Relazione progetto gestione di reti:

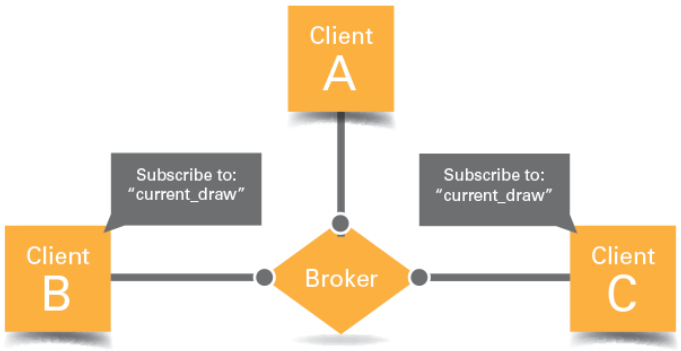
- **Scopo**
  Tramite protocollo MQTT, interrogare sensori di temperatura connessi ad internet e costruire grafici relativi.

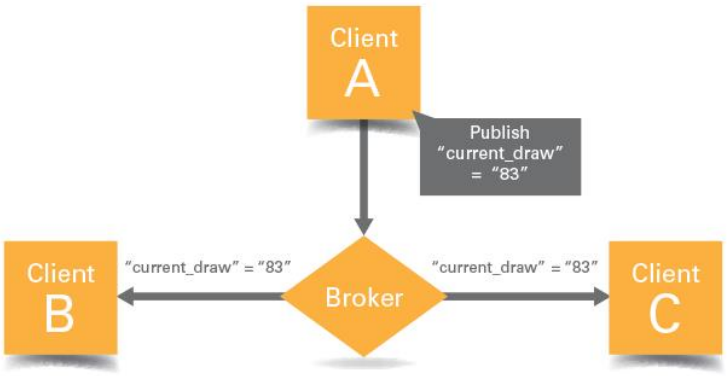
- **Fase 1: Individuazione dei sensori in rete**

  Per trovare sensori in rete ho utilizzato Shodan.io
  Shodan.io è un motore di ricerca che trova in internet e salva nel proprio database informazioni non rese private da dispositivi connessi ad internet, permettendo poi agli utenti ricerche specifiche nel database.

- **Fase 2: Creazione del client MQTT in java**

  - **Il protocollo MQTT**
    MQTT (Message Queue Telemetry Transport), protocollo di messaggistica leggero di tipo publish subscribe posizionato in cima a TCP/IP.
    Al posto del modello client/server di HTTP, il protocollo MQTT adotta un meccanismo di pubblicazione e sottoscrizione per scambiare messaggi tramite un appostivo "message broker". Invece di inviare messaggi a un determinato set di destinatari, i mittenti pubblicano i messaggi su un certo argomento (detto topic) sul message broker, Ogni destinatario si iscrive agli argomenti che lo interessano e, ogni volta che un nuovo messaggio viene pubblicato su quel determinato argomento, il message broker lo distribuisce a tutti i destinatari. In questo modo è molto semplice configurare una messaggistica uno-a-molti.





MQTT è un protocollo di messaggistica estremamente leggero progettato per dispositivi limitati e reti a bassa larghezza di banda, alta latenza o sostanzialmente inaffidabili. I principi su cui si basa sono quelli di abbassare al minimo le esigenze in termini di ampiezza di banda e risorse mantenendo nel contempo una certa affidabilità e grado di certezza di invio e ricezione dei dati. Protocollo orientato all IoT ed a quelle applicazioni mobili per le quali va tenuto in maggior conto il consumo di banda in rete e di energia dei dispositivi

- **Client in java**

```
public static void main(String[] args) {
    // TODO Auto-generated method stub

    try {

        while (true) {

            Scanner s = new Scanner(new File("./Servers.txt"));
            int i = 0;
            while(s.hasNextLine()) {
                System.out.println(i);
                i++;
                MemoryPersistence persistence = new MemoryPersistence();
                String ip =s.nextLine();
                String server = "tcp://"+ip+":1883";
                 MqttClient client = new MqttClient(server,"ciao",persistence);

                 client.setCallback(new MyMqttCallback(ip,client));
                 MqttConnectOptions o = new MqttConnectOptions();
                 o.setConnectionTimeout(10);
                 try {
                     client.connect(o);

                     client.subscribe("sensor/temperature");}
                 catch(MqttException e) {};
                 Thread.sleep(1000);

            }

            s.close();
            Thread.sleep(50000);
        }
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
```

Questo è il codice del main del client MQTT in java. Ho utilizzato Eclipse Paho una libreria java per MQTT.

Per ogni dispositivo il cui ip è salvato nel file creo un client mi collego col dispositivo sottoscrivendomi al topic “sensor/temperature” e aspetto quindi la il messaggio dal dispositivo.

Al momento dell’arrivo del messaggio viene richiamata la Callback

```java
@Override
public void messageArrived(String arg0, MqttMessage arg1) throws Exception {
    File dbFile = new File(cityDatabase);
     DatabaseReader reader = new DatabaseReader.Builder(dbFile).build();
     InetAddress ipAddress = InetAddress.getByName(ip);
       CityResponse response = reader.city(ipAddress);
       Country country = response.getCountry();
       System.out.println("Country Name: "+ country.getName());
       System.out.println(ip);
       Subdivision subdivision = response.getMostSpecificSubdivision();
       City city = response.getCity();
       System.out.println("City Name: "+ city.getName());
       Postal postal = response.getPostal();
       System.out.println("Codice Postale: "+postal.getCode());
    System.out.println("Temperatura "+arg1);
    InfluxDB influxDB = InfluxDBFactory.connect("http://localhost:8086");

    System.out.println(influxDB.databaseExists("Temperature"));
    if(!influxDB.databaseExists("Temperature"))
        influxDB.createDatabase("Temperature");
    Point point = Point.measurement(city.getName())
            .addField("stato", country.getName())
            .addField("codice postale", postal.getCode())
            .addField("temperatura",Double.parseDouble(arg1.toString()))
            .time(System.currentTimeMillis(), TimeUnit.MILLISECONDS).
            build();
    influxDB.setDatabase("Temperature");
    influxDB.write(point);
    this.client.disconnect();
    this.client.close();

}
```

Questo è il codice della callback. Ho utilizzato GeoIp, una libreria che mi fornisce classi per accedere a un database con informazioni geografiche a partire dall'ip.

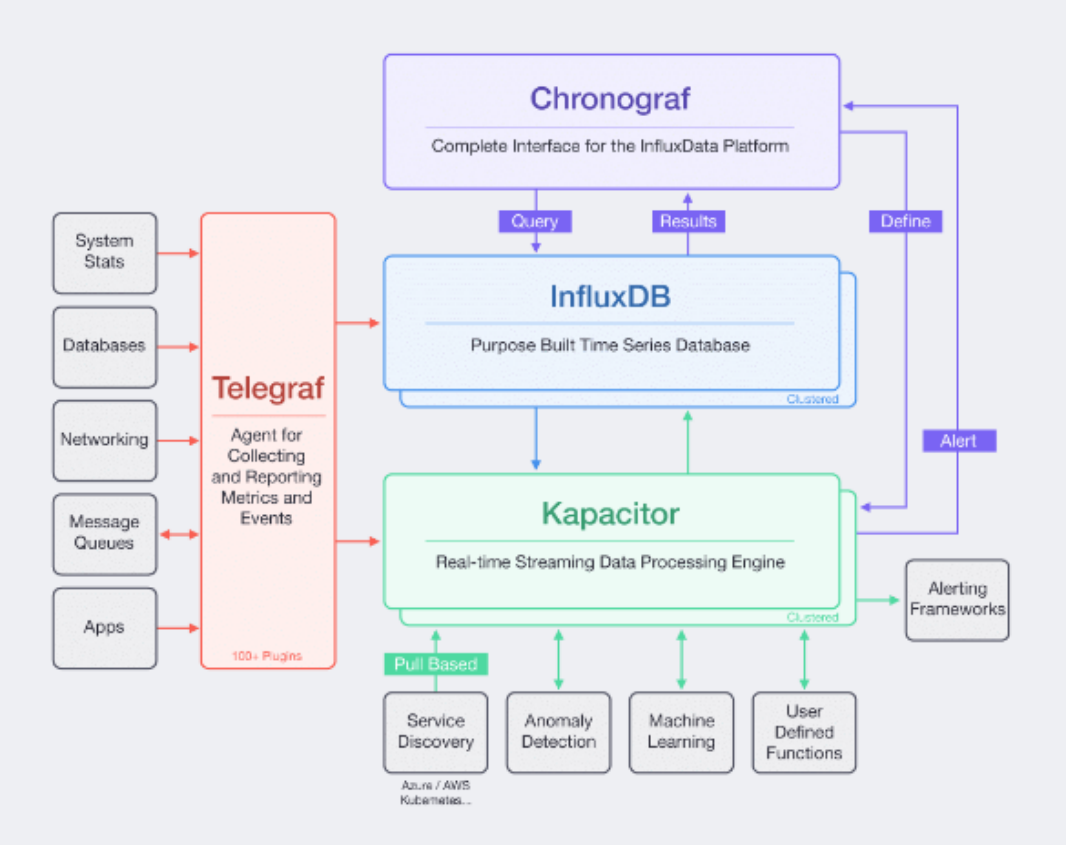
Quando arriva un messaggio dal dispositivo mi collego al database influxDB e inserisco una nuova entry con le informazioni prese dal database GeoIp e la temperatura ricevuta per messaggio.

Esempio di output del programma

```
0
Country Name: Croatia
176.62.39.73
City Name: Zagreb
Codice Postale: 10000
Temperatura 26.00
1
true
Country Name: Argentina
200.0.183.37
City Name: Mar del Plata
Codice Postale: 7600
Temperatura 23.00
true
2
Country Name: Italy
77.73.61.29
City Name: Rome
Codice Postale: 00187
Temperatura 32
true
3
Country Name: United States
98.176.121.123
City Name: San Diego
Codice Postale: 92114
Temperatura 66.50
true
4
Country Name: India
35.154.237.77
City Name: Mumbai
Codice Postale: null
Temperatura 158
true
```

- **Fase 3: Creazione dei grafici**

  - **InfluxData TICK Stack**
    InfluxData TICK Stack è una piattaforma open source che permette di accumulare, analizzare ed agire su dati di serie temporali.
    TICK stack è composto da 4 componenti: Telegraf, InfluxDB, Chronograf, Kapacitor.

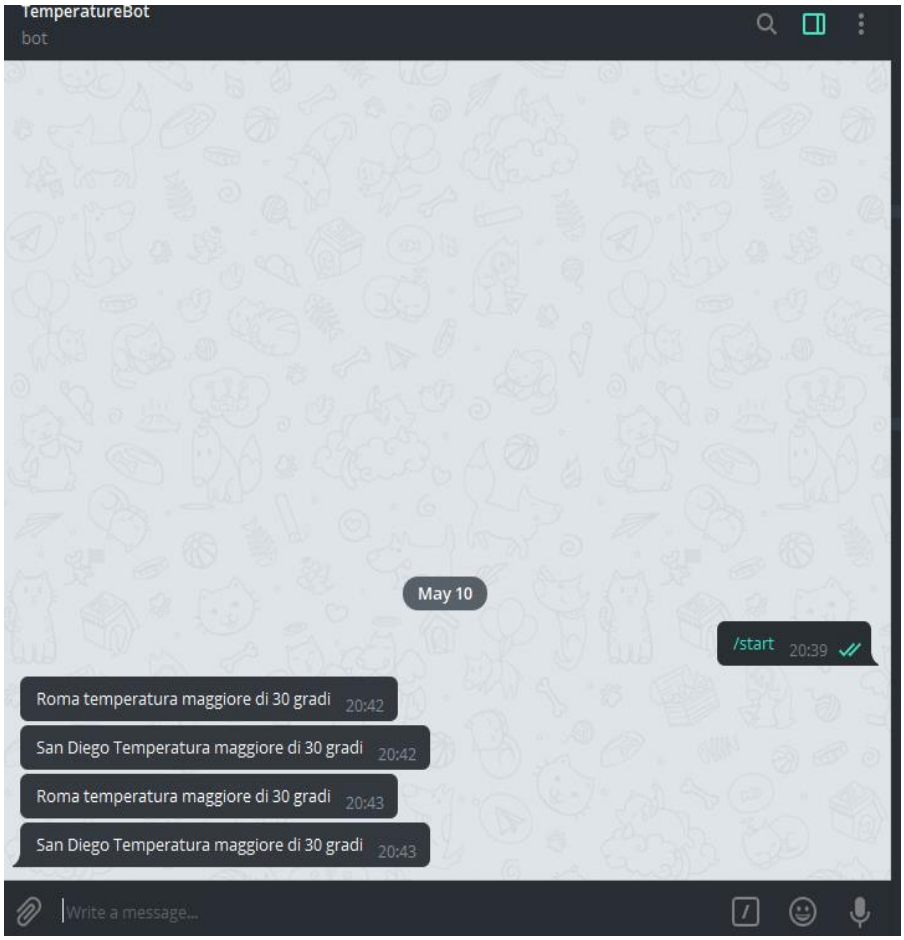


    - *Telegraf*
      Telegraf è un agente plugin driven che contiene plugin per collezionare metriche riguardo il sistema sul quale viene eseguito come demone. Comunica con InfluxDB inserendovi i dati rilevati.
    - *InfluxDB*
      InfluxDB è un database specializzato per dati di serie temporali.
    - *Chronograf*
      Chronograf è una interfaccia grafica, permette di creare grafici temporali eseguendo query su influxDB.
    - *Kapacitor*
      Kapacitor comunica con influxDB analizzando i dati e inviando allarmi a Chronograf secondo i criteri specificati.

- **Esempio di grafici**

Sulla sinistra ci sono i grafici temporali che mostrano le varie rilevazioni di temperatura nel tempo. A destra viene mostrato l'ultimo valore di temperatura rilevato.

- **Esempio di allarmi**



Ho configurato kapacitor per inviare gli allarmi ad un bot Telegram creato da me. In questo caso avevo creato degli allarmi nel caso che la temperatura avesse superato i 30 gradi.
Con kapacitor si possono creare vari tipi di allarmi con politiche diverse, vari tipi di soglie , tempo di inattività o cambiamenti significativi nei valori rilevati.

.