Table of Contents

[Docker 1](#__RefHeading___Toc1403_1035697085)

[Conda Environment 3](#__RefHeading___Toc622_2246420975)

[RecordLinkage 3](#__RefHeading___Toc1419_1035697085)

[Transfer Protocol 4](#__RefHeading___Toc1421_1035697085)

[Protobuf(Protocol buffers) 4](#__RefHeading___Toc1423_1035697085)

[Compiling Protobuf 7](#__RefHeading___Toc438_1441361802)

[Defining Services in Protobuf 7](#__RefHeading___Toc597_1608250445)

[How do Protobuf works 7](#__RefHeading___Toc440_1441361802)

[Advantages of Protobuf 7](#__RefHeading___Toc442_1441361802)

[JSON(JavaScript Object Notation) 8](#__RefHeading___Toc1425_1035697085)

[XML → Extensible Markup Language 8](#__RefHeading___Toc1427_1035697085)

[Cryptocurrency and Blockchain 8](#__RefHeading___Toc1429_1035697085)

[What is Wallet 8](#__RefHeading___Toc1431_1035697085)

[What is Metamask 9](#__RefHeading___Toc1433_1035697085)

[How to Connect Cryptocurrency with SIngularityNet 9](#__RefHeading___Toc1435_1035697085)

# Docker

docker ps -a → to see all the list of docker containers

docker start [container name] → to start the docker container we want

docker exec -it [container name] [command] → to login into the interactive shell of a container

docker stop [container name]

docker ps -l → to see the current docker container

docker rm [container name] or [list of container name]→ to delete the docker container we want

docker container kill [container name]

docker container kill ($docker ps -q) → kill all running containers

docker image ls → to see the list of docker image

-i makes it interactive

-t short for –tty, allocates a pseudo terminal that connects your terminal with the container SDIN

docker start -ai [container name]

* to start container name and at the same time login into the interactive shell
* -a shows all -i make it interactive
* using this is good because when you close the interactive shell with ctr+D at the same time it also close the Service

docker run -ti ubuntu:16.04 then ctl+d

docker start -ai container\_name

docker run [image-name]

docker start ~~ai [container~~name]

**Note**

once you pull the image of the docker then when you call

docker run [image] → it creates new docker container of the image each and every time

so dont use docker run [image] after you make it once otherwise just use docker start then container name of that respective docker image

because if you continously do docker run image then you will not get what you do before because your work is being saved in container so just call that container to access what you edit before

but if you create new container I mean if you do not want what you edit before and you start from the beginning of that docker image then create new container for that image and work on it

docker run [docker-image-name]

then docker itself assigns a new container name for this new container and puts in the latest postion

check it by running

docker ps -a

then start this newly created container

docker start [container name]

to check the current running container just type

docker ps -a then the one that has port number is the current running container

docker run -ti [image-name]

→ to create new docker container and login to interactive shell at the same time

docker image rm [image-name] → to delete specific image

docker image rm or (docker images -a -q) → to delete all image

cat /etc/issue – to check the version of ubuntu in terminal

docker pull ubuntu:16.04 → to pull or install ubuntu 16.04 inside the docker

to install library inside docker container

first apt-get update

apt-get upgrade -y

then install library you want

apt-get install -y git

to copy files or folder from a local machine to docker machine

just open terminal inside your local machine and run the ff

* docker cp <folders or file name> <containername>:<tofolders>

docker cp file.py MY\_SNET\_SERVICE2:/opt/snet

# Conda Environment

To list all the avialable or created environments

conda info –envs

to activate

conda activate [env]

to create new enviroment

conda create –name [env] python=..

# RecordLinkage

the Python Record Linkage Toolkit is a library to link records in or between data sources.

The main features of Python Record Linkage Toolkit

* provides tools needed for record linkage and deduplication
* it contains indexing methods, functions to compare records and classifiers.
* Clean and standardise data with easy to use tools
* make pairs of records with smart indexing methods such as blocking and sorted neighbourhood indexing
* compare records with a large number of comparison and similarity measures for dt types of variables such as strings, numbers and dates.

Record linkage is used to indicate the procedure of bringing information from two or more records that are believed to belong to the same entity.

Record linkage is used to link data from multiple sources or to find duplicates in a single data source.

Record linkage is also known as data matching or deduplication.

The steps of record linkage

cleaning → indexing → comparing → classifying → and evaluation.

Difference between compiler and interpreter in computer ?

Interpreter →

* In **computer** science, an interpreter is a **computer** program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.
* e.g Python is Interpreted langauge means, we do not need to change python language to machine language since computer can understand python code directly. So python has no compiler.

Compiler →

* A **compiler** is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a **computer's** processor uses.
* e.g Java is a compiled language means java code needs to be changed to machine language that computer can understand, so java has its own compiler
* in real world compile means collecting different information together

# Transfer Protocol

→ they are protocols used to encode data and then transfer or send over the network

both are used to encode structured data.

The data format

how to send data ?

What protocol should we use ?

What format should the data be in ?

Size and efficiency consideration ?

What is the requesting server using the data for ?

Good youtube videos

<https://www.youtube.com/watch?v=72mPlAfHIjs>

both Json and Xml are good for the front end but what about for the back end server

## Protobuf(Protocol buffers)

* its a protocol that Google developed internally to enable serialization and deserialization of structured data between different services.
* Serialization means a process of converting the information(attributes) of an Object instance into a binary form, to persist into storage medium or transported over a network.
* Deserialization is a reverse process of converting stream of bits into an Object.
* Google’s design goal was
  + to create a smaller, simpler and quicker than XML
  + to make systems communicate with each other over a wire
  + for the storage of data
  + they place an emphasis on simplicity and performance
* protobuf provides out of the box support in the most common languages including python, java, Objective-C, C#, and others, via Google’s new **proto3** language version
* Google used Protobuf widely for storing and interchaning structured information of all types.
* Protobuf are a language-neutral and platform-neutral extensible mechanism.
* Google describes their use as “You define how you want your data to be structed once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages”
* starting to use protocol buffers is straightforward, simply download and install the protocol buffer compiler.
* First Google developed Protocol Buffers for use in their internal services.
* It is a **binary encoding** format that allows you to specify a **schema** for your data using a specification language.
* Mostly used for internal protocols
* bench marked with XML
* Language Neutral → being neutral is good because it allows us to compile the same protobuf file into different languages like python, java, c++, Go, using the protobuf compiler(protoc). And for each language this protobuf compiler may be different.
* **e.g** to define a search request message format, where each search request has a query string, the particular page number of results, and a number of results per page. Here is the **.proto** message file.

syntax = “proto3”;

package tutorial;

message SearchRequest {

string query = 1;

int32 page\_number = 2; // which page number do we want ?

int32 result\_per\_page = 3; // Number of results to return per page.

enum Corpus {

UNIVERSAL=0; WEB=1; IMAGES=2;LOCAL=3;NEWS=4; PRODUCTS=5;VIDEO=6;

}

Corpus corpus = 4; // 4 means not 4 in the Corpus but the series for the field.

}

* syntax = “proto3” → shows that is protobuf 3 version, if not it assumes as protobuf 2
* package → help us to prevent naming conflicts between different projects.
* **Message →** this is a commands that show the message or Schema definitions

SearchRequest(Schemas)

* + shows the name of message to be defined or already defined
* query, page\_number, and result\_per\_page are attributes or field in Object of SearchRequest
* **Each field has**
  + **type** → that determine how to the data is encoded and sent over the wire
    - There are three field types
      * scalar types → string, double, int32/64, uint32/64, sint32/64, fixed32/64. sfixed32/64, bool, bytes, float
      * enum types → when you want that one fields to only have one of a pre-defined list of values. e.g let’s says you want to add a corpus field for each SearchRequest, where the corpus can be UNIVERSAL, WEB, IMAGES, LOCAL, NEWS, PRODUCTS or VIDEO. So you can do this simply by adding an **enum** to your message. Like above
        + and by the default value is the first defined enum value, which must be 0.
      * Other message types → you can use other message types as field types. e.g if you want to include Result messages in each SearchResponse message in the same .proto

message SearchResponse {

repeated Result results = 1;

}

message Result {

string url = 1;

string tile = 2;

repeated string snippets = 3;

}

* + - * + but if the message type you want to use as a field is already defined in another .proto file, you can import this .proto files to your current .proto files

import “myproject/other\_protos.proto”;

* + - * + Nested Types → you can also define and use message types inside other message types.
  + **Unique number** → used to identify your fields in the message binary format, and should not be changed once your message type is in use. This number identify the unique “tag” that field uses in the binary encoding.
    - * Field numbers in the range 1 through 15 take **one** byte to encode, including field number and field’s type.
      * Field numbers in the range 16 through 2047 take **two** bytes. So you should reserve the numbers 1 through 15 for very frequently occuring message elements.
  + **Keywords or Field Rules** **or Modifiers**→ they are provided for validation and structure, each field must be annotated with one of the following keywords or rules
    - * **required** →a value for the field must be provided, Otherwise the message will be considered “uninitialized”. And serializing an uninitialized message will raise an exception.
      * **optional →** the field may or may not be set. If an optional field value isn’t set, a default value for the field types is used. You can also provide it.
      * **repeated →** this field can be repeated any number of times(including zero). The order of the repeated values will be preserved in the protocol buffer.

Think of repeated fields as dynamically sized arrays.

* + - * + Repeated fields of scalar numeric types use **packed** encoding by default.
      * **singular →** a well-formed message can have zero or one of this field(but not more than one). And this is the default field rule for proto3 syntax.
      * **reserved** →
      * each field must be annotated with one of the above modifiers
      * Note → you should be very careful about marking fields as **required.** If at some point you wish to stop writing or sending a required field, it will be problematic to change the field to an optional field. Some engineers at Google have come to the conclusion that using **required** does more harm than good; they prefer to use only **optional** and **repeated**. However, this view is not universal.
* to add comments in protobuf(.proto) file use // and /\* … \*/
* When you run the protocol buffer compiler on a .proto file, then the compiler generates the code in your chosen language
* you can add an optional package specifier to a .proto file to prevent name clashes between protocol and message types

### Compiling Protobuf

Compiling Protobuf means generating the java, python, C++, Go, Ruby, Objective-C or C# code that you need to work with the message types defined in your .proto file.

And the file type of this generated code is binary format which is saved as in respective of your language choice. Either .py, .java , .cpp, ….

note the file is also saved as name\_pb2.py

With Protocol buffers, you write a .proto description of the data structure you wish to store.

From this .proto, the protobuf compiler creates a class that implements automatic encoding and parsing of the protocol buffer data with an efficient binary format.

The generated binary format class

* provides getters and setters for the fields that make up a protocol buffer and
* takes care of the details of reading and writing the protocol buffer as a unit.

To do this you need to run the protobuf compiler(protoc) on the .proto file.

So first you have to install protobuf compiler, check if its installed or not on you computer by opening terminal and then type **protoc -help.** If not installed install it by ff [here](https://developers.google.com/protocol-buffers/docs/downloads)

you can also provide one or more .proto file at the same time.

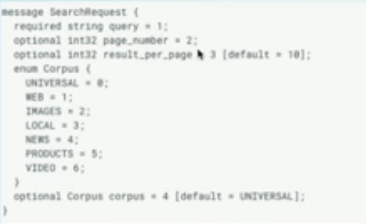
What will be compiled is the .proto file or the **Schema**

so there will be two different file types

1. the .proto – which shows the Schema of the data, the attributes it contains, their types, rules, keywords, unique number and other

w/n you write the schema of protobuf it must be similar to what is found in your code.

1. the compiled file into the language of you choice. The message of the protobuf all are in binary format.



**E.g**, the Protobuf compiler is invoked as follows:

protoc -I=$SRC\_DIR --python\_out=$DST\_DIR $SRC\_DIR/path/to/file.proto

* -I → specifies a the source directory where your application’s source code lives(the current directory is used if you don’t provide a value). Here you can also use --proto\_path=$SRC\_DIR
* --python\_out → specifies language of you choice, you have also an alternative like [--java\_out, --cpp\_out, --got\_out, --ruby\_out, --objc\_out, --csharp\_out]
* $DST\_DIR → specifies the destination directory where you want the generated code to go; often the same as $SRC\_DIR

**E.g** check from the code, that is a very simple “address book” with Python. Its an application that can read and write people’s contact details to and from a file. Each Person in the address book has a name, an ID, an Email address, and a contact phone number.

The protobuf file or .proto file syntax is similar to Java or C++

Note the proto2 and proto3 are very different from each other. There is many updations in protobuf 3 so some codes that work in proto2 is not work in proto3. Check both in the code proto2 → addressbook2.proto proto3 → addressbook3.proto

**Note** the protobuf compiler generates as **name\_pb2.py** for both proto2 and proto3.

compile the defined proto2 and proto3 by writing the ff code in the terminal

* protoc --python\_out=/home/amante/snet/snet-platform/Tutorials/code/ addressbook2.proto
  + generated class→ addressbook2\_pb2.py
* protoc --python\_out=/home/amante/snet/snet-platform/Tutorials/code/ addressbook3.proto
  + generated class → addressbook3\_pb2.py

some difference between Protobuf 2 and Protobuf 3

probuf 3 is defined as syntax=”proto3” while protobuf 2 is as syntax=”proto2”

**In protobuf 3**

* the “required” keyword is not allowed because ratherthan its advantage its disadvantage is bigger
* the “optional” keyword is not allowed because by default field types are optional
* Explicit default value by our self is not allowed since it takes the first elt from the list as default value

### Defining Services in Protobuf

If you want to use your message types with an **RPC**(Remote Procedure Call) system,

* + - define an RPC service interface in a .proto file, then
    - The protocol buffer compiler will generate **service interface code** and **stubs** in your chosen language

**e.g** if you want to define an RPC service with a method that takes your SearchRequest and returns a SearchResponse, you can define it in your .proto file as follows:

service SearchService {

rpc Search (SearchRequest) returns (SearchResponse)

}

* + 1. The most straightforward RPC system to use with protobuf is **gRPC**

**gRPC →** is a language and platform neutral open source RPC system developed at Google. It works particularly well with protocol buffers and lets you generate the relevant RPC code directly in your .proto files using a special protocol buffer compiler plugin.

You can also use protobuf with other RPC system.

### How do Protobuf works

* Binary Serialization means they encode data into binary format and which is easy to transfer.
* Uses a determined schema to encode and decode
* compiled to many languages

### Advantages of Protobuf

* Light Weight
  + Takes up less space
  + Faster transmission
* validation of data structure or objects
* easy to modify schema

## JSON(JavaScript Object Notation)

* is minimal(encoded) readable format used to structure data
* its an alternative to XML and is mainly used as a way of transmitting data between a server and web application.
* JSON uses human-readable way, attribute – value paris and array data types (and another type of serializable value)
* JSON was derived from JavaScript

## XML → Extensible Markup Language

# Cryptocurrency and Blockchain

The purpose of cryptocurrency is to allow people to manage their funds in a anonymous and secure way, from any location, without relying on third parties.

On the blockchain, your digital assets are not controlled by any bank or government. You are the only one who has access to your funds and you can instantly transfer them to any other address on the blockchain without depending on authorizations, permissions, or limits.

Your public address and your private key are the only pieces of information you need to hold and manage your funds from anywhere in the world.

### What is Wallet

Creating wallet [https://www.myetherwallet.com](https://www.myetherwallet.com/)

### What is Metamask

### How to Connect Cryptocurrency with SIngularityNet

watched videos

<https://www.youtube.com/watch?v=PC2sAJyyUTs&source=post_page--------------------------->

<https://www.youtube.com/watch?v=S7Rtk_k4Hs0>

References

<https://developers.google.com/protocol-buffers/docs/pythontutorial>