüfung. Ein möglicher Aufruf könnte dabei wie folgt aussehen:

```
Inject_client -target http://<Domönen Namen order IP>:3344 -start 10000
```

Der Parameterstart ist optional. Wird er nicht mitgegeben, so startet der Client immer bei 0 und geht bis 5999. Sollen mehrere Revisionen über ein Set von Keys gelegt werden, muss der Client immer mit der gleichen Startposition aufgerufen werden.

Hinweis: Sollte es zu Fehlern in goroutinen kommen, werden diese stets auf den aufrufenden Standard Output der Konsole geschrieben. Mit Ctrl + c kann das Tool gestoppt werden.

15.2 inject_disk

Das Tool inject_disk wurde zwecks kvmstoreserver Loader Testing entwickelt. Der einzige Sinn dieses Tools besteht darin, eine definierte Zahl an Files zu generieren, welche einen JSON Dokumentinhalt besitzen. Das Design des JSON Dokumentes ist dabei eine Kopie des im inject_client eingesetzten Patterns.

Möchte man den kvmstoreserver testen und hat keine Lust, manuell Daten einzuspielen oder gar einen Testclient zu schreiben, kann dieses Kommandozeilen – Tool verwendet werden. Da keine Obergrenze für die Anzahl an zu erzeugenden JSON Objekten definiert ist, kann auch eine grosse Anzahl erzeugt werden. Dies ist für Tests nahe den Realbedingungen ideal geeignet und kann helfen, die Skalierung der Maximalzahl an Dokumenten, welche unter Abschnitt 9.7.6 beschrieben wird, besser zu bestimmen.

Ein möglicher Aufruf von inject_disk könnte wie folgt aussehen:

```
Inject_disk -target /home/kvmstoreserver/kvmstoreserver/warehouse -objects 200'000
```

Inject_disk wird durch den Installer mitinstalliert und liegt unter */home/kvmstoreserver/kvmstoreserver/utils/inject_disk*. Wird er vor Ort oder innerhalb des Home – Verzeichnisses des Users kvmstoreserver (mit seinen Rechen) ausgeführt, kann auf die Angabe des Parameters *-target* verzichtet werden.

Dieser Client liegt ebenfalls in Versionen für Windows, Linux und MacOs X vor.

15.3 Godoc mit http Repräsentation

Möchten Sie die Source Code Dokumentation in http Form sehen, können Sie *godoc* benutzen. Hierfür muss zuerst die Go Compiler Collection herunterladen und installieren werden:

https://golang.org/dl/

Nach der Installation müssen Sie eine Eingabeaufforderung oder ein Terminal öffnen und folgendes eingeben:

(Windows) → set GOPATH=<Pfad zum Datenträger>/raw_source_code

(Linux / MacOS) → export GOPATH=<Pfad zum Datenträger>/raw_source_code

Hiermit hinterlegen Sie die Umgebungsvariable zum Projekt. Anschliessend müssen Sie folgendes in die noch offene Eingabeaufforderung oder ein Termin eingeben:

godoc -http <IP / Domänenname falls gewünscht>:<Port beliebiger Wahl, empfohlen 3355>

Nun können Sie einen Browser öffnen und die IP / Domänenname, falls angegeben, sonst http://localhost:3355 aufrufen. Hier können Sie nach den Package – Namen suchen, die in diesem Projekt verwendet wurden.

Falls Sie noch umfangreicher mit *godoc* arbeiten wollen, ist nachfolgender Link empfehlenswert:

https://godoc.org/golang.org/x/tools/cmd/godoc