Teemu Pahkala

Tiedonkeruu- ja etävalvontajärjestelmä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Automaatiotekniikka

Insinöörityö

1.5.2017

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Teemu Pahkala Tiedonkeruu- ja etävalvontajärjestelmä 42 sivua 1.5.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Automaatiotekniikka
Suuntautumisvaihtoehto	
Ohjaaja(t)	Lehtori Jukka Pirinen Ohjausasiantuntija Jouni Rikkonen

SKS:llä oli tarvetta saada konsepti, jolla voidaan kerätä ajonaikaisia arvoja Unidrive M700 -taajuusmuuntaja ja -servokäytöstä. Opinnäytetyön tarkoituksena oli rakentaa tällainen järjestelmä. Järjestelmän ensimmäisessä versiossa päädyttiin käyttämään avoimen lähdekoodin ratkaisuja, kuten Node-RED, InfluxDB ja Grafana. Näillä ohjelmistoilla pystyttiin osoittamaan ratkaisun toimivuus ja kyettiin rakentamaan toimiva järjestelmä parissa kuukaudessa. Järjestelmä koostuu kolmesta osasta Datankeräin, tietokantapalvelin ja käyttöliittymä.

Työssä käydään läpi, minkälaisista laite- ja ohjelmistokomponenteista järjestelmä koostuu. Selitetään yksityiskohtaisesti Datankeräimen Node-RED-ohjelman toiminta, sekä esitellään tuloksia Datankeräimen käyttötesteistä.

Tiedonkeruu- ja etävalvontajärjestelmä rakennettiin toimivaksi laboratorioverkossa. Kenttätestissä käytettiin VPN-yhteyttä tietokantapalvelimen ja Datankerääjän välillä. Kenttätestejä ehdittiin ajaa noin kaksi viikkoa, joiden aikana Datankerääjä toimi odotetusti.

Avainsanat	Iot, Node-RED, InfluxDB, Grafana, Raspberry Pi

Author(s) Title Number of Pages Date	Teemu Pahkala Data collecting and remote monitoring system 42 pages 1 May 2017
Degree	Bachelor of Engineering
Degree Programme	Automation technology
Specialisation option	
Instructor(s)	Jukka Pirinen, Senior lecturer Jouni Rikkonen, Control specialist

SKS had need for a concept, which could be used to gather data from Unidrive M700 AC drive during use. The purpose of this thesis was to build a system for this kind of data collecting. The first version of this system was made by using open source software like Node-RED, InfluxDB and Grafana. With these software, it was possible to prove that this concept works and build a working system. The system consists of three components which are Data collector, Database and user interface.

This thesis explains what kind of hardware and software components the system consists of. It gives a detailed explanation of Data Collector's node-RED program's operation and presents the results of the test done on Data Collector.

The data collection and remote monitoring system was built in laboratory network. VPN-connection was used in field test to connect the database server to Data Collertor. Field tests were conducted for about two weeks. During that time, Data Collector worked as expected.

Keywords	Iot, Node-RED, InfluxDB, Grafana, Raspberry Pi

Sisällys

Lyhenteet

IOPS *input/output operations per second*. Käytetään määrittelemään, kuinka monta luku- ja kirjoitusoperaatiota sekunnista kovalevylle tai muistikortille voidaan tehdä.

TCP *Transmission Control Protocol*. Tietoliikenneprotokolla, jolla luodaan yhteyksiä laitteiden välille.

VPN *Virtual Private Network.* Virtuaalinen erillisverkko muodostaa suojatun yhteyden kahden eri verkon välille.

CPU *Central Processing Unit.* Prosessori tai suoritin.

1 Johdanto

SKS Controllilla on ollut tarvetta järjestelmälle, jolla voidaan seurata yrityksen myymän Unidrive M700 -taajuusmuuntaja ja -servokäytön toimintaa. Toimintaa seuraamalla voidaan saada tietoa lisätietoa koko laitteiston toiminnasta, jonka moottoreita ohjataan M700-drivella. Tällöin järjestelmää voidaan myydä asiakkaille, jotta he voivat seurata tarkemmin laitteidensa toimintaa. Varsinkin laitteiden toiminnan seuraaminen pitkällä aika välillä saattaa paljastaa tuotannosta jotain, jota olisi muuten hankala havaita.

Opinnäytetyön tarkoituksena oli rakentaa tiedonkeruu- ja etävalvontajärjestelmä, jolla olisi mahdollista seurata laitteiden toimintaa reaaliajassa, sekä katsoa historiatietoja laitteen toiminnasta.

Tiedonkeruu- ja etävalvontajärjestelmä kehitettiin avoimenlähdekoodin ohjelmilla. Järjestelmässä tuli olla palvelimella toimiva tietokanta ja ohjelma, jolla voidaan visualisoida tietokantaan kerättyä dataa, sekä Datankerääjälaite, joka kerää tietoa taajuusmuuntaja ja servokäytöltä ja lähettää tiedon tietokantaan.

2 Ohjelmistot

2.1 Node.js

Node.js on JavaScript-pohjainen ohjelmistokehitysalusta, joka on rakennettu Google Chromen JavaScript V8 Enginen päälle (1).

Node.js sopii hyvin I/O sidottuihin sovelluksiin, joiden täytyy käsitellä useita samanaikaisia yhteyksiä. Hyvä esimerkki I/O-sidotuista sovelluksista on data-intensive realtime -sovellukset (DIRT), kuten striimaus-sovellukset. Node.js sopii huonommin CPU-raskaisiin sovelluksiin (1;2.)

Node.js on non-blocking, eli se ei jää odottamaan vastausta esimerkiksi I/O-laitteelta vaan jatkaa koodin suorittamista (2).

2.2 Node-RED

Node-RED on IBM:n kehittämä, Node.js:n päällä toimiva, nettiselainpohjainen avoimen lähdekoodin työkalu, jolla voidaan rakentaa sovelluksia esineiden internettiin. Node-RED:ä suunniteltaessa on keskitytty tekemään ohjelmoinnista yksinkertaista käyttämällä koodiblokkeja eli nodeja, joita voidaan kytkeä yhteen johdoilla. Node-RED-ohjelma koostuu yleensä sisääntulo-, prosessointi- ja ulostulo-nodeista. Yhdessä nämä nodet muodostavat tietovuon eli Flow'n. Flow'ssa viestiä siirrellään nodejen välillä JavaSricpt-objektina nimeltä *msg*. *msg.payload* on paikka, jonne kuljetettava viesti on tallennettu (3.)

Node-RED sai alkunsa vuonna 2013 Open Source projektina IBM:llä. IBM:n tarvitessa ohjelman, jolla voitiin nopeasti kytkeä rautaa ja laitteita nettipalveluihin ja toisiin ohjelmiin. Tämä kuitenkin kasvoi nopeasti IoT:n yleiskäyttöiseksi ohjelmointityökaluksi. Node-RED on nopeasti kehittänyt merkittävän, kasvavan käyttäjäkunnan ja aktiivisen kehittäjä yhteisön, jotka tuottavat uusia nodeja, mikä laajentaa Node-RED:n käyttömahdollisuuksia ja -alueita (3.)

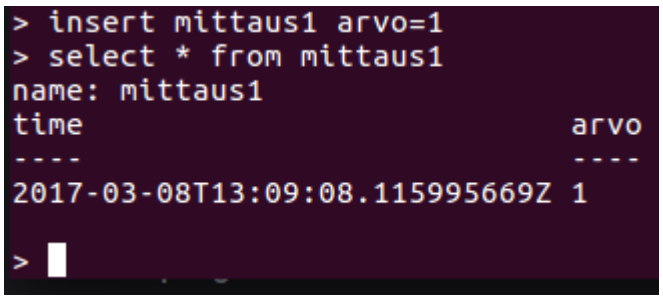
2.3 InfluxDB

InfluxDB on avoimen lähdekoodin aikasarjatietokanta, jonka on kehittänyt InfluxData. InfluxDB on kirjoitettu GO-kielellä, sekä se on optimoitu nopeaa kirjoittamista ja lukemista varten. InfluxDB on kehitetty datakeskuksia ja IOT-laitteita silmällä pitäen (4;5).

Aikasarjatietokannassa tiedot indeksoidaan aikaleiman mukaan. InfluxDB:ssä normaalin SQL-pohjaisen tietokannan taulukkoa vastaa measurement, jonka alle tallentuvat kaikki pisteet (Point). Measurement koostuu vähintään aikaleimasta ja yhdestä ”field”-arvosta (4;5).

Kuvassa 1 on esimerkki measurementista, jossa on aikaleima ja yksi Field Key. Measurement on *mittaus1*, aikaleima on *time* ja Fiel Key on *arvo*. *mittaus1*-measurementiin on tallennettu yksi piste, jonka aikaleima on *2017-03-08T13:09:08.115995669* ja Field Key -arvo on *1*.

InfluxDB-tietokantaa voidaan hallita komentorivipohjaisella CLI-ohjelmalla. CLI-ohjelmassa InfluxDB käyttää perus SQL-syntaksia (5.)



Kuva 1. InfluxDB CLI käyttöliittymässä on tehty ylimmällä rivillä pisteen tallennus mittaus1 measurementiin. Toisella rivillä on tehty pisteiden haku mittaus1 measurementista.

Raspberry Pille InfluxDB:stä oli asennettava versio, joka on tehty ARM-prosessorille. ARM-versiot näyttävät tulevan jäljessä PC-versioihin nähden. Näin ollen Raspberryyn jouduttiin asentamaan vanhempi versio InfluxDB-ohjelmasta, kuin palvelimena toimineelle Linux-koneelle.

2.4 Grafana

Grafana on ohjelma, jolla voidaan helposti luoda graafisia valvontanäyttöjä. Grafana on avoimeen lähdekoodiin perustuva ja täten ilmainen ohjelma. Grafana on tehty käytettäväksi aikasarjaisten tietokantojen kanssa ja se tukeekin monia eri aikasarjatietokantoja kuten InfluxDB:tä, jota käytetään tässä opinnäytetyössä tietokantana. Grafana on alun perin tehty datakeskusten ja konesalien monitorointia varten, mutta sen käyttö on levinnyt muille teollisuuden aloille (6.)

2.4.1 Käyttäjätasot

Grafanassa on mahdollista luoda neljä eri käyttäjätasoa, joilla on eri oikeuksia. Käyttäjätasot ovat Admin, Editor, Read Only Editor ja Viewer (6.)

2.4.2 Tuki ja dokumentaatio

Grafanalle löytyy netistä ohjeet peruskäyttöä varten. Jos haluaa tehdä jotain vaativampia ominaisuuksia käyttöliittymään tai jos Grafanan käytössä tulee jotain ongelmia vastaan, on ratkaisujen löytäminen näihin yleensä hankalaa. Käyttöohjeiden ja dokumentaation puute johtuu Grafanan liiketoimintamallista, jossa ohjelma on ilmainen, mutta palveluista joutuu maksamaan (6.)

3 Laitteisto

3.1 Palvelin

Tietokantapalvelimeksi ostettiin käytetty PC, johon asennettiin Linux-käyttöjärjestelmä. Taulukossa 1 on palvelimen tekniset tiedot.

Taulukko 1. InfluxDB- ja Grafana-palvelin

Käyttöjärjestelmä Ubuntu 16.04 - InfluxDB v1.2.0 - Grafana v4.1.1
AMD Athlon 2 x2 220 2.8GHz
ATI Radeon HD 4200
4GB RAM
250GB HDD

3.2 Datankerääjä

Datankerääjäksi ostettiin Raspberry Pi 3. Alla on luettelo ohjelmistoista, jotka asennettiin Raspberryyn, sekä ohjelmistojen versionumerot.

Raspberry Pi 3

- muistikortti 16Gb
- käyttöjärjestelmä Raspbian
- Node.js v6.9.4
- Node-RED v0.16.2

  lisätyt nodet
  - node-red-contrib-modbus 1.0.1
  - node-red-contrib-influxdb 0.0.6
  - node-red-contrib-isonline 1.1.15

- InfluxDB v1.0.2 armhf (https://packages.debian.org/stretch/influxdb)

3.3 Unidrive M700

Kuvassa 2 olevalla Unidrive M700:lla pystytään ohjaamaan servomoottoreita, sekä oikosulkumoottoreita. M700:ssa pyörii Modbus TCP -palvelin, sekä siinä on Codesys-pohjainen ohjelmointi ympäristö. Datankerääjä kommunikoi M700:sen kanssa Modbus TCP -protokollaa käyttäen. Yleisesti Unidrive M700:sta käytetään yleisesti nimitystä drive.

Kuva 2. M700 on Emersonin tekemä servovahvistin/taajuusmuuntaja.

3.4 Conel UR5i v2 Libratum

Kuvassa 3 on 3G VPN-reititintä. Reititintä tarvitaan kenttätestauksessa. Tietokantapalvelimeen ei ollut mahdollista saada yhteyttä muuten kuin VPN:n avulla.

Kuva 3. Conel UR5i v2 Libratum on kaksiporttinen 3G-teollisuusreititin, jossa on VPN-tuki.

4 Datankerääjä

Datankerääjä on tiedonkeruu- ja etävalvontajärjestelmän pääkomponentti. Datankerääjän raudaksi valittiin Raspberry Pi 3. Raspberry Pi 3 on pienikokoinen ja halpa verrattuna teollisuus PC:hin, mutta kuitenkin Linux-pohjaisen Raspbian-käyttöjärjestelmänsä ansiosta monipuolinen.

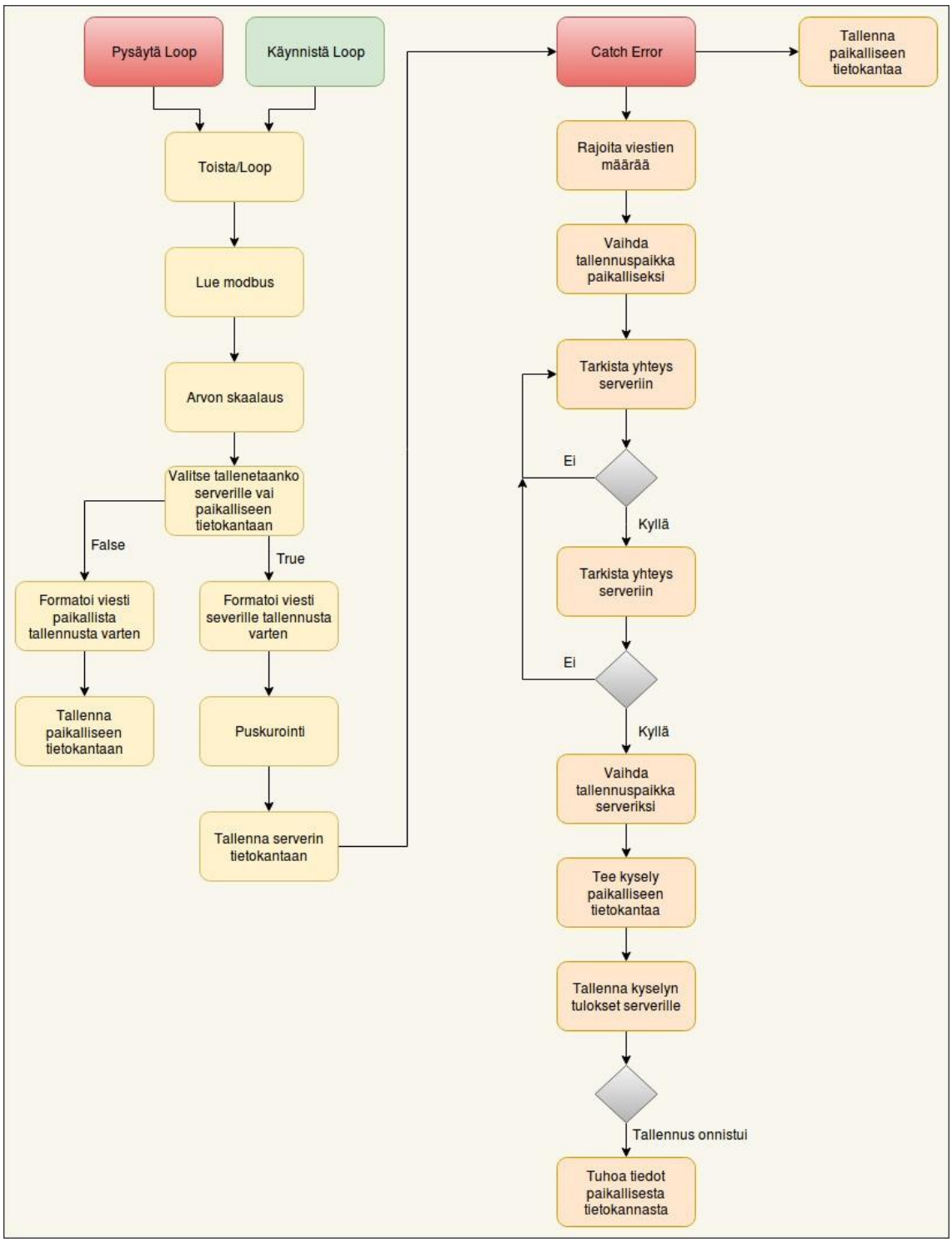
Datankerääjän rakentaminen alkoi vaatimusmäärittelyiden tekemisellä (Liite 1). Työn tekeminen aloitettiin SKS:n tiloissa laatimalla vaatimusmäärittelyt yhdessä SKS:n Jouni Rikkosen ja Tommi Suomelan kanssa. Kun vaatimusmäärittelyt oli tehty, piti selvittää, mitä ohjelmia Raspberryyn on asennettava. Ensimmäisenä Raspberryyn piti päivittää uusimmat versiot Node.js:stä ja Node-RED:stä. Node-RED:iin oli myös asennettava tarvittavat lisä-nodet. Datankerääjässä pitää olla mahdollisuus puskuroida mittaustietoja paikalliseen muistiin, silloin kun ei ole yhteyttä palvelimeen. Tietojen puskurointia varten Raspberryyn asennettiin InfluxDB-tietokantaohjelmasta versio, joka toimii arm-mikroprosessoriarkkitehtuuriin perustuvalla prosessorilla.

Suurin osa opinnäytetyön ajasta meni Node-RED-ohjelman tekemiseen, koska ohjelma ei ollut kovinkaan tuttu. Aikaa kului huomattavasti selvittämiseen, miten Node-RED-ohjelmaa käytetään ja miten sillä voidaan tehdä haluttuja asioita. Yksi aikaa vievimmistä

asioista oli ohjelmoida logiikka, joka yhteyden katketessa palvelimelle vaihtaa tallennuspaikaksi Raspberryssä pyörivän InfluxDB-tietokannan ja yhteyden palautuessa vaihtaa tallennuspaikan takaisin palvelimelle. Jokaisessa nodessa on vain yksi sisääntulo, joten Node-RED:ssä ei pysty logiikkaohjelmoinnin tapaan ohjaamaan lähtöä useamman tulon perusteella.

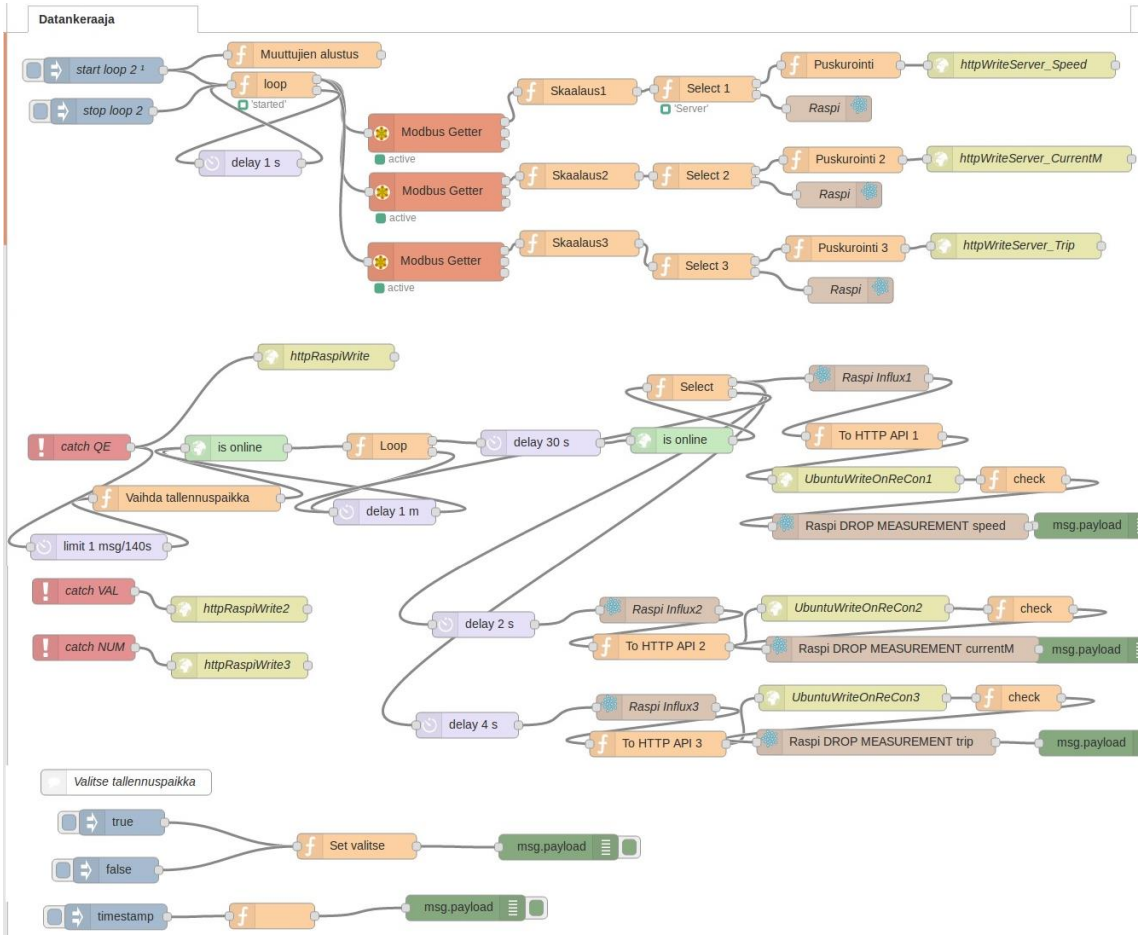
5 Datankerääjän ohjelmakoodi

Node-RED:llä datankerääjään tehty ohjelma käynnistyy itsestään Raspberryn käynnistyksen yhteydessä. Ohjelma on mahdollista pysäyttää ja käynnistää manuaalisesti Start- ja Stop Inject -nodeilla. Ohjelma käy lukemassa Modbus-palvelimelta halutun tiedon ja lähettää sen palvelimella olevaan InfluxDB-tietokantaan. Jos yhteys palvelimeen on poikki, tallennetaan tiedot Raspberryssä olevaan InfluxDB-tietokantaan. Kun yhteys palvelimeen saadaan uudelleen muodostettua, lähetetään Raspberryn tietokantaan kerätyt tiedot palvelimelle, jonka jälkeen Raspberryssä olevat tiedot poistetaan. Kuvassa 4 on lohkokaaviokuva Datankerääjän ohjelman toiminnasta.



Kuva 4. Lohkokaavio datankerääjän ohjelmasta.

Kuvassa 5 näkyy koko Datankerääjän Node-RED-koodi. Kaikki flow't on sijoitettu yhdelle välilehdelle. Flow't on jaettu kolmeen osaan. Ensimmäinen osa on Pää-flow, toinen osa yhteyden katkeamisen käsittely ja kolmas on manuaalinen käyttö ja testaus.



Kuva 5. Kaikki Datankerääjän flow’t.

5.1 Käytetyt nodet

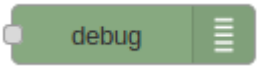
Kuvan 6 *inject*-nodella voidaan käynnistää flow. Inject-nodessa pystyy määrittelemään mitä node laittaa msg.payload:iin. Vakioasetuksilla *inject*-node laittaa aikaleiman *msg.payload*iin

Kuva 6. *Inject*-node

Kuvan 7 *catch*-nodella saadaan kiinni samalla välilehdellä olevien nodejen lähettämät virheviestit. Samalla *catch*-node saa virhetiedot ja viestin jota käsiteltäessä virhe syntyi.

Kuva 7. *Catch*-node

Kuvan 8 *debug*-nodella pystyy helposti näyttämään viestit debug-paneelissa. *Debug*-nodessa voi valita haluaako näyttää pelkän msg.payload:n vai koko msg-objektin.



Kuva 8. *Debug*-node

Kuvan 9 *function*-nodeen voi itse kirjoittaa standardi JavaScript-koodia, joten se voidaan ohjelmoida tekemään monenlaisia monimutkaisia tehtäviä. *Function*-nodeen voidaan luoda useampia ulostuloja.

Kuva 9. *Function*-node

Kuvan 10 *delay*-nodella voidaan viivästyttää viestien kulkua flow'ssa tai rajoittaa ajallisesti viestien määrää jotka pääsevät läpi nodesta. Esimerkiksi node voidaan säätää niin, että vain se päästää läpi vain yhden viestin minuutissa ja hylkää muut sinä aikana tulleet viestit. *Delay*-node voi myös puskuroida nopeasti tulevia viestejä ja päästää ne läpi minuutin välein tulo järjestyksessä.

Kuva 10. *Delay*-node

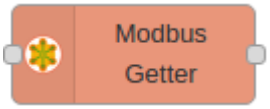
Kuvan 11 *influxdb in* -nodella voidaan lähetää käskyjä, kuten kyselyt ja poistot, InfluxDB-tietokantaan. Tehtäessä kysely tietokantaan *influxdb in* -node palauttaa kyselyn tulokset msg.payloadissa taulukkona (array).

Kuva 11. *Influxdb in* -node

Kuvan 12 *influxdb out* -nodella voidaan kirjoittaa InfluxDB-tietokantaan.

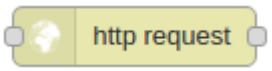
Kuva 12. *Influxdb out* -node

Kuvan 13 *Modbus Getter* -nodella voidaan lukea Modbus-palvelimelta tietoja. *Modbus Getter* -node palauttaa luetun arvon taulukossa.



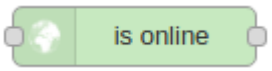
Kuva 13. *Modbus Getter* -node

Kuvan 14 *http request* -nodella on mahdollista lähettää HTTP-pyyntö määriteltyyn osoitteeseen. Headers ja payload ovat täysin konfiguroitavissa.



Kuva 14. *http request* -node

Kuvan 15 *is online* -nodella voidaan pingata haluttua ip-osoitetta. Jos pingiin vastataan, node lähettää ”true”-viestin eteenpäin. node voidaan myös konfiguroida päästämään tullut viesti läpi sen sijaan, että lähettäisi ”true”- tai ”false”-viestin.



Kuva 15. *Is online* -node

5.2 Datankerääjän ohjelmakoodin ensimmäinen osa

Kuvassa 16 on Node-RED-koodin ensimmäinen osa, joka on edelleen jaettu 5 osaan. Ensimmäistä flow'a voidaan sanoa Pää-flow'ksi. Pää-flow on koko ajan toiminnassa. Siinä tapahtuu tietojen luku M700 Drivesta ja niiden tallennus tietokantaa.

Osa 1:ssä on käynnistys ja pysäytys napit toistosilmukalle. *loop*- ja *delay*-node luovat toisto silmukan, jossa *delay*-node määrittelee millä nopeudella *flow*'n lähetetään viestejä. *start loop*- ja *stop loop* -nodeilla ohjataan toistosilmukkaa. *start loop* -node on myös yhdistetty *Muuttujien alustus* function -nodeen. Aina kun *start loop* -node syöttää viestin flow'n *Muuttujien alustus* -nodessa alustetaan globaalit muuttujat default arvoihin. (7)

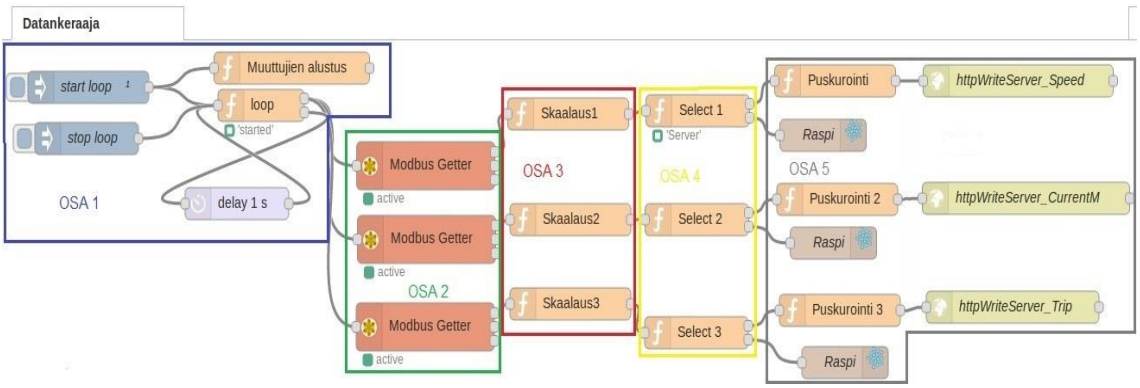
Osa 2:ssa luetaan Modbus TCP -protokollalla tietoja M700:sta ja lähetetään ne eteenpäin. M700:sta luetaan kolmea eri rekisteriä, koska osoitteet eivät ole peräkkäin joudutaan käyttämään kolmea eri *Modbus Getter* -nodea. Suositeltava tapa olisi käyttää yhtä *Modbus Getter* -nodea ja lukea sillä useampaa osoitetta. *Modbus Getter* palauttaa rekisteri arvon taulukossa (Array). (8)

Osa 3:ssa M700:sta tuleva arvo on kokonaisluku, joten siinä ei ole desimaalierotinta. Tässä käytetään *function*-nodea, jossa M700:lta tullut arvo skaalataan haluttuun yksikköön.

Osa 4:ssä valitaan globaalin muuttujan perusteella, tallennetaanko viesti paikalliseen tietokantaan vai palvelimelle. Formatoidaan msg.payload tallennuspaikan perusteella sellaiseen muotoon, että se voidaan lähettää tietokantaan tallennettavaksi.

Osa 5:ssä tallennetaan viestin sisältö paikalliseen tietokantaan *Raspi*-nodella tai palvelimelle *httpWriteServer_Speed*-nodella. Tallennettaessa palvelimelle kerätään 14–16 viestiä *Puskurointi*-nodessa yhdeksi merkkijonoksi, joka lähetetään *httpWriteServer_xxx*-nodelle. InfluxDB:ssä on HTTP API -rajapinta, joka mahdollistaa viestien lähettämisen InfluxDB:hen käyttäen HTTP-protokollaa. Palvelimelle tallennuksessa käytetään *http request* -nodea, koska yhteyden katketessa *http request* -node ja *influxdb*-node lähettävät virheviestin, jossa on virheen koodi ja alkuperäinen payload. Mutta toisin kuin *http request* -node *influxdb*-node jättää alkuperäisestä payloadista aikaleiman pois. *Catch*-node saa nämä virheviestit kiinni ja lähettää ne tallennettaviksi paikalliseen tietokantaan. Yhteyden katkeamisen havaitsemiseen menee noin 2 minuuttia ja paikallinen

tietokanta laittaa viesteihin oman aikaleiman niiden tallennus hetkellä, jos niissä ei ole muuta aikaleimaa. Tällaisessa tapauksessa tiedot laskostuvat tietokannassa.



Kuva 16. Pää-flow.

5.2.1 OSA 1:n nodet

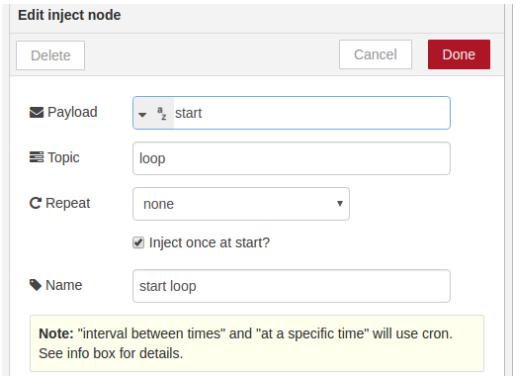
Kuvassa 17 on *Start loop* -noden Edit-välilehti.

- *Payload*-kohtaan valitaan string muuttuja ja kirjoitetaan teksti ”start”.

- *Topic*-kohtaan voidaan kirjoittaa teksti ”loop”, mutta sillä ei ole sovelluksen toiminnan kannalta väliä.

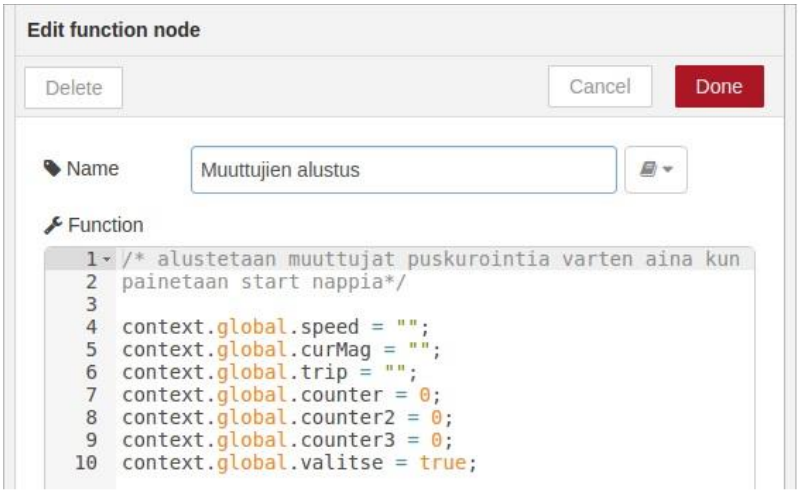
- *Inject once at start* -kohta on aktivoituna, joten node lähettää yhden viestin käynnistyessään.

- *Stop loop* -nodessa *Payload*-kohtaan tulee ”stop” ja *Inject ones at start* ei ole aktivoituna.



Kuva 17. *Start loop* -node.

Kuvassa 18 on esitetty *Muuttujien alustus* -noden sisältämä JavaScript-koodi. *Funktion* -nodessa alustetaan globaalit muuttujat. *context.global.<muuttujan nimi>* -käskyllä voidaan luoda globaaleja muuttujia, jotka näkyvät kaikille kyseisen välilehden nodeille. Tässä nodessa alustettavia muuttujia käytetään puskuroinnissa ja tallennuspaikan valinnassa.



Kuva 18. *Muuttujien alustus* -node.

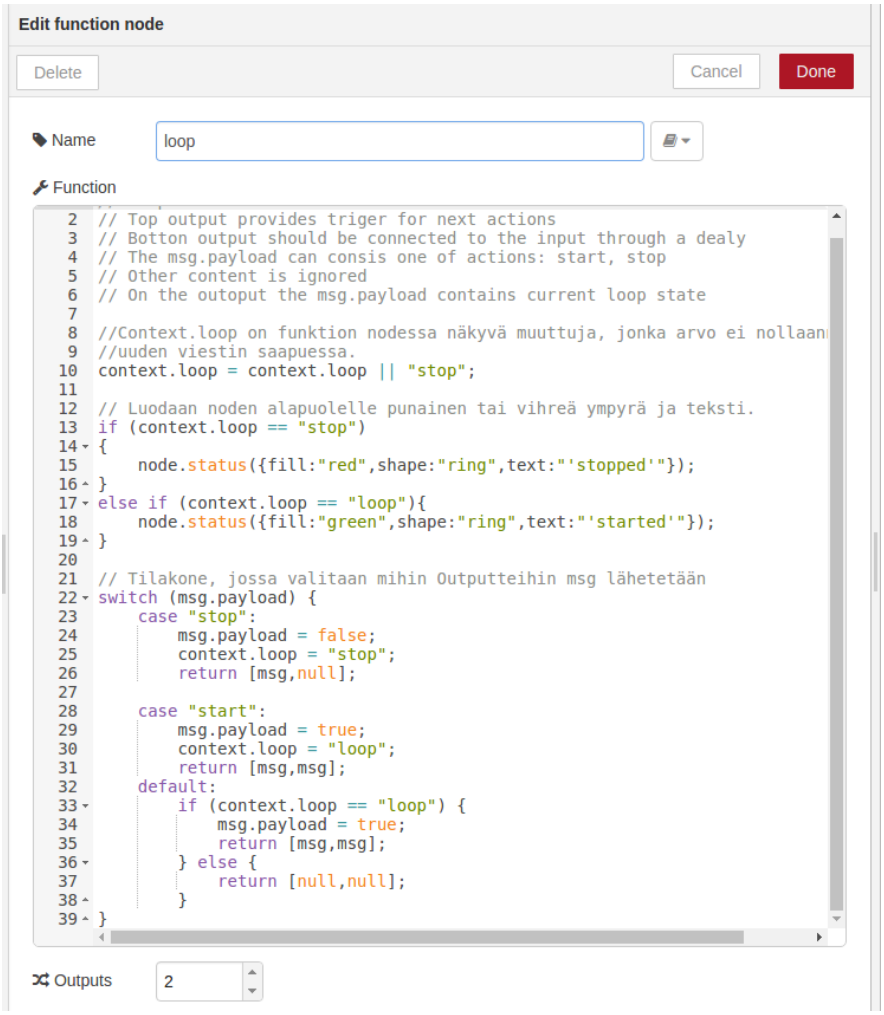
Kuvassa 19 on esitetty *loop*-noden sisältämä JavaScrip-koodi. (7)

Nodessa on kaksi ulostuloa (Outputs) joista toinen johtaa *Modbus Getter* -nodeihin ja toinen *delay*-nodeen.

Rivillä 10 olevalla *context* käskyllä voidaan luoda *function*-nodeissa sisäinen muuttuja, jonka arvo ei nollaannu uuden viestin saapuessa. *context.loop* -muuttujaa käytetään tilakoneessa.

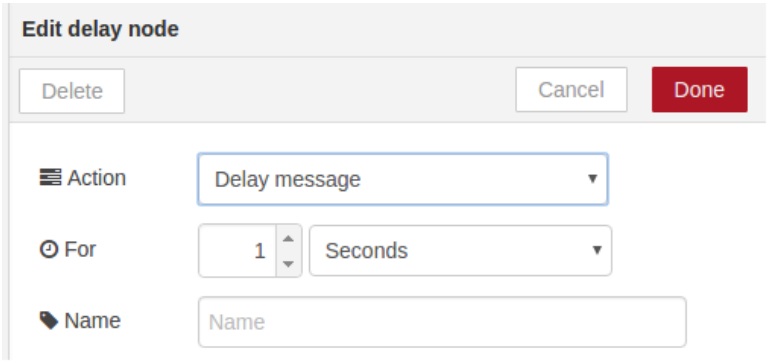
Riviltä 22 alkava *switch case* tilakone lukee msg.payloadiin sisällön ja toimii sen mukaan, mitä payloadissa on. msg.payload voi tässä ohjelmassa sisältää kolmenlaisia viestejä, merkkijonon joka on ”stop” tai ”start” tai boolean arvon ”true”. Boolean arvo ”true” tulee *delay*-nodelta.

Kun tilakoneeseen saapuu viesti jonka payload:ssa on boolean arvo ”true” menee tilakone riville 32. Riippuen sisäisen muuttujan *loop* arvosta lähetetään viesti edelleen kumpaankin ulostuloon tai vaihtoehtoisesti ei kumpaankaan. Kun ulostulo porttiin kaksi ei lähetetä mitään, toistosilmukka pysähtyy.



Kuva 19. *Loop*-node.

Kuvassa 20 on *delay*-noden Edit-välilehti. Välilehden *Action*-kohtaan valitaan *Delay message*, jolloin voidaan valita kuinka kauan node viivästyttää viestiä. Viiveellä säädetään viestien lähetys nopeutta, joka on myös samalla nopeus, jolla M700-drivelta luetaan tietoja.



Kuva 20. *Delay*-node.

5.2.2 OSA 2:n nodet

Kuvassa 21 on *Modbus Getter* -noden Edit-välilehti.

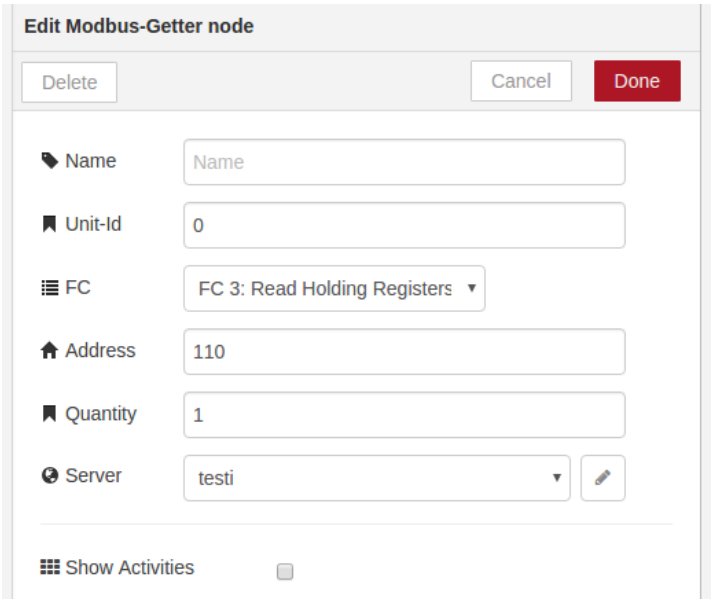
- *Unit-Id*-kohtaan tulee Modbus-osoite, joka tässä tapauksessa on 0. M700:ssa on kaksi palvelinta. Toisen palvelimen Modbus-osoite on 4. Palvelimessa 4 sijaitsevat verkkoyhteyksiin liittyvät parametrit.

- *FC*-kohtaan valitaan *Read Holding Registers*.

- *Address*-kohtaan tulee rekisterin numero, jota halutaan lukea.

- *Quantity*-kohdassa määritellään, kuinka monta rekisteriä halutaan lukea *Address*-kohdan rekisteristä eteenpäin.

- *Server*-kohtaan valitaan palvelin, johon ollaan yhteydessä.



Kuva 21. *Modbus Getter* -node.

Kuvassa 22 on *Modbus Getter* -noden server asetusten -välilehti. (8)

- *Type*-kohtaan valitaan TCP. *Host*-kohtaan laitetaan laitteen, jossa Modbus-palvelin sijaitsee, IP-osoite. Tässä tapauksessa M700:n IP-osoite.

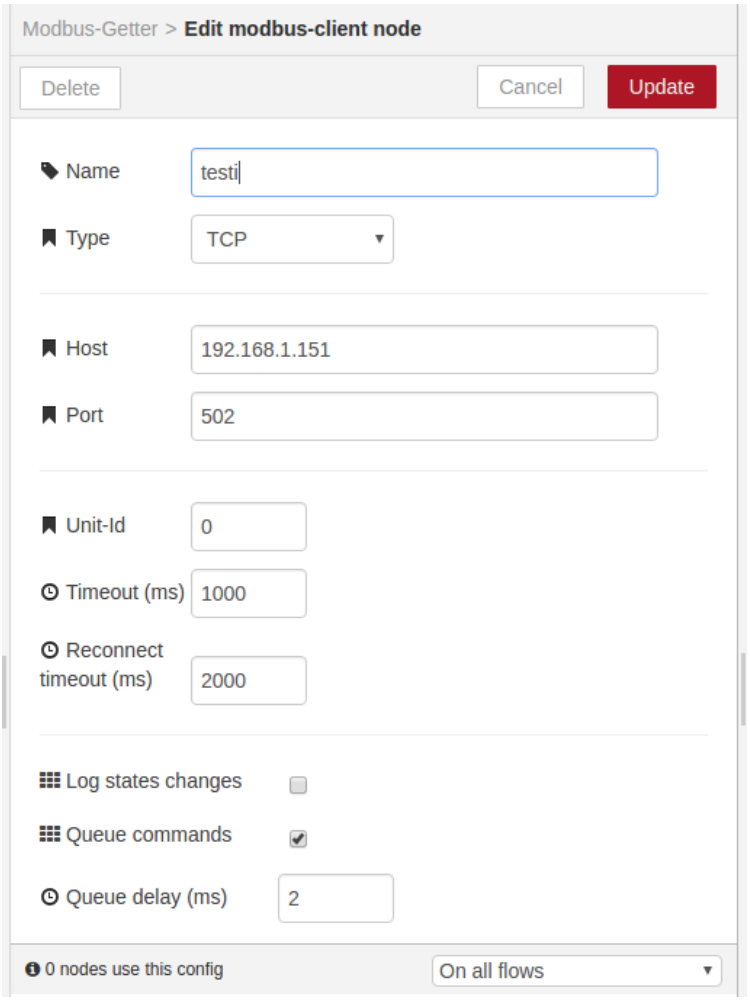
- *Port*-kohtaan tulee Modbus TCP -portti, joka yleensä on 502.

- *Unit-Id*-kohtaan tulee Modbus-osoite.

- *Timeout*-kohdassa määritellään, kuinka kauan vastausta odotellaan ja *Reconnect*-kohdassa määritellään aika yhteyden katkeamisen jälkeen, jolloin yritetään uudelleen yhdistää.

- *Queue commands* pitää olla aktivoituna, jolloin samaa aikaa tulevat viestit voidaan laittaa jonoon. *Flow*'ssa on 3 Modbus Getter -nodea, jotka lähettävät samanaikaisesti. Jos *Queue commands* ei ole aktivoituna, samanaikaisesti M700:ssa olevalle Modbus-palvelimelle lähtevistä viesteistä vain yksi menee läpi.

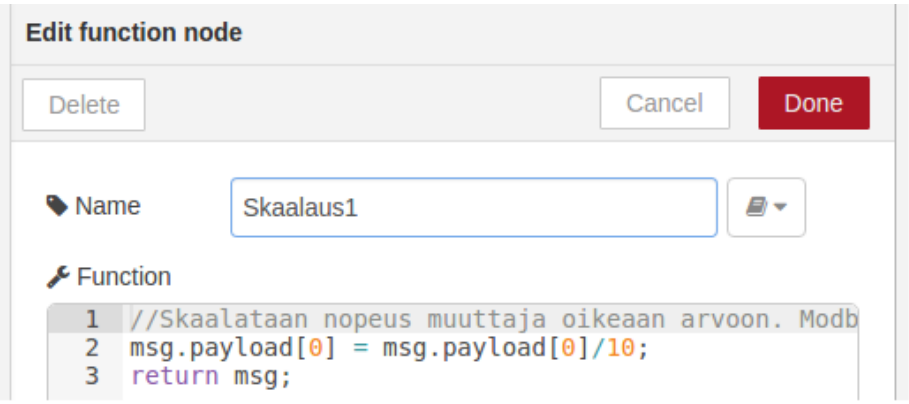
- *Queue delay* -kohdassa määritellään jonossa olevien viestien välille laitettava viive.



Kuva 22. *Modbus Getter* -noden server asetusten -välilehti.

5.2.3 OSA 3:n nodet

Kuvassa 23 on esitetty *Skaalaus*-noden JavaScript-koodi. Nodessa skaalataan M700:sta saatu nopeusarvo. Nopeusarvossa on yksi desimaali, joten jakamalla arvo 10:llä saadaan tulokseksi kierroksia minuutissa. *Modbus*-nodelta tulevan viestin payload on taulukko, vaikka se sisältää vain yhden arvon. Rivillä 2 msg.payloadin ensimmäinen alkio jaetaan 10:llä ja sijoitetaan takaisin msg.payloadin ensimmäiseen alkioon.



Kuva 23. *Skaalaus*-node.

5.2.4 OSA 4:n nodet

Kuvassa 24 on esitetty *Select 1* -noden JavaScript-koodi. Rivillä 1 luodaan taulukko, joka sisältää objektin, jossa on kaksi kenttää value ja time. Rivillä 2 tämä taulukko tallennetaan msg.payloadiin.

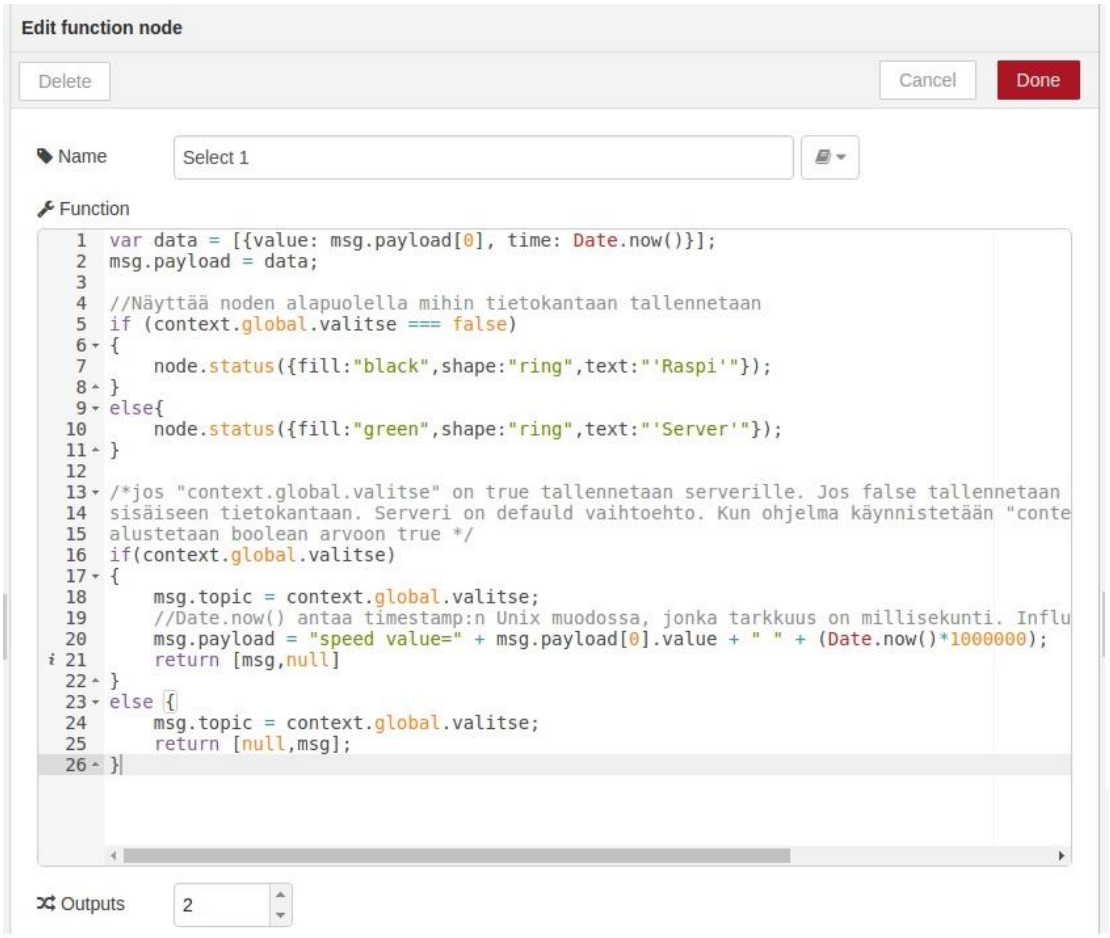
Rivillä 2 msg.payloadiin tallennettava objekti on muotoiltu niin, että se voidaan semmoisenaan lähettää *influxdb* -nodelle tallennettavaksi InfluxDB-tietokantaan.

Rivillä 16:sta alkavassa *If*-lauseessa tarkistetaan, onko globaalin boolean muuttujan context.*global.valitse*-arvo ”true”. Arvon ollessa ”true” suoritetaan rivit 18–21.

Payloadin sisältö täytyy muuttaa merkkijonoksi, koska tallennus palvelimen tietokantaan tapahtuu http-viestinä.

Rivillä 20 luodaan InfluxDB-tietokannan määritelmien mukainen merkkijono yhden pisteen tallentamista varten. Ensimmäisenä annetaan *measurement*in nimi, jonka jälkeen tulee fieldkey tässä tapauksessa *value* ja viimeisenä aikaleima (*Date.now()*). Aikaleima annetaan Unix-aikana. InfluxDB:ssä käytetään nanosekunnin tarkkuutta, mutta Node-RED käyttää vain millisekunnin tarkkuutta. Tästä syystä aikaleima on kerrottava miljoonalla, jotta saadaan tarvittava määrä numeroita nanosekunnin tarkkuutta varten.

Rivillä 21 lähetetään msg *Select 1* -noden lähtö 1:een, lähtö 2:een ei lähetetä mitään (null). Samalla luodaan msg-objektiin *topic*, johon tallennetaan ”valitse”-muuttujan arvo. *Topic* luodaan virheenkorjausta varten.



Kuva 24. *Select 1* -node.

5.2.5 OSA 5:n nodet

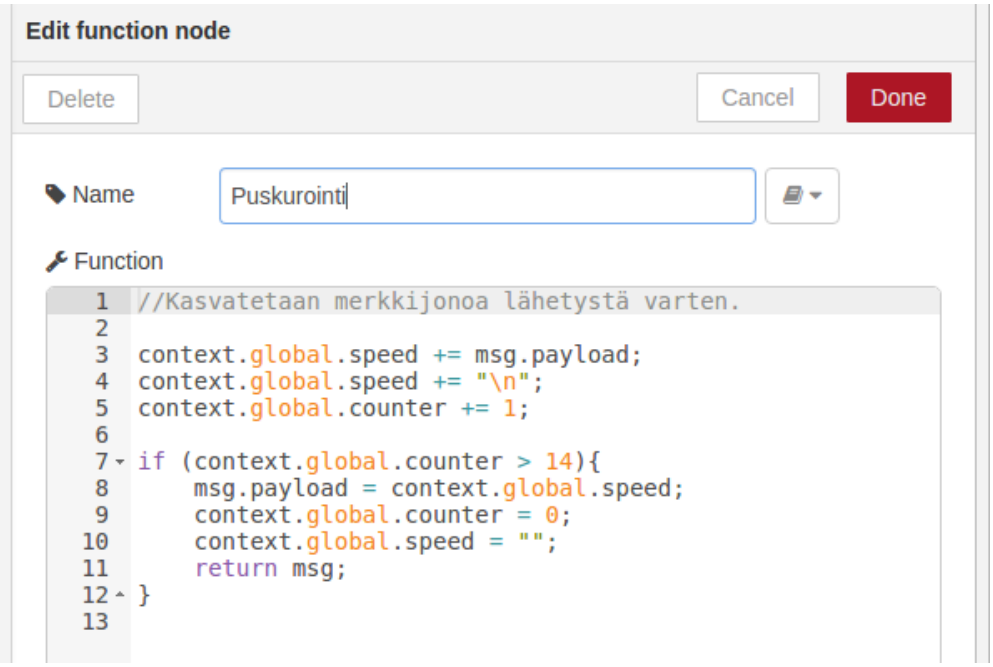
Kuvassa 25 on esitetty *Puskurointi*-noden JavaScript-koodi. Globaaliin muuttujaan *context.global.speed*, on sen luonnin yhteydessä *Muuttujien alustus* -nodessa sijoitettu tyhjä merkkijono. Rivillä 3 tähän muuttujaan lisätään jokaisen viestin saapuessa viestin msg.payloadin sisältämä merkkijono.

Rivillä 4 *speed*-muuttujaan lisätään vielä merkkit ”\n”, jonka InfluxDB tulkitsee komennon lopuksi. Tällä tavalla voidaan yhteen merkkijonoon asettaa useita kirjoitus komentoja.

Rivillä 5 kasvatetaan *Muuttujien alustus* -nodessa luodun globaalin muuttujan *counter* arvoa yhdellä. Muuttujaa käytetään *if*-lauseessa.

Rivillä 7 määritellään, kuinka monta tallennus käskyä merkkijonoon laitetaan ennen kuin se lähetetään eteenpäin. Flow'ssa on kolme *Puskurointi*-nodea, niissä on syytä muodostaa eripituisia merkkijonoja, jotta kaikki kolme nodea eivät lähettäisi http-viestiä samaan aikaan InfluxDB-tietokantaan.

Merkkijonojen pituuden muuttaminen tapahtuu rivillä 7 *if*-lauseen vertailuosassa, jossa verrataan *context.global.counter*-muuttujaa lukuun 14.



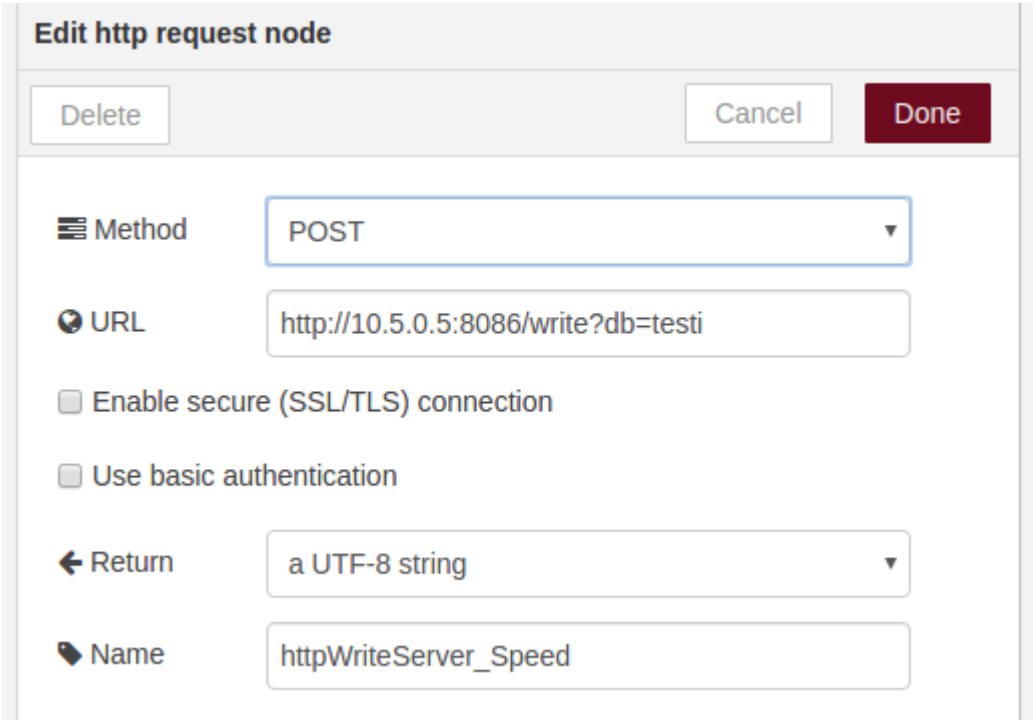
Kuva 25. *Puskurointi*-node.

Kuvassa 26 on *httpWriteServer*-noden Edit-välilehti.

- *Medhod*-kohtaan valitaan POST.

- *URL*-kohdassa on palvelimen, jossa on InfluxDB, IP-osoite sekä porttinumero. IP-osoitteen ja portin jälkeen tulee kirjoituskäsky ja tietokannan nimi, tässä tapauksessa *testi*.

- *Return*-kohta voidaan jättää default-arvoon.

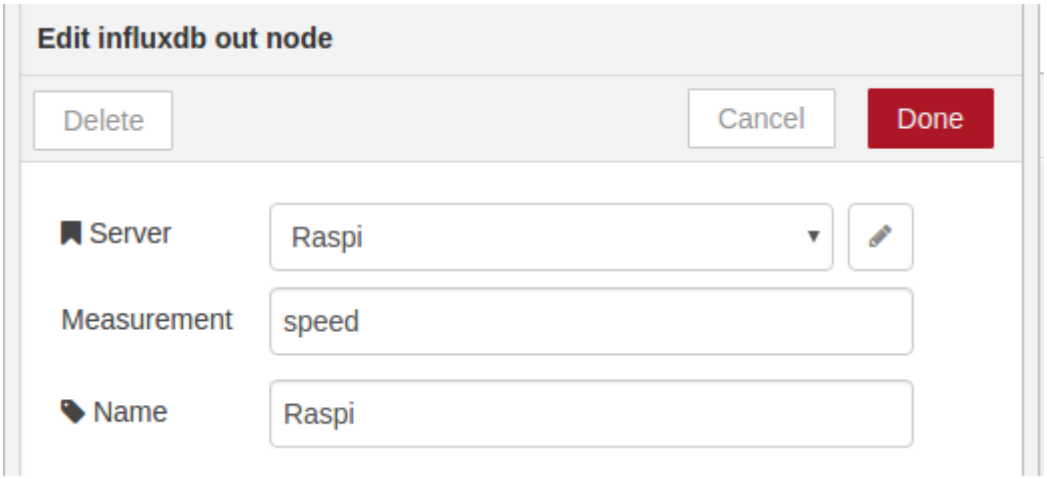


Kuva 26. *httpWriteServer_Speed*-node.

Kuvassa 27 on *Raspi*-noden Edit-välilehti.

- *Server*-kohtaan valitaan tietokanta, johon halutaan tallentaa.

- *Measurement*-kohtaan tulee measurementin nimi, jonka alle pisteet halutaan tallentaa.



Kuva 27. *Raspi*-node

Kuvassa 28 on *Raspi*-noden Server konfigurointi -välilehti.

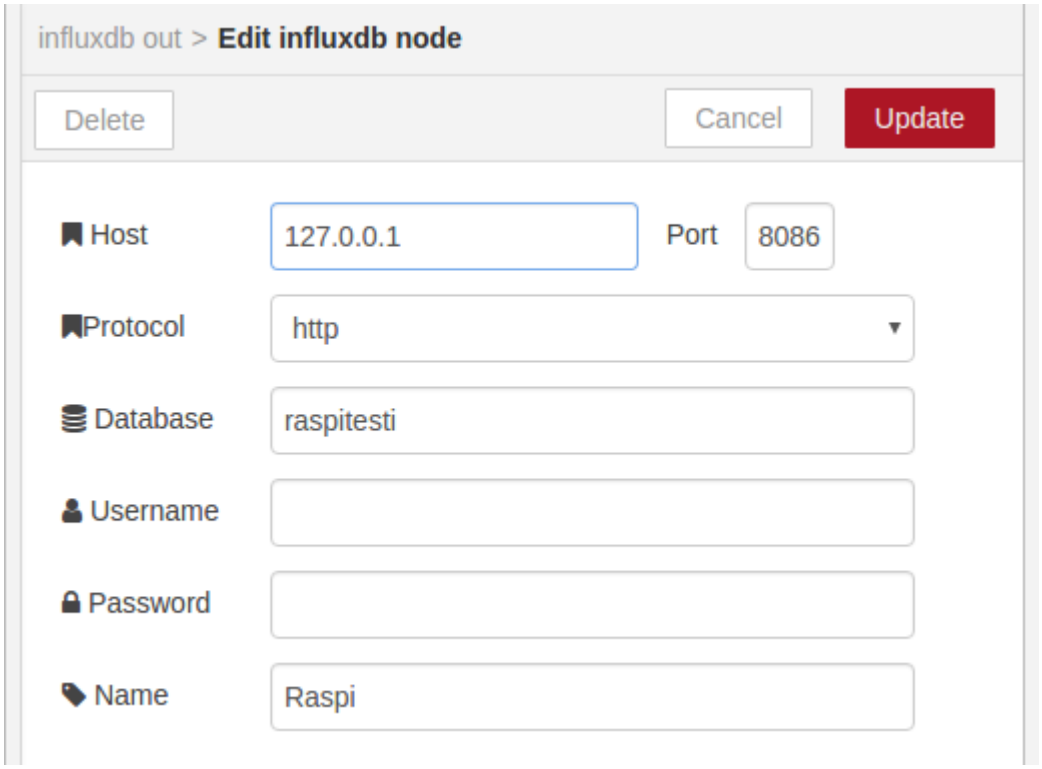
- *Host*-kohtaan tulee InfluxDB-palvelimen IP-osoite. InfluxDB:n pyöriessä localhostina käytetään IP-osoitetta 127.0.0.1.

- *Port*-kohtaan tulee 8086, joka on InfluxDB:n käyttämä portti.

- *Protocol*-kohtaan valitaan *http*.

- *Database*-kohtaan tulee tietokannan nimi, johon halutaan tallentaa.

- *Username*- ja *Password*-kohtiin ei tule mitään. Tunnistautuminen on perusasetuksilla pois päältä InfluxDB-tietokannassa.



Kuva 28. *Raspi*-noden Server-välilehti.

5.3 Datankerääjän ohjelmakoodin toinen osa

Kuvassa 19 on Node-RED-koodin toinen osa. Tässä osassa hoidetaan yhteyden katkeamisesta aiheutuvat toimenpiteet. Toinen flow on toiminnassa vain silloin kun yhteyttä InfluxDB-palvelimeen ei ole.

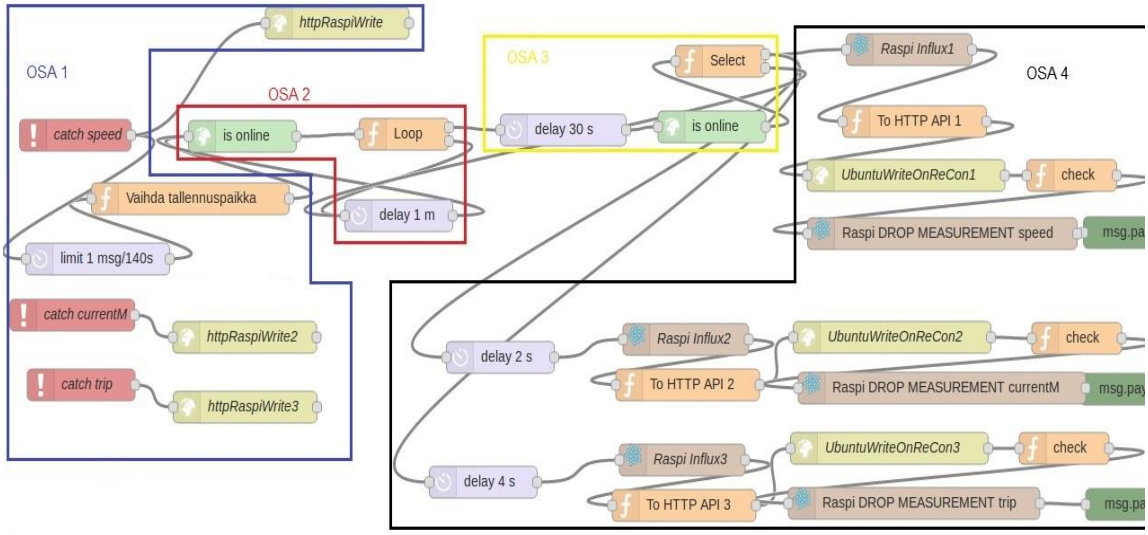
Pää-flow'n *http request* -nodelta, menee noin kaksi minuuttia huomata, ettei sen lähettämä viesti ole mennyt perille. Tämän jälkeen Pää-flow'n *httpWriteServer_Speed*-node lähettää virheviestin *catch*-nodelle. Virheviestin saapuessa *catch speed* -nodelle lähettää se virheviestin payloadissa olevan alkuperäisen viestin *httpRaspiWrite*-nodelle, joka tallentaa sen raspberryllä olevaan InfluxDB-tietokantaan. Viesti lähtee myös *limit1 msg/140s* -nodelle (*delay*-node), tämä node päästää vain yhden viestin läpi 140 sekunnissa. *Vaihda tallenuspaikka* -noden saadessa viestin vaihtaa se tallennuspaikan paikalliseksi, jolloin Pää-flow alkaa tallentaa pisteitä Raspberryllä olevaan tietokantaa. 140 sekunnin viestien läpipääsyrajoitin tarvitaan, koska yhteyden katkeamisen havaitsemiseen menee kaksi minuuttia. Kaikista Pää-flow'n viimeisen kahden minuutin aikana lähetetyistä viesteistä saapuu virheviesti *catch*-nodelle. Ilman viestien läpipääsyn määrän rajoitinta *osa 2*:n silmukkaa päätyisi kiertämään useita viestejä. Jokaiselle Pää-flow'ssa olevalle *httpWriteServer_xxx*-nodelle on oma *catch*-node.

Osa 2:ssa on silmukka, jota kierretään, kunnes InfluxDB-palvelin vastaa *is online* -noden pingiin. Tämä silmukka toimii samalla tavalla kuin Pää-flow'n silmukka.

Osa 3:ssa on 30 sekunnin viive, jonka jälkeen pingataan palvelinta uudelleen. Jos palvelin ei vastaa, palautetaan viesti *osa 2*:n silmukkaan. Palvelimen vastatessa katsotaan yhteyden toimivan ja viesti lähetetään eteenpäin.

Osa 4:ssä tehdään *Raspi influx1* -nodella kysely paikalliseen tietokantaan, jolla saadaan pisteet, jotka on tallennettu yhteyden ollessa poikki. Kyselyn tulokset muokataan *To http API 1* -nodessa merkkijonoksi, joka lähetetään *UbuntuWriteOnReCon*1-nodelle. *UbuntuWriteOnReCon1*-node lähettää merkkijonon InfluxDB-palvelimenlle tallennettavaksi. Jos tallennus onnistuu, InfluxDB-palvelin lähettää siitä vahvistukseksi tyhjän merkkijonon. *Check*-node lähettää valvelimelta tulevan viestin eteenpäin, jos payloadissa on tyhjä merkkijono. *Raspi DROP MEASUREMENT* -node tuhoaa measurementin, jonka pisteet lähetettiin palvelimelle. Kysely, tallennus ja tuhoaminen tehdään kaikille measurementeille, joita tallennetaan paikalliseen tietokantaan yhteyden ollessa poikki. Eri measurementeillä kysely, tallennus ja tuhoaminen toimenpiteiden välillä on oltava vähintään

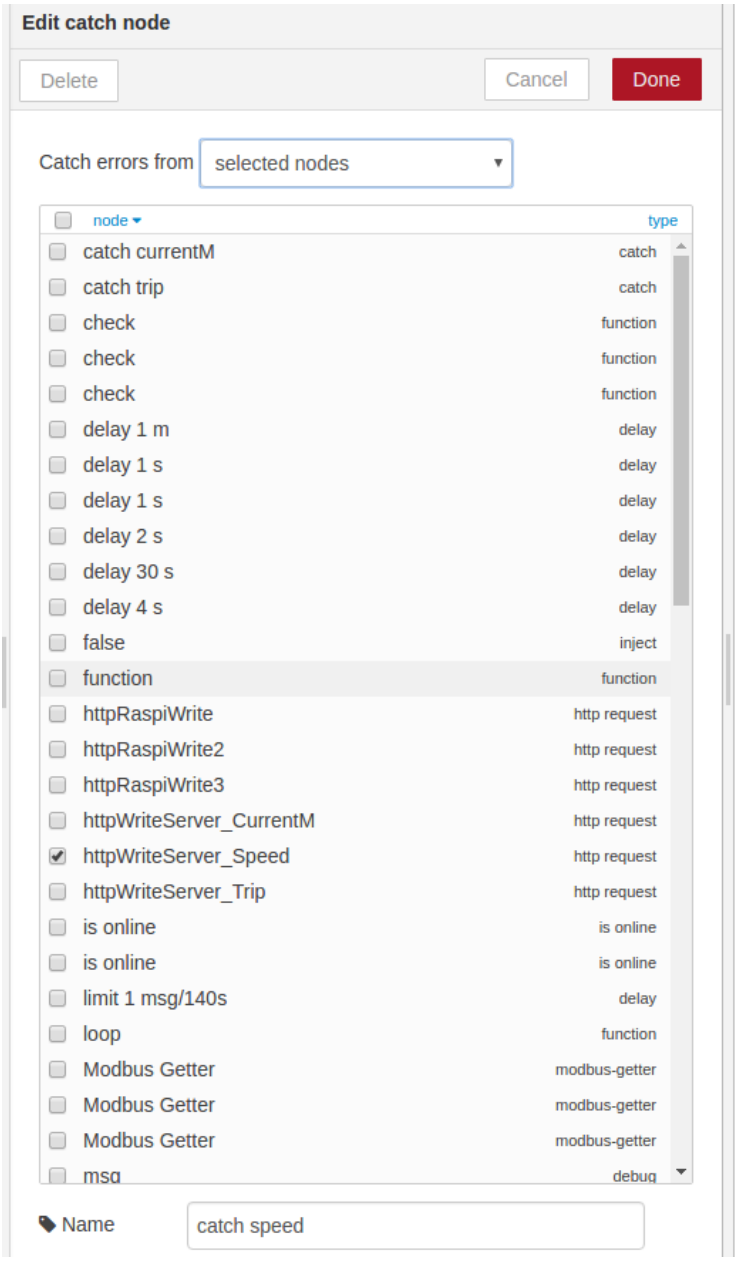
2 sekuntia. Liian pieni intervalli toimenpiteiden välillä aiheuttaa *influxDB*-nodessa virheen, joka johtuu luultavasti liian monesta IOPS:sta sekunnissa.



Kuva 29. Yhteyden katkeamisen käsittely flow

5.3.1 OSA 1:n nodet

Kuvassa 30 on *catch*-noden Edit-välilehti. *Catch*-nodessa valitaan listalta se node, jonka virheviesti halutaan saada tähän nodeen.



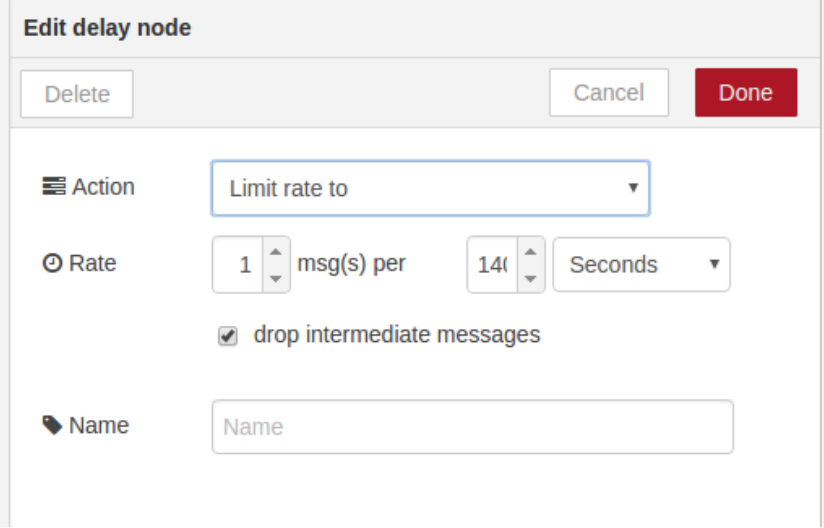
Kuva 30. *catch speed* -node.

Kuvassa 31 on *Limit msg/140s* -noden Edit-välilehti.

- Action-kohtaan valitaan Limit rate to.

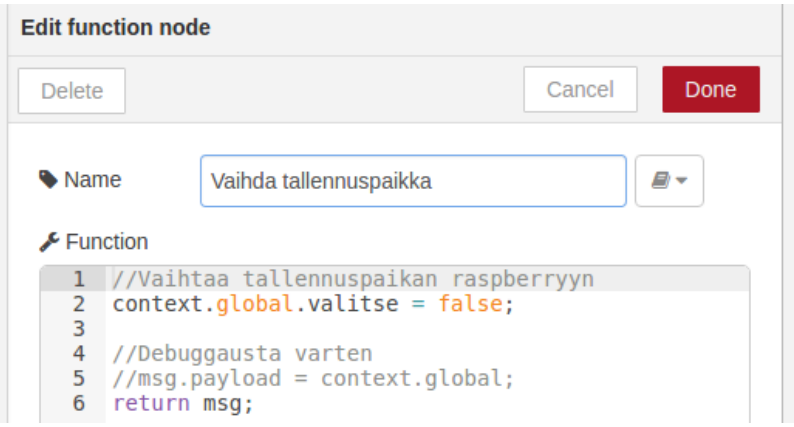
- *Rate*-kohtaan tulee 1 viesti 140 sekunnissa.

- Aktivoidaan-kohta *drop intermediate messages*. Tällöin noden päästäessä viestinläpi lähtee 140 sekunnin laskuri päälle, jonka aikana node hylkää kaikki saapuvat viestit. Jos tätä kohtaa ei aktivoida, node alkaa puskuroimaan siihen tulevia viestejä ja päästää niitä läpi 140 sekunnin välein.



Kuva 31. *Limit 1 msg/140s* -node.

Kuvass 32 on esitetty *Vaihda tallennuspaikka* -noden JavaScript-koodi. Rivillä 2 muutetaan globaalin muuttujan *valitse* boolean arvo *false*:ksi. Tällöin Pää-flow'n *Select*-nodet alkavat lähettämään viestejä Raspberryn paikalliseen tietokantaan tallennettaviksi.

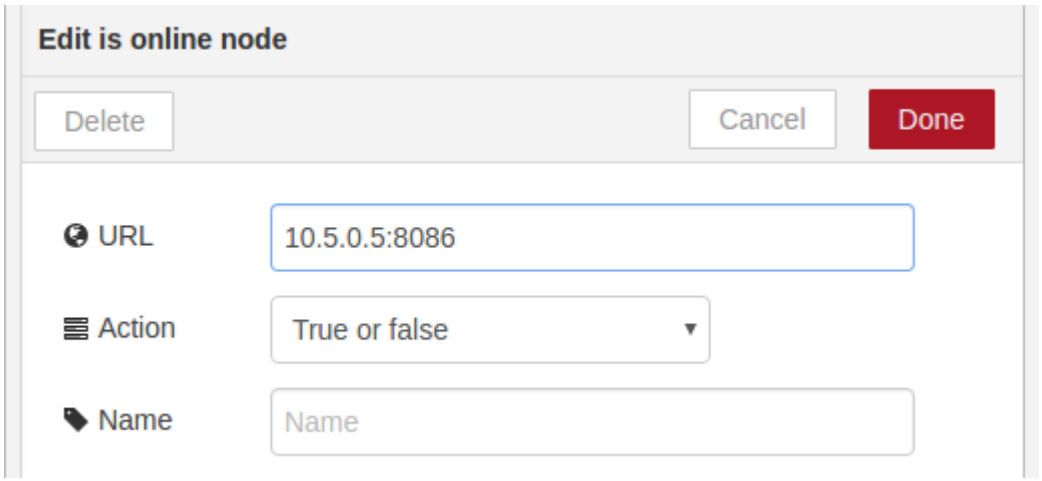


Kuva 32. *Vaihda tallennuspaikka* -node.

httpRaspiWrite-nodet toimivat samalla tavalla kuin Pää-flow'ssa olevat. URL-kentän IP-osoitteeksi tulee 127.0.0.1.

5.3.2 OSA 2:n nodet

Kuvassa 33 on *is online* -noden Edit-välilehti. *Is online* -node pingaa URL-kentässä olevaa osoitetta. Jos pingiin vastataan lähettää node eteenpäin viestin, jonka payloadissa on boolean arvo ”true”. Jos pingiin ei vastata, viestin payloadissa on boolean arvo ”false”.

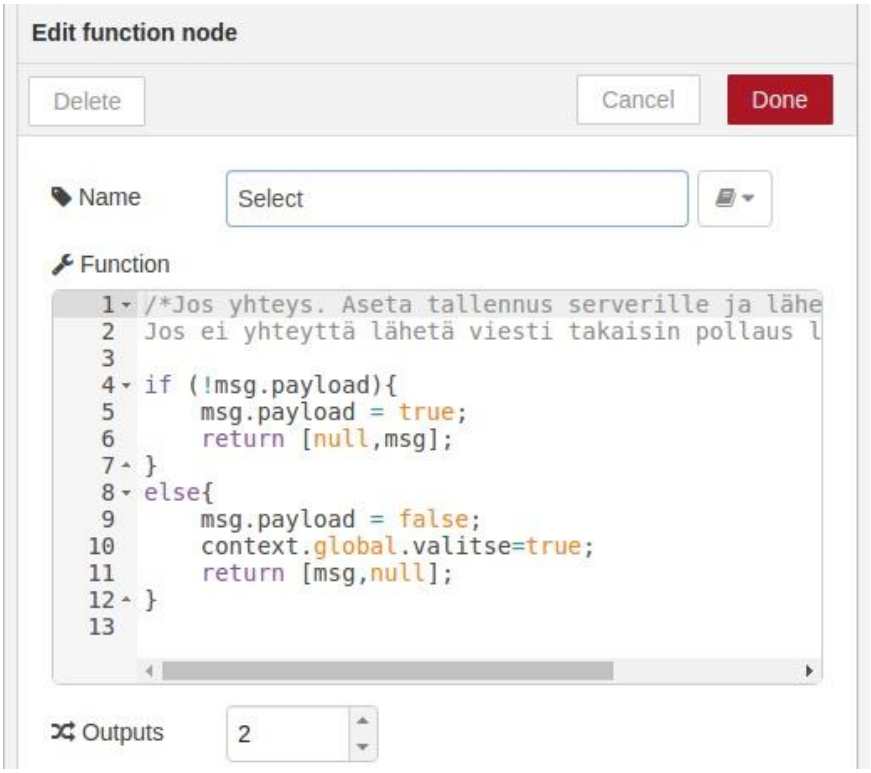


Kuva 33. *is online* -node.

Osa 2:den silmukka (loop) toimii samalla tavalla kuin Pää-flow’n.

5.3.3 OSA 3:n nodet

Kuvassa 34 on esitetty *Select*-noden JavaScript-koodi. Riviltä 4 alkaa *If*-lause. Jos saapuvan viestin msg.payload on ”true” *If-Else*-lause lähettää viestin osaan 4. Jos msg.payload on ”false”, viesti palautetaan osan 2 silmukkaan. Riveillä 6 ja 11 määritellään, mitä kuhunkin ulostuloon lähetetään.



Kuva 34. *Select*-node.

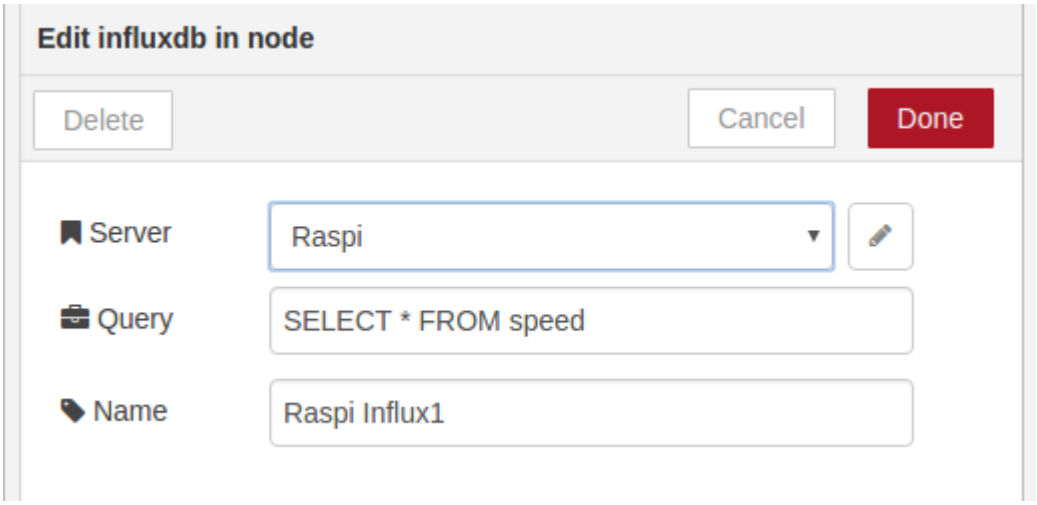
5.3.4 OSA 4:n nodet

Kuvassa 35 on *Raspi influx1* -noden Edit-välilehti.

- *Server*-kohtaan valitaan tietokanta, johon halutaan tallentaa.

- *Query*-kohtaan voidaan kirjoittaa perus SQL-syntaksia. Tässä tapauksessa kysely hakee kaikki speed measurementin alla olevat pisteet.

Node palauttaa msg.payloadissa taulukon, jonka sisällä on olio-taulukon. Taulukossa on niin monta oliota kuin measurementissa on pisteteitä. Olio koostuu pisteen aikaleimasta sekä Tag- ja Field-kentistä ja niiden arvoista. Datankerääjän tallentamissa pisteissä on vain aikaleima ja yksi Field-kenttä.

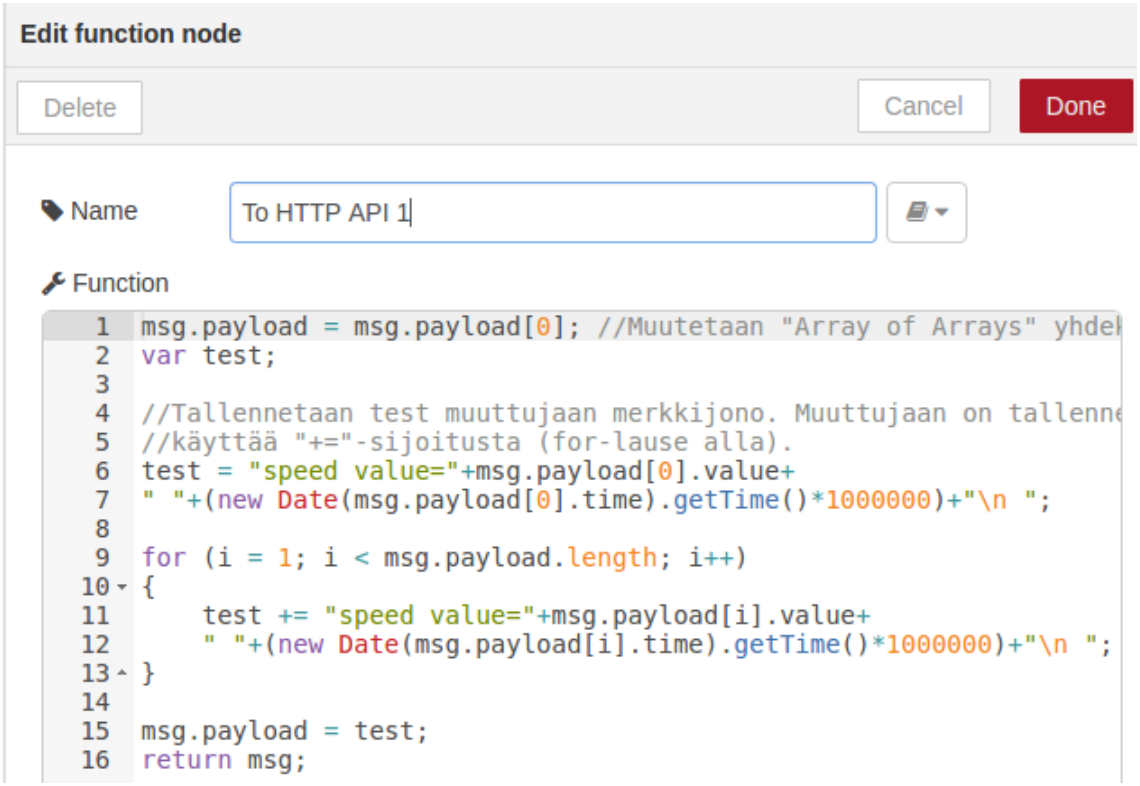


Kuva 35. *Raspi influx1* -node. Nodella voidaan tehdä kyselyjä tietokantaan.

Kuvassa 36 on esitetty *To http API 1* -noden JavaScrip-koodi. Tietokanta kysely palauttaa taulukon jonka sisällä on taulukko. Ensimmäisellä rivillä sijoitetaan taulukon sisällä oleva olio-taulukko msg.payloadiin, jotta payloadia olisi helpompi käsitellä.

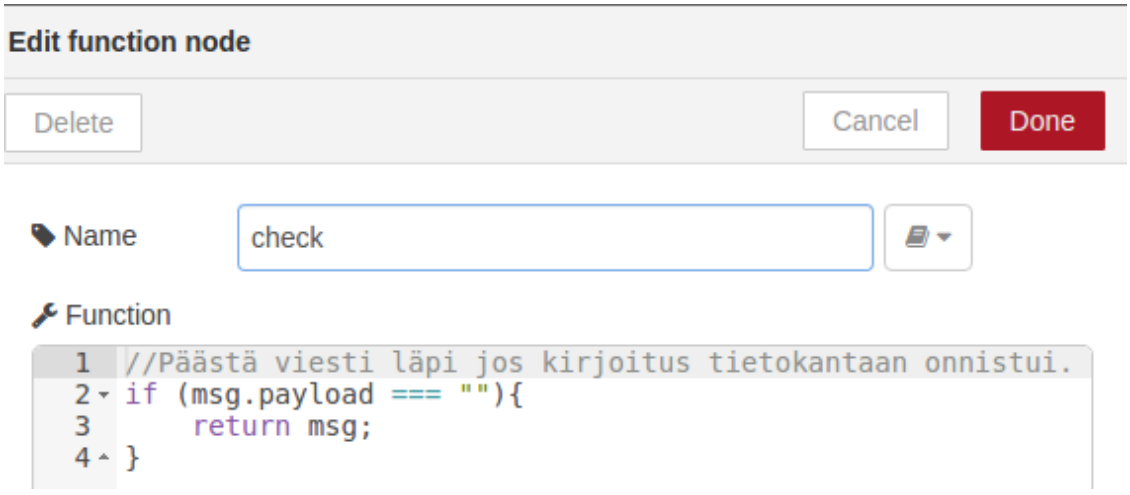
Rivillä 6 sijoitetaan-*test* muuttujaan taulukon ensimmäinen olio. Koska muuttuja on tyhjä, on käytettävä sijoitus operaatiota. Kun muuttujassa on merkkijono, voidaan riviltä 9 alkavassa *for*-lauseessa käyttää lisäys operaatiota.

Riveillä 6–13 luodaan merkkijono, jotta tiedot voidaan tallentaa palvelimella olevaan InfluxDB-tietokantaan http-viestillä. Rivillä 15 *for*-lausessa luotu merkkijono sijoitetaan msg.payloadiin ja lähetään eteenpäin.



Kuva 36. *To HTTP API 1* -node.

Kuvassa 37 on esitetty *check*-noden JavaScript-koodi. Rivillä 2 tarkastetaan, että edellinen node lähetti tyhjän merkkijonon. Tyhjä merkkijono tarkoittaa tallennuksen onnistumista, jos tulee jokin muu merkkijono, flow pysähtyy tähän. Tällöin seuraava node ei tuhoa Measurementia paikallisesta tietokannasta.

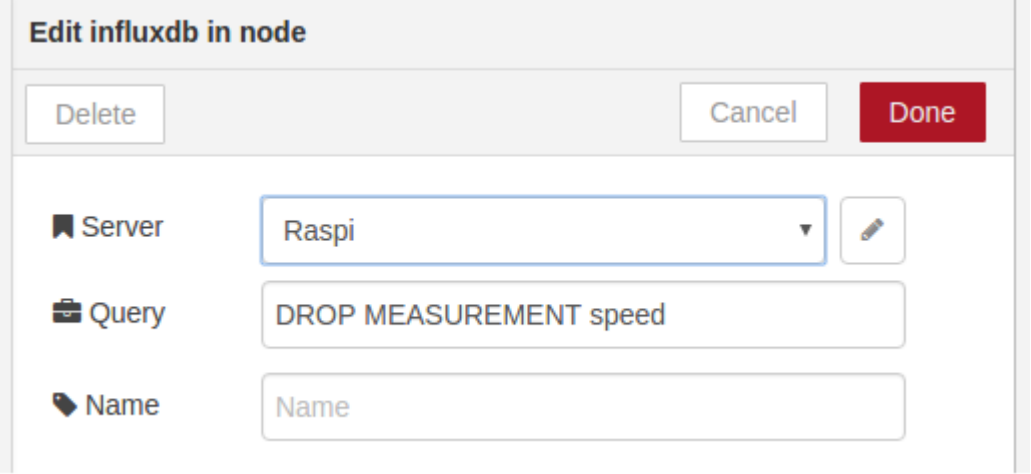


Kuva 37. *Check*-node.

Kuvassa 38 on *Raspi DROP MEASUREMENT* -noden Edit-välilehti.

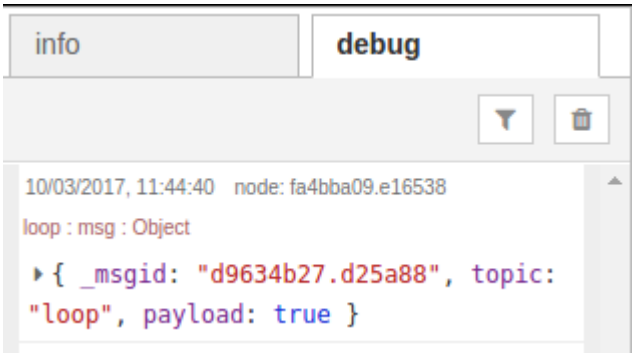
- *Server*-kohtaan valitaan palvelin, johon halutaan ottaa yhteys.

- *Query*-kohdan “DROP MEASUREMENT speed”-käsky poistaa kaikki pisteet speed measurementin alta, sekä speed measurementin.



Kuva 38. *Raspi DROP MEASUREMENT speed* -node.

Kuvassa 39 on *debug*-noden lähettämän viestin sisältö debug-välilehdellä. Flow’n viimeinen node on *debug*-node, jota voidaan käyttää koodin testauksessa ja virheiden löytämisessä. *Debug*-nodessa voidaan valita näyttääkö se koko msg-objektin sisällön vai vain msg.payloadin sisällön.



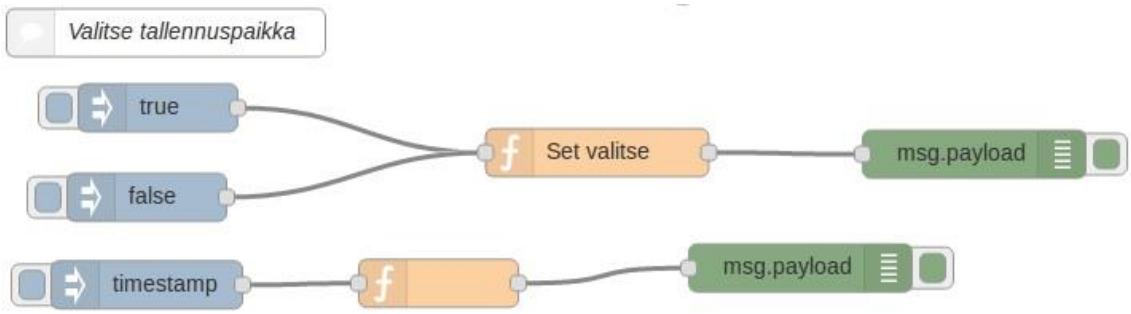
Kuva 39. *Debug*-node kirjoittaa msg:n sisällön debug-välilehdelle.

5.4 Datankerääjän ohjelmakoodin kolmas osa

Kolmannessa osassa on kaksi flow’ta. Ylemmällä flow’lla voidaan, manuaalisesti valita tallennetaanko palvelimen tietokantaan vai paikalliseen tietokantaan. *true*- ja *false*-nodet

lähettävät boolean arvon, joka *Set valitse* -nodessa sijoitetaan globaaliin muuttujaan valitse.

Alempi flow lähetää aikaleiman, jonka arvo on Unix-muodossa. Function-node muuttaa tämän arvon helpommin ymmärrettäväksi ja *debug*-node näyttää sen debug välilehdellä. Tällä voidaan tarkastaa, onko Raspberryn kello oikeassa.



Kuva 40. Kolmas Flow on manuaalista ohjausta ja aikaleiman tarkistamista varten.

6 Grafanan käyttöliittymä

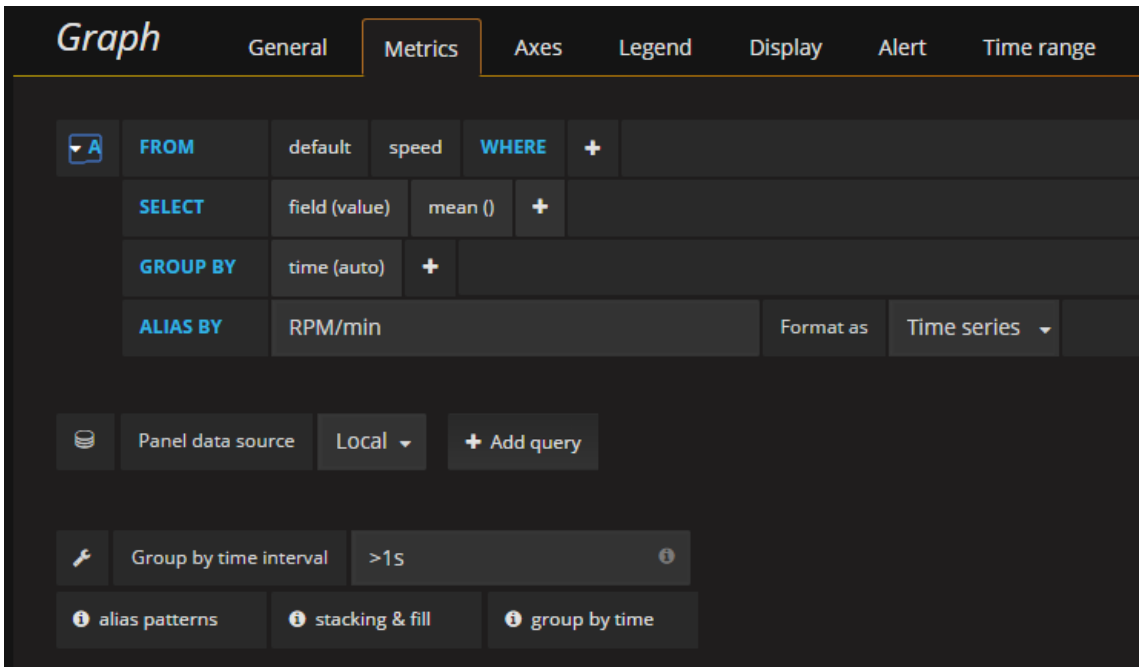
Grafanalla tehtiin kenttätestejä varten käyttöliittymä, jossa näkyy kolme arvoa (Kuva 41): nopeus kierroksia minuutissa, virta-arvo ampeereissa ja Trip-arvo. Käyttöliittymässä on mahdollista valita valikosta eri aikavälejä, joilta näytetään tallennetut arvot. Hiirellä pystyy valitsemaan kuvaajasta aikavälin, johon haluaa zoomata. Kun zoomauksen tekee yhdessä kuvaajassa, käyttöliittymä zoomaa automaattisesti kaikki kuvaajat samalle aikavälille.

Kuva 41. Kuva Grafanalla tehdystä käyttöliittymästä, jossa näkyy tallennetut tiedot viimeisen 7 päivän ajalta.

Asiakkaalle annettiin Viewer-oikeudet käyttöliittymään, mikä tarkoittaa, ettei asiakas pysty muokkaamaan kuvaajien asetuksia.

6.1 SQL-kyselyn teko kuvaajan asetuksissa

Kuvassa 42 on kuva kuvaajan asetuksista metrics-välilehdestä, jossa laitetaan parametrit SQL-kyselyä varten. SQL-kyselyyn on tärkeää laittaa GROUP BY ajan perusteella ja toimimaan automaattisesti (auto). *Panel data source* -kohtaan valitaan tietokanta, jota halutaan käyttää. *Group by time interval* -kohtaan asetetaan 1 sekunti. Jos tämä kohta jätetään auto tilaan, lähelle zoomattaessa kuvaajaan ei piirry mitään.



Kuva 42. Kuvaajan Edit-valikon Metrics-välilehti.

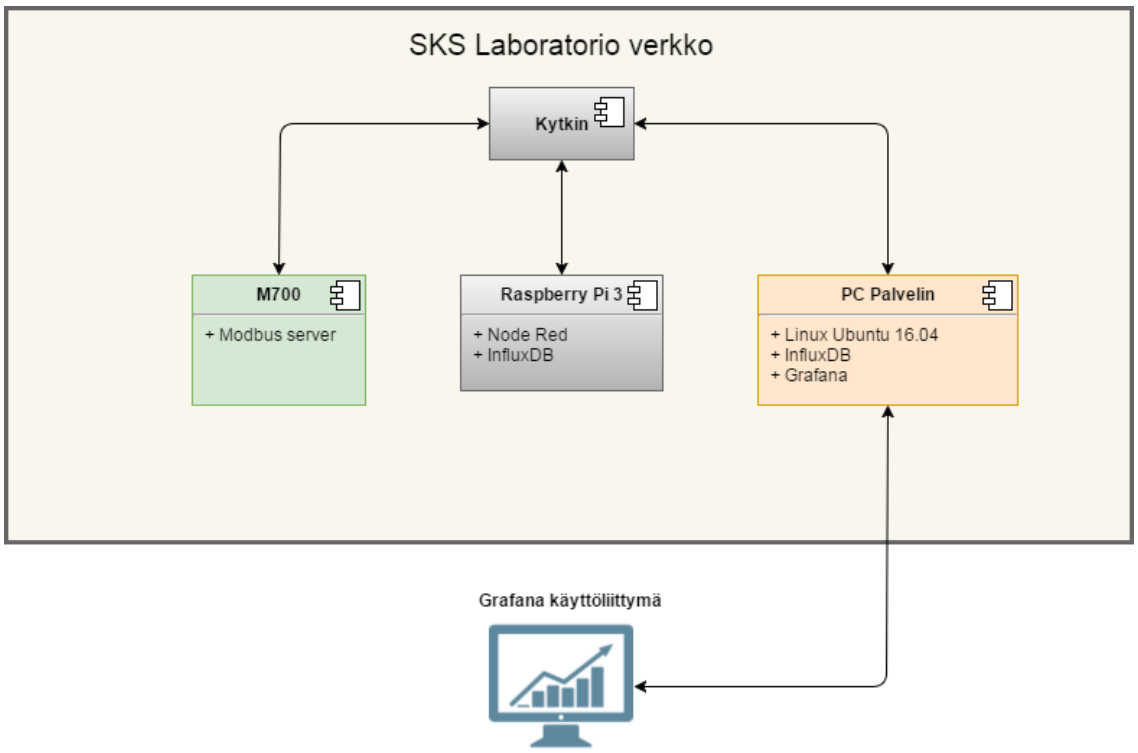
7 Testaus

7.1 SKS-laboratoriotestit

Tiedonkeruu ja etävalvontajärjestelmän testikytkennät rakennettiin aluksi SKS:n laboratorioverkkoon (Kuva 43). Laboratorioverkossa pystyttiin testaamaan tiedonsiirtoa Datankerääjän ja palvelimen välillä, sekä Datankerääjän ja M700:n välistä Modbus TCP -kommunikointia.

Modbus-väylän lukunopeustesteissä kävi ilmi, etteivät pisteet tallennu tietokantaan tasaisin väliajoin. Pisteiden tallennuksessa tulee useasti 10 millisekunnin viiveitä. Välillä viive voi olla 100 millisekuntia. Ajattelin tämän johtuvan Raspberryn Piin raudasta, mutta samanlaisia viiveitä esiintyi myös PC:llä käytetyssä Node-RED-ohjelmassa. Viiveiden syy ei koskaan selvinnyt, mutta viiveet eivät johdu Raspberry Piin raudasta. Välillä tapahtuvat viiveet voivat olla ongelmallisia, jos halutaan tallentaa pisteitä tarkasti pienillä aikaväleillä. Puhutaan alle 500 millisekunnin tallennus väleistä. *Modbus Getter* -noden maksimilukunopeutta testattiin niin, että oli vain yksi Flow, jossa oli *inject*-node, *Modbus Getter* -node, *Influxdb*-node ja *debug*-node. Lukunopeus säädettiin laittamalla *Inject* -node lähettämään viestejä tietyllä nopeudella. Tällöin alle 40 millisekunnin nopeudella Node-RED ohjelma web-käyttöliittymä jumittui pahasti, eikä sitä pystynyt enää käyttämään. Tällaisessa tapauksessa flow'ta ei saanut enää pysäytettyä, joten Node-RED-ohjelma oli pysäytettävä komentoriviltä ja käynnistettävä uudelleen. Samat tulokset saatiin myös käytettäessä Node-RED-ohjelmaa PC-koneella. Lopullisella Datankerääjän ohjelmalla Node-RED web-käyttöliittymä jumiutuu jo nopeuden ollessa 100 millisekunttia. Ei ole suositeltavaa käyttämään Datankerääjän ohjelmassa alle 500 millisekunnin nopeuksia.

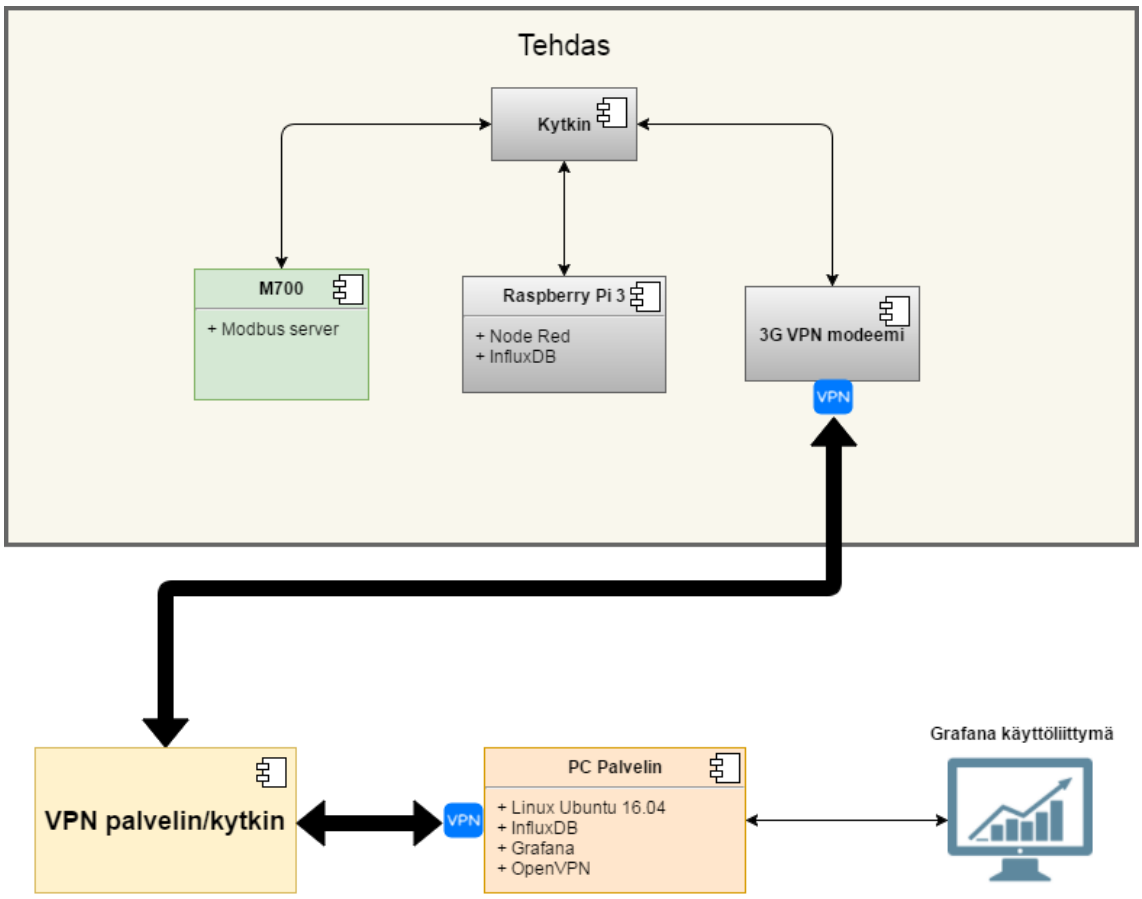
Moottorin pysähtyessä moottorin nopeusarvo käy ennen nollaan menemistään 6550:ssa. Luettaessa Modbus-rekisteriä sekunnin nopeudella, pysäytettäessä moottori saatiin välillä yksi virheellinen luku, joka oli 6550.



Kuva 43. SKS:n tiloissa tehtyjen testien kokoonpano.

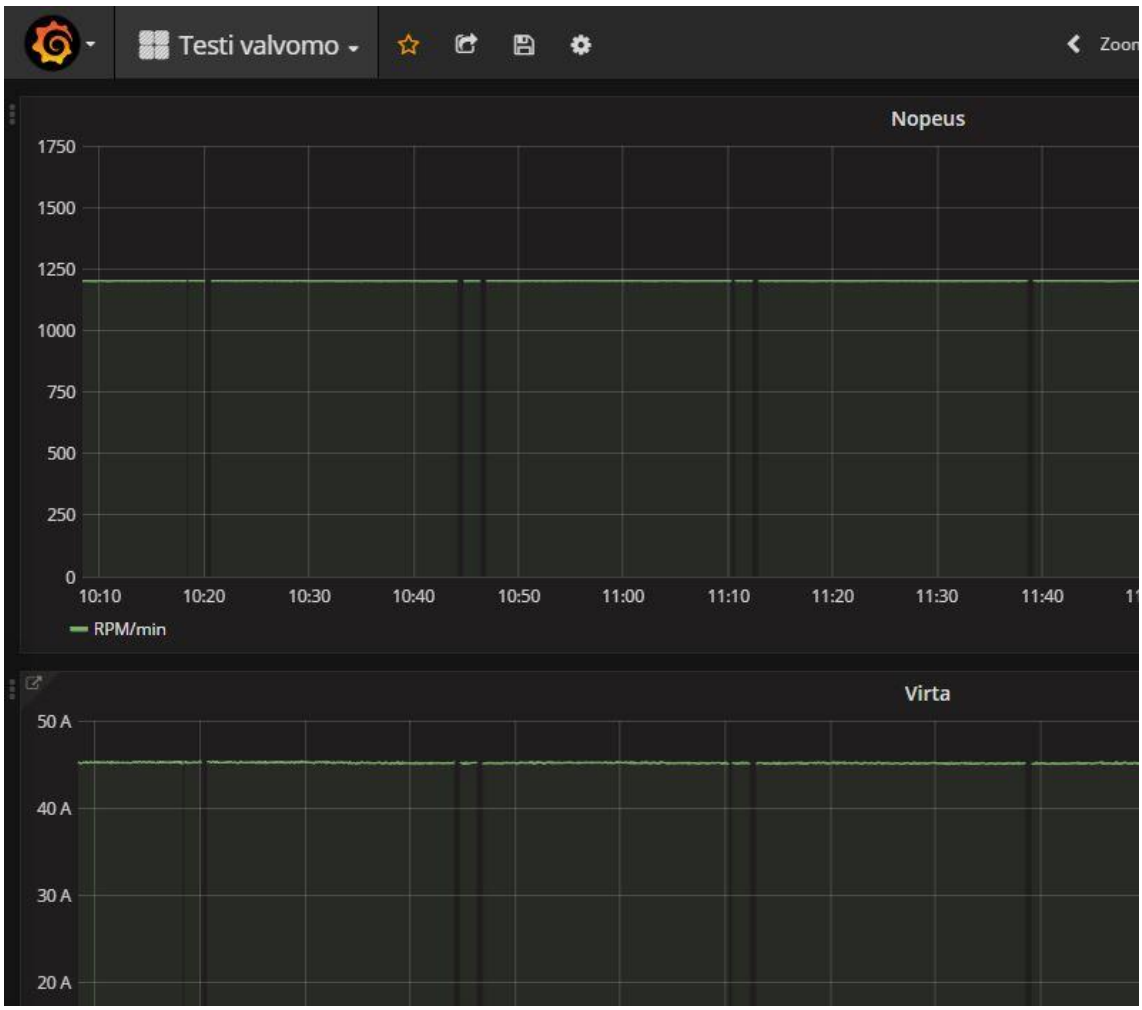
7.2 Kenttätestit

Kenttätestit tehtiin asiakkaan tehdastiloissa. Asiakkaalla on Unidrive M700, joka käyttää kumimurskainta pyörittävää oikosulkumoottoria. Kenttätestejä varten tarvittiin VPN-yhteys PC-palvelimen ja Datankerääjän (kuvassa 42. Raspberry Pi 3) välille. Yhteys toteutettiin kytkemällä Datankerääjä 3G VPN -modeemiin, joka loi VPN-yhteyden SKS:n tiloissa olevaan VPN-palvelimeen (Kuva 44). InfluxDB- ja Grafana-palvelimena toimivaan Ubuntu-koneeseen asennettiin OpenVPN-ohjelma, jolla luotiin yhteys SKS:n VPN-palvelimeen. VPN-palvelin konfigurointiin niin, että InfluxDB- ja Grafana-palvelimena toimiva PC ja Datankerääjä olivat samassa aliverkossa. Kytkentää testattiin aluksi SKS:n tiloissa, jotta voitiin varmistua sen toiminnasta ennen kuin laitteisto vietiin kenttätesteihin.



Kuva 44. Kenttätestien kokoonpano.

Kenttätesteissä tallennettiin M700 drivelta tietokantaan nopeusarvo, virta-arvo ja trip-arvo. Trip-arvo kertoo viimeisen vikakoodin. Kenttätesteissä Datankerääjän tiedonkeruuseen tuli välillä katkoksia, joita ei ollut tullut laboratoriotesteissä. Kuvassa 45 näkyy tallennuksessa tulleita katkoksia. Katkokset saattavat johtua liian lyhyestä välistä Modbus-jonossa (sivu 16, *Queue delay*). Syytä ei ollut aikaa selvittää tarkemmin.



Kuva 45. Käyttöliittymässä kenttätesteissä tallennettuja pisteitä, jossa näkyy katkoksia.

8 Huomioita

Raspberry on tarkoitettu sammutettavaksi hallitusti. Kenttätesteissä Raspberry sammutettiin kääntämällä virrat pois. On hyvin todennäköistä, että todellisessa käytössä raspberryä tullaan sammuttamaan katkaisemalla virrat eikä hallitusti, kuten on tarkoitettu. Tällainen ei hallittu sammuttaminen voi aiheuttaa pitemmällä aika välillä Raspberryn muistin korruptoitumisen. Mittaustietojen puskurointi Raspberryn muistikortille saattaa ajan myötä vahingoittaa muistikorttia. Muistikortin vahingoittumista voidaan ehkäistä ostamalla muistikortti, joka on tehty kestämään tallentamista ja poistamista. Sammutusnapin käyttöä kokeiltiin, mutta sitä ei saatu toimimaan tarpeeksi luotettavasti.

Node-RED-ohjelmaa kehitetään koko ajan, joten siihen tulee koko ajan lisää nodeja eri käyttötarkoituksiin. Node-RED:n käyttöön löytyy paljon ohjeita, sekä valmiiksi tehtyjä Flow'ta. Datankerääjän ohjelmaa on helppo muokata tarvittaessa toimimaan muiden tietokantojen kanssa, sekä käyttämään jotain muuta protokollaa kuin Modbus TCP:tä kommunikointiin servo-ohjaimien tai vastaavien kanssa.

Lähteet

1 Node.js. 2017. Verkkodokumentti. Wikipedia. https://en.wikipedia.org/wiki/Node.js. Luettu 11.3.2017.

2 Node.js Tutorial Introduction. 2017. Verkkodokumentti. Tutorialpoint. https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm. Luettu 11.3.2017.

3 Rodger, Lea. 2016. Node-RED Programming Guide. Lecture 1 – A brief introduction to node-RED. Verkkodokumentti. http://noderedguide.com/nr-lecture-1/. 1.1.2016. Luettu 12.3.2017.

4 InfluxDB. 2017. Verkkodokumentti. Wikipedia. https://en.wikipedia.org/wiki/InfluxDB. Luettu 12.3.2017.

5 InfluxDB V1.2 Key Concepts. Verkkodokumentti. Influxdata. https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/. Luettu 12.3.2017.

6 Grafana Labs Getting started. 2017. Verkkodokumentti. Grafana. http://docs.grafana.org/guides/getting_started/. Luettu 18.3.2017.

7 data flow ”gearbox”. 2016. Verkkodokumentti. node-RED. https://flows.nodered.org/flow/37f68e4279304a7f864e2d9dd76fbba2. Luettu 6.2.2017.

8 Klaus, Landsdorf. 2017. node-red-contrib-modbus. Verkkodokumentti. https://github.com/biancode/node-red-contrib-modbus/wiki. 20.1.2017. Luettu 27.1.2017.