eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Lana

Software plataforma baten diseinu, zabaltze eta balidazioa, zerbitzu industrialen sorkuntza eta zabaltzea errazteko.

Egilea

Alex de Miguel Rikondo

2022

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritzako Gradua

Konputagailuen Ingeniaritza

Gradu Amaierako Lana

Software plataforma baten diseinu, zabaltze eta balidazioa, zerbitzu industrialen sorkuntza eta zabaltzea errazteko.

Egilea

Alex de Miguel Rikondo

Zuzendariak

Jose A. Pascual Saiz (UPV/EHU)

Ander Garcia Gangoiti (Vicomtech)

Laburpena

Industria 4.0 aren indarra areagotzen doan einean, makina industrialetan sortzen diren datuak geroz eta garrantzi gehiago hartzen hasi dira. Hala nola, sortzen diren datuen bolumena geroz eta handiagoa denez datu hauen tratamenduak ere balio eta konplexutasun handia hartu du.

Proiektu honetan datu hauen tratamendua, analisia, eta bistaratzea erraztuko duen edge computing motako software plataforma bat sortzea izango da helburua, gerora zerbitzu industrialen sorkuntza eta zabaltzea errazteko.

Proiektu hau aurrera eramateko industria munduan ezagunak diren OPC UA eta MQTT protokoloak aztertu eta erabiliko dira IT (*Information Technology*) eta OT (*Operational Technology*) munduak konektatzeko. Gero, denbora serieak, datu ez estrukturatuak eta frekuentzi altuko datuak biltegiratzeko sistema bat inplementatu beharko da, kasu honetan *TimescaleDB* datu basea baliatuz. Amaitzeko datu hauen analisia eta bistaratzea egiteko *Grafana* datu bistaratzailea izango da erabilia.

Guzti hau makina edo sistema ezberdinetan inplementatu ahal beharko da, inoiz ez baitago jakiterik PLC batera zein sistema duen makina konektatuko den. Horretarako proiektuko zati guztiak *Docker Container* erabiliz enkapsulatuak egongo dira.

Proiektuak bi inplementazio fase izango ditu: lehenengoa, simulazio bidezko inplementazioa izango da. Fase honetan makina komunikazioko protokoloak ulertzen eta dituzten konfigurazio ezberdinak lantzean oinarrituko da, komunikazioko zerbitzari zein bezero aldeak kontrolatuz. Bigarren fasea aldiz, simulatzailean oinarritutako inplementazioan lortutako funtzionamendua balioztatu beharko da. Horretarako softwarea benetako makina batean funtzionatzeko aldatu beharko da, oraingo honetan lortutako datuak erabilera erreal batera bideratzeko.

Gaien aurkibidea

Irudien aurkibidea

Taulen aurkibidea

1. KAPITULUA

Sarrera

Lehen industria iraultza gertatu zenetik, prozesu industrialen automatizazioa bilatzen joan da sektoreko arlo guztietan. Teknologiak aurrera egin ahala industria mundua ikaragarrizko aldaketak jasaten joan da. Gaur egun arte industriak lau iraultza izan dituela esaten da, azkenekoa honi industriaren 4.iraultza edo industria 4.0 deituz.

Iraultza honek prozesuen automatizazioan jasotako datuen analisia eta honen prozesamendua du oinarritzat. Teknologiak aurrera egin duen ahala honen zabalkundea ikaragarria izan da sektore guztietan. Zabalkunde honek teknologien fabrikatze prozesua asko merketu du, amaierako produktuen prezioa asko jaitsiz hasiera batean zituztenekin konparatuta, eta hau da industria sektorean erabiltzen diren sentsoreei pasa zaiena. Denborak aurrera egin ahala, sentsore hauen prezioa asko jaitsi da, hauen erabilera sektorean asko handituz. Sentsore hauen erabilerak prozesuen inguruko informazioa geroz eta zehatzagoa izatera ahalbidetzen du. Informazio zehatz hau izateak atzetik geroz eta datu gehiago izatera eraman du, honek biltegiratze arazoak izatera ekarri duelarik. Orain arte oso ohikoa zen enpresa bakoitzak bere biltegiratze zerbitzaria izatea bere eraikinean bertan. Gaur egun geroz eta ohikoagoa bilakatzen ari da hirugarren enpresa batek dauzkan biltegiratze sistemak erabiltzea saretik konektatuta. Enpresa hauek, harpidetza baten truke beraiek dauzkaten biltegiratze sistema aurreratuak erabiltzeko aukera ematen dute, "cloud"edo hodeiko biltegiraketa bezala ezagutuz.

Cloud teknologia hau oso famatua eta erabilia izaten ari da mota guztietako sektoreen artean, izan ere enpresa ez hain handiei biltegiraketa sistema bat izateko aukera ematen die hasierako inbertsio handirik egin gabe. Zerbitzu hau erabiltzeko, harpidetza bat eta inter-

neterako konexio bat besterik ez baita behar. Teknologia honek onura asko dituen arren, bere alde txarra ere badu. Industriako sektorean esaterako, aipatu dugun datu kantitate handi hau biltegiratzeko interneterako konexioan inbutu efektua sor dezake banda zabalera ez badago biltegiratu nahi diren datuetara eginda. Honi erantzun bat emateko edge computing izeneko teknika erabiltzen hasi zen. Teknika hau datuen aurre tratamenduan oinarritzen da, sentsoreetatik lortzen diren datuak zuzenean hodeira igo beharrean, sistema lokal batean datuak tratatu eta tratamendu honetan lortzen diren emaitzak bakarrik igoz momentuan hodeira, beste datu gordin guztiak, nahi izan ezkero, beranduago edo konexioa libre dagoen momentuan igoz.

2. KAPITULUA

Proiektuaren helburuen dokumentua

Proiektu honen helburu nagusia zerbitzu industrialen sorkuntza eta zabaltzea erraztuko duen software plataforma bat sortzea izango da, industria 4.0 munduko makinetan erabiltzen diren komunikazio protokoloa erabiliz. Kontuan hartuta makina bakoitzak bere PLC propioak dituela eta bakoitzetik datu ezberdinak ateratzen direla, softwarea eraldatzeko erraza izatea bilatuko da, modulu txiki eta generalizatuetan banatuta.

Hau aurrera eramateko, sektoreko makina komunikazio protokoloa ongi ikertu beharko da. Proiektuan erabiliko diren protokoloen aukeraketa Vicomtech-ek industria munduan aurretik duen esperientzian oinarrituta egin da, erabiliko diren protokoloak OPC UA eta MQTT izanik. Protokolo hauekin hasiera batean bezero zein zerbitzariaren konfiguraketak landuko dira, protokoloek eskaintzen dituzten funtzionalitate ezberdinak proban jartzeko.

Ongi ikertu beharrekoa beste teknologia bat Docker container enkapsulaketa softwarea da. Honek emango baitio gure proiektuari konpatibilitate aldetik erraztasuna makina eta sistema ezberdinetan erabiltzeko garaian.

Probak egoki egiteko, kontrolpean daukagun ingurune bat sortu beharko da. Horretarako Python eta bertako Transitions paketea erabiliko da denbora errealeko datuak sortzen dituen simulatzaile bat sortzeko. Simulatzailean zerbitzariaren aldea jarriko da proban, konfigurazio ezberdinak probatzeko. Bestalde, bezeroaren aldea ahalik eta modulagarriena egin beharko da gero benetako makina batera migrazioa erraza izan dadin. Horretarako, programazio modua asko zaindu beharko da.

Datuen analisia eta bistaratzea egiteko, Timescaledb datu basea, PostgreSQL datu base

gestio lengoaia eta Grafana datu bistaratzailea erabiliko dira. Software hauek, bakoitzaren Docker container bertsioan erabiliko dira. Horretarako software bakoitzaren konfigurazio ezaugarriak ikertu eta analizatu beharko dira, guk beharko ditugun funtzionalitateak bakarrik erabiltzeko eta ahalik eta software sinple eta arinena sortzeko.

Behin probak egin direnean eta funtzionamendua egokia dela ziurtatu denean, softwarea enpresak bertan duen industria makina baten maketa batean aplikatuko da. Hemengo funtzionamendua egokia baldin bada, makinako datuak hirugarren batek erabiltzeko prest utziko ditu softwareak.

Laburbilduz, honako hauek izango dira proiektuan zehar bete beharko diren helburu nagusienak:

- OPC UA eta MQTT protokoloen inguruan ikertu.
- Docker container-aren erabilera ikertu.
- Diseinu ulergarri eta modular bat sortzea.
- Analisi eta bistaratze modulua sortu.
- Softwarearen migrazioa burutu.
- Hirugarrengo batzuentzat erabilgarri utzi plataforma.

Proiektuaren gaiarekin zuzenki harremanik eduki ez arren, enpresa baten barruan lan egiten ikastea eta talde baten parte izaten jakitea helburu garrantzitsu bezala hartu dira. Etorkizun hurbil batean modu pertsonalean baliagarriak izango diren ezaugarriak baitira.

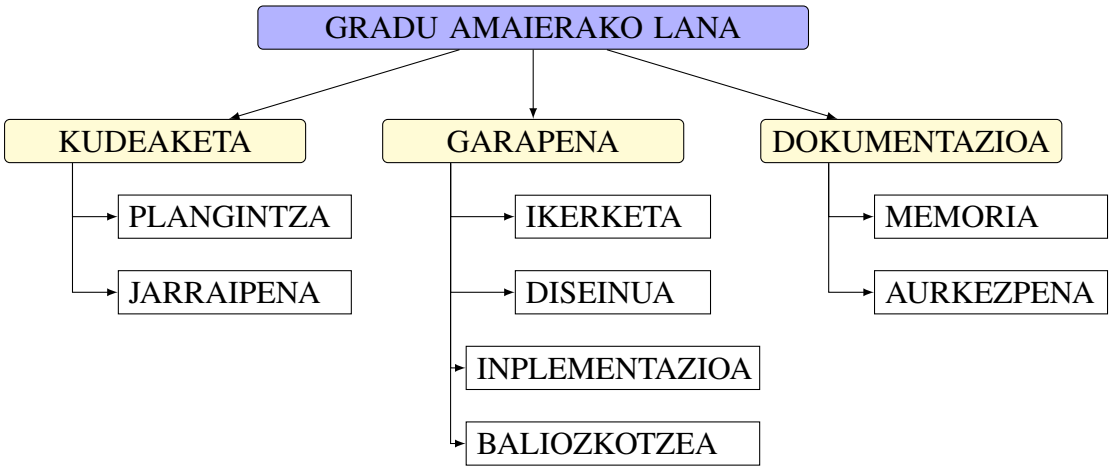
3. KAPITULUA

Plangintza

Proiektu bat egiteko garaian ezinbestekoa da lanean hasi aurretik honen plangintza bat egitea. Plangintza honetan, proiektuan zehar burutu beharko diren atazak azaldu eta hauetan sor daitezken arazoak identifikatu beharko dira, eginiko lana errazteko eta arazoak sor ditzazketen puntuetan atzerapenik sor ez daitezen. Proiektuaren plangintza amaitzeko, ataza bakoitza burutzeko estimatzen den denborak erabaki beharko dira, atazaren lan kargarekin koherentzia mantenduz eta proiektuaren helburuekin bat eginez.

Proiektu honen garapena hiru fase nagusitan banatu da: lehen fasea proiektuaren kudeaketa izango da. Fase honetan proiektuaren helburuak definituko dira eta honen antolakuntza burutuko da. Bigarren fasea proiektuaren garapena izango da. Zati honetan proiektuan ezarri diren helburuak betetzeko beharrezkoak izango diren zati praktikoak gauzatuko dira. Azken fasea dokumentazioa izango da. Fase honetan, proiektu osoan zehar eginikoa eta lorturiko emaitzak azalduko dira.

Bestalde, orain aipatutako hiru fase hauek azpi fase ezberdinetan banatuta daude, eta azken hauek beste oinarrizko ataza batzuetan daude banatuta, hurrengo atalean azaltzen den bezala.



3.1 Irudia: Proiektuaren LDE diagrama

3.1 LDE diagrama

Proiektua aurrera eramateko aipatutako faseak modu errazago batean azaltzeko LDE (Lanaren Deskonposaketa Egitura) diagrama batean jarri dira. 3.1 irudian ikus daitekeen bezala geratuko zen faseen deskonposaketa.

3.2 Faseen deskribapena

Atal honetan, izendatu ditugun proiektuaren faseak xehetasun gehiagorekin azalduko dira. Ataza bakoitza zertan datzan eta bakoitzaren barnean dauden azpiataza guztiak ongi azalduz.

3.2.1 Kudeaketa

Kudeaketa fasearen helburu proiektuaren plangintza eta gainontzeko fase guztien iraupena estimatzea izango da. Kudeaketa proiektua sortu ostean egin beharreko lehen ataza izango da, eta proiektuaren amaiera iritsi arte egongo da martxan, fase guztien jarraipena egiten. Beraz esan daiteke kudeaketaren iraupena proiektuaren bizitza iraupenaren berdina izango dela. Fase hau 3.1 irudian ikusi daiteken bezala bi azpi atazetan dago banatuta:

- **Plangintza:** Atal honetan proiektu osoak izango duen lan egitura prestatuko da. Proiektu honekin lortu nahi diren helburuak erabaki, hauek betetzeko atazak diseinatu eta ataza bakoitza osatzeko estimatzen den denbora kalkulatu beharko da.

Denbora estimazioa egiteko kontuan hartu beharko da ataza burutzen den bitartean sor daitezken desbiderapenak.

- **Jarraipena:** Atal honi dagokionez, proiektuan zehar eginiko lanak plangintzan planeatu diren moduan doazen kontrolatuko dira. Bestalde, proiektuaren jarraipen zehatzago bat eramateko, atazen mugarriez gain, astean behin enpresako tutorearekin buruz-buruko bilera labur batzuk egingo dira eginiko aurrerapenak erakutsi eta hauetan egon daitezken akatsak edo hobekuntzak azaleratzeko. Metodologia honekin, atazetan sor daitezkeen atzerapenak edo desbiderapenak ekiditzen saiatuko da.

3.2.2 Garapena

Garapenaren fasea, lan karga handiena izango duen fasea izango da dudarik gabe. Bertan proiektuan erabili beharreko teknologien ikerketa, sortu beharreko softwarearen diseinua, honen inplementazioa eta amaierako baliozkotzea egongo dira bilduta.

- **Ikerketa:** Ataza honetan proiektua aurrera eramateko teknologien inguruko informazioa bilatu eta hauen funtzionamenduarekin trebatuko da proiektuaren garapenareki hasi aurretik. Proiektu honetan hurrengo teknologiak erabiliko dira:
  - Python programazio lengoaia.
  - Docker container.
  - OPC UA komunikazio protokoloa.
  - MQTT komunikazio protokoloa.
  - Timescaledb datu basea.
  - Grafana datu bistaratzailea.
- **Diseinua:** Ataza honetan, proiektuan sortu behar den softwararen egitura osoa diseinatuko da. Diseinuaren planteamendua ondoren egingo den inplementazioan pentsatuz egin beharko da.
- **Inplementazioa:** Diseinatutakoa inplementatzerako garaian, proba ezberdinak egingo dira, teknologiek eskaintzen dizkiguten aukera ezberdinak probatu eta gure proiekturako egokien etortzen zaigun konbinazioarekin jo arte.
- **Baliozkotzea:** Ataza honetan, inplementazioan sorturiko sofwarea bizitza errealeko egoera batera eramango da. Enpresako laborategietako makina batean jarriko da martxan funtzionamendu egokia balioztatzeko.

3.2.3 Dokumentazioa

Azken fase honetan, proiektuan zehar eginikoa dokumentatu beharko da.

- **Memoria:** Ataza honetan, gainontzeko faseetan eginiko lanaren memoria idatzi beharko da, eta horretarako *Overleaf* web aplikazioa erabiliko da LaTeXeditore bezala.
- **Aurkezpena:** Proiektuaren azken ataza honetan, eginiko memoriaren laburpen bat egin beharko da, 15-20 minutu inguru iraungo dituen ahozko aurkezpen batean azaltzeko.

3.3 Lanen estimazioak eta desbiderapenak

Proiektuko lehen atazan eginiko plangintzan ataza bakoitzari eskaini beharko zitzaion lan kargaren estimazioa egin da (lan karga ordutan neurtu da). 3.1 taulan estimazio hori eta errealitatean eskaini zaion lan karga konparatu ahalko da.

	Aurreikusitako orduak(h)	Benetako orduak(h)
KUDEAKETA	**19**	**16**
Plangintza	10	8
Jarraipena	9	8
GARAPENA	**220**	**240**
Ikerketa	40	40
Diseinua	40	40
Inplementazioa	100	110
Baliozkotzea	40	50
DOKUMENTAZIOA	**95**	**85**
Memoria	80	75
Aurkezpena	15	10
GUZTIRA	**334**	**341**

3.1 Taula: Proiektuaren aurreikusitako eta benetako orduak.

3.4 Arrisku plana

Proiektu bat egiterako garaian, eta are gehiago luzapen jakin bat duten lanetan, ohikoa da atzerapenak sortzen dituzten ezustekoak agertzea. Beraz ezinbestekoa da proiektuak

irauten duen bitartean gerta daitezken arriskuen analisi bat egitea. Behar bada ezinezkoa izan daiteke sortu den arazoa atzerapenik gabe konpontzea, baina behar bada honek sor ditzaken arazoak murriztea bai, eta honela hurrengo atazetan ahalik eta eragin gutxiena sortu.

Arrisku handienak garapen fasean egongo direla aurreikusi dira, izan ere lan karga handiena fase honetan aurkitzen da. Lehen arriskua, teknologia berrien erabilera eta inplementazioa aurreikusten da. Beste arrisku momentu bat inplementazioaren garaian egon daitekela ikusi da, programa guztietan bezala hemen ezusteko erroreak eta hasierako diseinuaren berdiseinaketak gerta daiztezkelako. Azkenik, arrisku handiena baliozkotze atazan egon daitekela aurreikusi da. Softwarea inplementatu beharko den maketa ez baita asko erabili azken denboraldian eta honek arazoak izatera eraman gaitzake.

Bestalde, bizitzen ari garen unean, Covid 19-ak sor ditzaken arazoak beti izan behar dira kontuan. Horregatik proiektua egiten den bitartean enpresan ezarrita dauden arauak jarraituko dira mota honetako arazorik gertatu ezkero.

3.5 Lan-metodologia

Lan-metodologiari dagokionez, proiektuaren jarraipen egoki bat eramateko, astean behin enpresako tutorearekin biltzea adostu da ordurarte eginiko lana erakutsi eta aurrera jarraitzeko prest dagoen baieztatzeko. Modu honetara sor daitezken erroreak eta dudak lehenbailehen argituko dira. Bilera hauek ez dute egun ez ordu zehatzik izango, ikasle eta tutoreareak libre duten momentu batean egingo dira. Bestalde, duda edo arazo puntualak momentuan kontsultatzeko ez da arazorik izango ez tutorearekin edo enpresako saileko beste edozeinekin aurrez aurre edo korreo bidez.

Proiektuaren helburua kontuan hartuta, garrantzi handia izango du softwarea programatzeko moduak, honen eskalabilitatea eta modulartasuna. Beraz, puntu hau ondo errebisatua izango da.

Proiektua enpresan egin denez, hemengo egonaldia luzatzea erabaki da eta honen harira gradu amaierako lana irailean entregatzea erabaki da, lan honetaz gain borondatezko praktika periodo bat egin delako.

3.6 Desbiderapenen analisia

Proiektuak aurrera egin ahala zenbait desbiderapen joan ziren agertzen. Lehendabiziko arazoak proiektuan erabili beharko nituen teknologien inguruan ikertzen hastean azaleratu ziren, izan ere, hauetako asko inoiz ikusi gabekoak zirenez, uste baina denbora gehiago hartu zuten modu egoki batean funtzionatzen jartzeak.

Proiektuko lehen desbiderapena Grafana datu bistaratzailea inplementatzerako garaian gertatu zen. Hemen software honen bertsio berri baten harira gertatu zen, oraindik ez zuen dokumentazio oso detailaturik, beraz aurreko bertsio batera pasa behar izatera eraman gintuen.

Inplementazio fasean bestelako desbiderapenik egon ez arren, proiektuaren baliozkotze momentuan iritsi ziren arazoak. Proiektuaren eraldaketa egiten ari nintzen momentuan laborategian zegoen makinak funtzionatzerai utzi zion, eta enpresan makina honen berri zuen pertsona bajan egon zen denbora askoz beraz honen itzulera arte proiektua geldirik egon zen.

4. KAPITULUA

Teknologien deskribapena

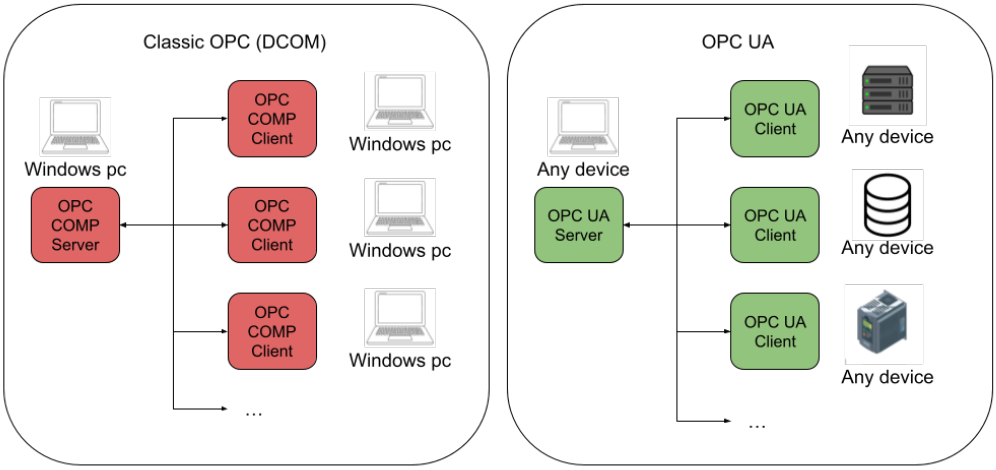
Proiektua nola sortua izan den azaltzen hasi aurretik, bertan erabili diren teknologiak deskribatuko dira atal honetan. Proiektuaren helburua kontuan hartuta, erabiliko diren komunikazio protokoloak, datu biltegiratzea eta datu bistaratzea egiteko erabilitako teknologiak deskribatuko dira. Bestalde, proiektu osoaren portabilitatea handitzen lagunduko duen docker softwarea ere azalduko da atal honetan.

4.1 Makina komunikazioa

Makinen arteko komunikazioa proiektu honen funtsa da. Industria munduan azken aldian, datuen tratamenduaren balioa eta garrantzia handitzen doan bezala komunikazio hau geroz eta ezinbestekoagoa da. Proiektu honen kasuan, momemtuan pil-pilean dauden bi teknologia deskribatu eta alderatuko dira.

4.1.1 OPC Unified Architecture

OPC (OLE for Proccess Control) [Mahnke et al., 2009], Microsoft-en teknologian oinarritua dagoen komunikazio estandar bat da, prozesu industrialen gainbegiraketan eta kontrolean finkatua zegoena. Industria automatizatuko beharren gaineko soluzio bat eskeintzen sortu zen. Sektorean oso erabiliak ziren HMI (Human Machine Interface) eta SCADA (Supervisor Control And Data Adquisition) bezalako sistemei erraztasunak emateko izan zen diseinatua. OPC protokoloak industrian izaniko onarpena eta hazkundea ikusirik



4.1 Irudia: OPC vs OPC UA

OPC foundation-ek sare ezberdinetan zeuden makinak komunikatzeko protokoloaren bertsio berri bat argitaratu zuten, gaur egun oraindik erabilia den OPC Unified Architecture.

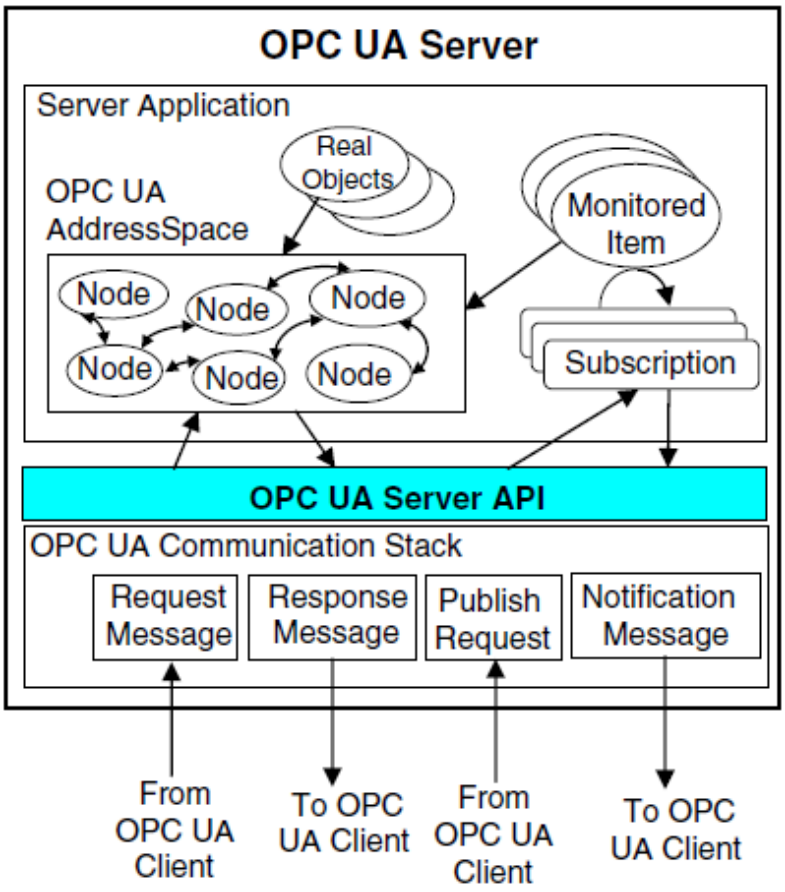
OPC UA (Open Platform Communication Unified Architecture), M2M(Machiine to machine) komunikazio, elkarreragingarritasun eta integrazio bertikal zein horizontalean erabilia izateko izan zen diseinatua, bere aurrekari zen OPC protokoloak zituen hutsuneak asetzeko nahiarekin, 4.1 irudian ikus daiteken bezala.

Arkitektura

OPC UA protokoloak, aplikazio programa interfazea (API) erabiltzen du bezero eta zerbitzari arteko komunikazioak egiteko [Leitner and Mahnke, 2006]. API honek, bezero/-zerbitzari komunikazioa, OPC UA komunikazio pilatik isolatua mantentzen du. Komunikazio pilatik dei bat hartu, dei hori mezu bilakatu eta komunikazio bidera bidaltzen da.

OPC UA komunikazio pilak ez du inolako tegnologiara dependentziarik, honek, diseinuan inolako moldaketarik egin gabe, egungo teknologietara zein etorkizunekoetara moldatzeko erraztasuna ematen dio protokoloari [Cavalieri and Chiacchio, 2013]. Gaur egun, datuak kodetzeko bi metodo daude definituta: XML/text eta UABinary. Bestalde, garraioari begira, bi mapeatu daude erabilgarri: UA TCP eta SOAP Web Services HTTP gainetik.

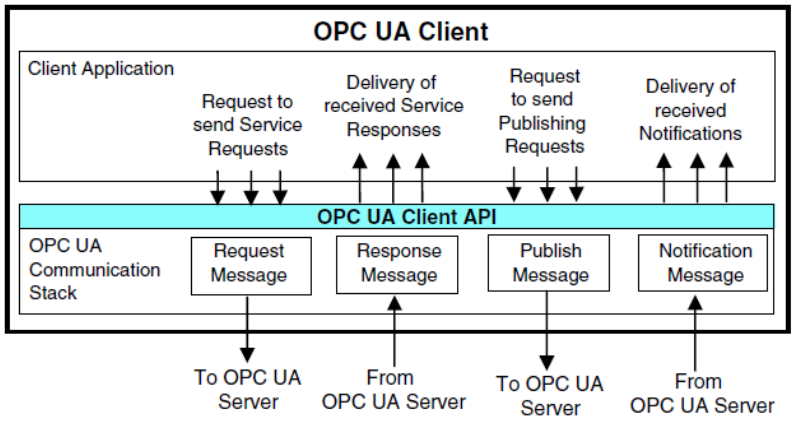
4.2 irudian OPC UA ko zerbitzariak duen arkitektura ikus daiteke. Zerbitzariak jatorri fisikoetatik zein software bidezko jatorritik jaso dezake informazioa. Informazio hau Nodo



4.2 Irudia: OPC UA zerbitzaria. Iturria: Analysis of OPC UA performances

izeneko objektuetan gordetzen eta tratatzen da gerora komunikazio pilan jartzeko. Nodo hauek, komunikazioan gastuak gutxitzeko sortzen da, datuen konprimaketa baten antzera funtzionatuz.

Bestalde, bezeroaren aldean, honek zerbitzariak sorturiko Nodoetara harpidetu daiteke. Behin nahi edo behar den harpidetza egin dela, bertako informazioa une oro eskuratu, edo Nodoak jasan ditzaken aldaketetan alarmak edo gertaerak sortu hauetaz jakinarazpenak jaso ahalko ditu bezeroak, 4.3 irudian ikusi daiteken bezala.



4.3 Irudia: OPC UA bezeroa. Iturria: Analysis of OPC UA performances

OPC UA protokoloak, bezeroaren aldean datuak eskuratzeko garaian, zenbait aukera ematen dizkigu datu inguruan egin nahi den erabilerara moldatzeko. Zehazki hiru modu jartzen ditu erabiltzailearen esku datuen tratamendua egiteko:

- **Datuetara sarbidea**

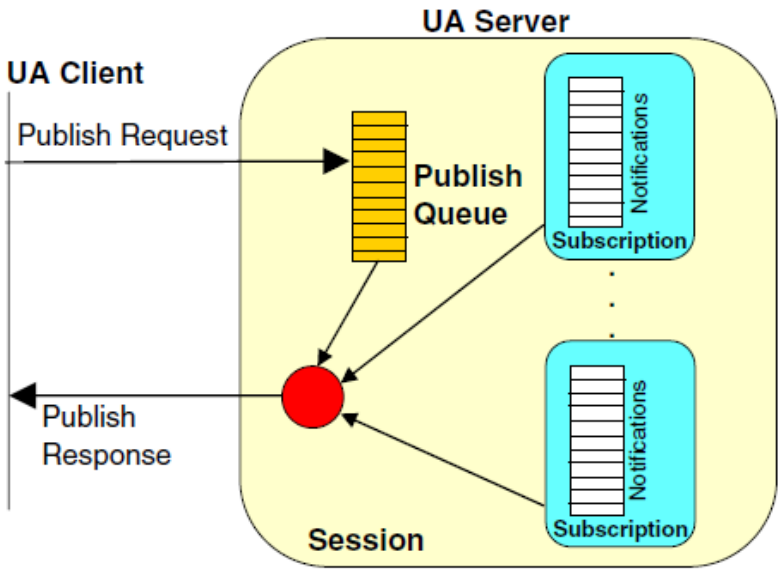
  OPC UAko komunikazioa sortzen denetik, konexioa itxi arte mezuen joan etorria konstantea denez, protokolo hau ezin hobea da denbora errealeko datuekin lan egiteko. Datu hauek irakurtzeko, zerbitzariak sortzen dituen Nodoak jasotzen ditu bezeroak eta hauetatik nahi diren aldagaien datuak eskuratu ahalko dira denbora errealean.

- **Datu historikora sarbidea**

  Denbora errealeko datuetara sarbidea izateaz gain, OPC UA protokoloak datuen historikora sartzeko aukera dauka baita ere. Aztertu edo jaso nahi diren datuen data tarte bat jarrita, tarte honetako datu historikoa itzuliko digu.

- **Alarma eta gertaerak**

  Datuak modu ezberdin eta sofistikatu batean eskuratzeko, OPC UAk datuak monitorizatzea edo hauetara harpidetzeko aukera ematen du, 4.4 irudian ikus daiteken modura, alarmak edo gertaerak sortuz. Gertaera batean, hautatutako aldagaian erabiltzaileak adierazi duen gertaera bat gauzatzen bada, protokoloak abisu bat jaurtiko du bezeroan. Alarma batean aldiz, bezeroak aukeratutako aldagai batek aldaketa bat jasaten duen bakoitzean jasoko du aldaketaren berri bezeroak.



4.4 Irudia: Datuetara harpidetza. Iturria: Analysis of OPC UA performances

Segurtasuna

OPC UAk bezero eta zerbitzari komunikazioa bermatzeko segurtasun politikak ezartzeko aukera ematen du. Bezero eta zerbitzari arteko komunikazioa ezartzeko garaian, zerbitzariak komunikazio kanala babesteko erabiliko diren politikak ezartzen ditu. Babes metodo bezala, mezuen enkriptazioa edo zertifikazio digitalak eska ditzakeen bezala, segurtasunik gabeko komunikazioak sor daitezke. Bestalde, erabiltzaileak komunikazio honen gainetik segurtasun maila gehiago aplika ditzake nahi izan ezkero.

4.1.2 MQTT

MQTT (Message Queuing Telemetry Transport) komunikazio protokoloa, azken urtetan geroz eta indar gehiago hartzen joan den protokolo bat da [Soni and Makwana, 2017]. Protokolo hau IBM enpresak sortu eta erabilera publikora zabaldu zuen M2Mko pisu gutxiko komunikazio protokolo bezala. Azkenaldian geroz eta indar gehiago hartzen joan den IoT teknologiari lotu da, eskaintzen dituen ezaugarri eta arintasunagatik.

Protokolo honek publikazio/harpidetza konexio arkitektura jarraitzen du. MQTTren publikazio/harpidetza arkitekturan hiru elementuk hartzen dute parte: makina publikatzailea edo informazioa publikatzean duen makina; "Broker"bitartekaria edo makina publikatzailetik jasotako informazioa gorde eta bezeroei bidaltzeko lana izango duen makina; eta

bezeroa, broker-eko datuetara harpidetuko den makina.

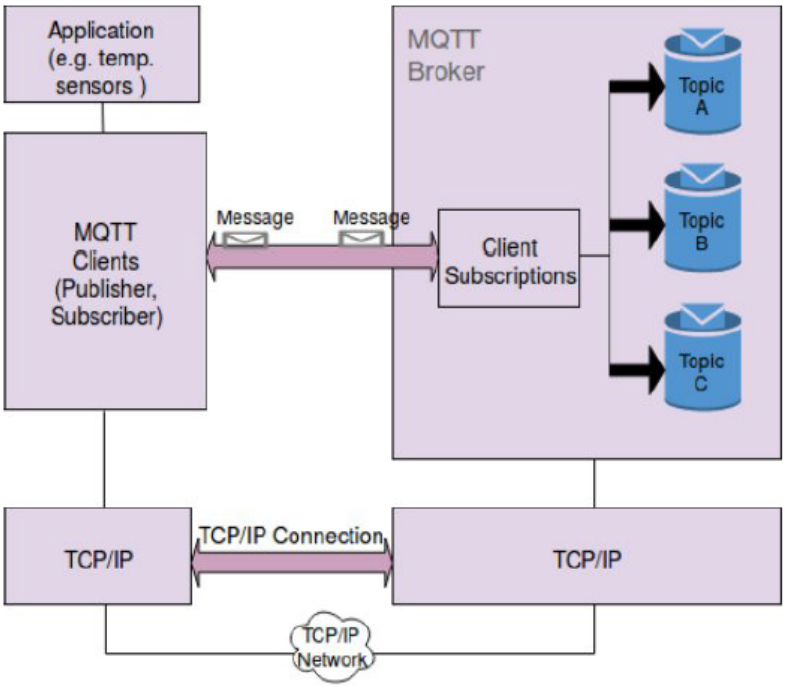
Arkitektura

MQTT oinarrituta dagoen publikazio/harpidetza sistema, komunikazio ez zuzen batean edo bi fasetako komunikazio batean dago oinarritua. Komunikazio honetan bi elementu nagusi bereiztu ahalko dira, bezeroak eta brokerra, bereizketa honetan baita ere, bezero batek rol ezberdinak izan ditzake, publikatzaile zein harpidetua izan daiteke, biak batera izateko aukera izanik momentu guztietan.

- **Bezero publikatzeilea:** MQTT sistema batean esan bezala bezero mota ezberdinak aurki ditzakegu. Publikatzailea, izenak dioen bezala, datuak sisteman publikatzeaz arduratzen da. Bezero honek, brokerrarekin konexioa egin eta *Publish* barne deia erabiliz aldagai ezberdinen datuak publikatzen joango da. Lehenago aipaturiko bi fasetako komunikazioan, bezero publikatzailea komunikazio honen lehen fasea osatuko du.

- **Bezero harpidetua:** Bigarren bezero mota hau, bezero rol klasikoa izango du. Bezero publikatzaileak ez bezala, bezero mota honek bi noranzkotako konexio bat sortuko du brokerrarekin, noranzko bat jaso nahi duen datuetara harpidetza eskaerak egiteko erabiliko du, bitartean kontrako noranzkoan, brokerrean harpidetuta dagoen aldagaien datuak jasotzeko erabiliko du. Bezero honen eta brokerraren artean konexioa onartu izan denean, bezeroa modu estatiko batean hasiko da lanean, brokerraren lana izango baita datuak bidaltzea bezeroa harpidetuta dagoen aldagaiak eguneraketarik jasaten badu. Komunikazio honek osatuko du aurretik aipaturiko faseen bigarren zatia.

- **Broker:** Brokerra komunikazio protokolo honen erdigunea da, komunikazio guztiak bere barrutik pasa behar dira. Bezero publikatzaileak bidaltzen dituen aldagaien publikazio eskaerak onartu behar ditu alde batetik. Brokerrak publikatzeko jasotzen dituen aldagaiak "gai"moduan jasoko ditu, zuhaitz formako egiturak sortuz bezerotik jasotako gai eta azpigaiekin. Bestetik, bezerotik jasotzen dituen harpidetza eskaeren kontrola ere bere gain eroriko da, sistemaren komunikazioen segurtasuna aldi oro bermatuz.

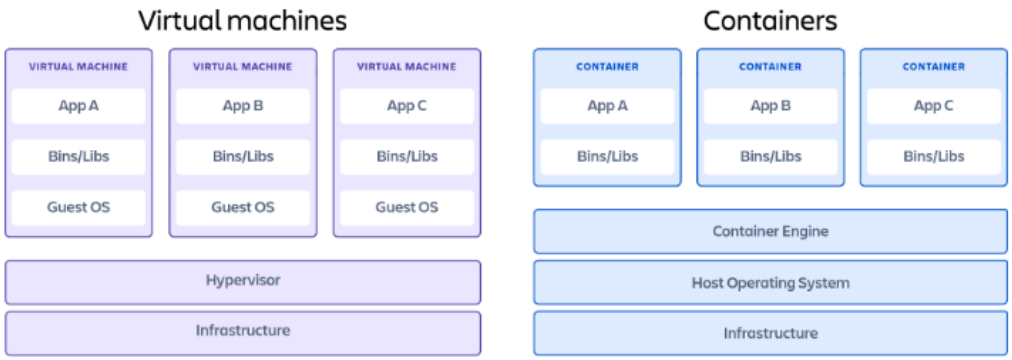


4.5 Irudia: MQTTren arkitektura. Iturria: A survey on mqtt

Brokerra sare bidez atzitzen den elementu bat da, TCP/IP gainetik ezartzen da 4.5 irudian ikus daiteken bezala, baita TLS gainetik segurtasun gehiago aplikatu nahi bada. Bestalde, hiru elementuko komunikazioa izanik, konexioaren eta datuen erredundantzia bermatzea modu errazago batean lor daiteke. Broker elementua, bakar bat izan beharrean sare baten antzera clusteriza daiteke datuak eta konexioak broker guztietan bikoiztuta. Sistemak momenturen batean arazoren bat izan ezkero bertan behera gera ez dadin eta bertan dauden datuak galdu gabe.

Segurtasuna

MQTT protokoloan segurtasuna bermatzeko garaian brocker-aren gainean jarri behar da begirada. Komunikazioko elementu hau da datuen transmisio eta kontrola eramaten duen elementua, honek segurtasunaren pisu guztia bere gainean erortzera eramaten du. Ala ere, protokoloa diseinatua dagoen moduan, ez du inolako segurtasun protokolorik ekartzen araututa. Ezaugarri hau, inplementatzaileari beharrezkoa ikusten duen segurtasun inplementazioa sortzeko askatasuna emateko dago egina, komunikazioaren testuingurura moldatzeko.



4.6 Irudia: Edukiontziak vs Makina Birtualak. Iturria: atlassian.com

4.2 Docker Container

Proiektu honen helburu nagusietako bat, sorturiko softwarea edozein makina edo sistemetan arazorik gabe inplementatu eta martxan jartzea da. Horretarako proiektuaren zabaltzea errazteko softwarea edukiontzitan sartuko da Docker container erabiliz.

4.2.1 Edukiontziak vs Makina Birtualak

Edukiontzia eta Makina Birtualetaz ari garenean, sistema konkretuak birtualizatzeko metodoetaz ari gara hizketan. Metodo hauek, jada gure makinak dituen osagaiak erabilita, software bidez beste makina batek izango zituen RAM memoria, CPU, Sareak, ... simulatzea ahalbidetzen dute, 4.6 irudiak erakusten duen moduan.

Makina Birtualak

Makina Birtual bat, gure makina fisikoa motore bezala hartuta, honen gainean beste sistema "birtual"bat sortzea ahalbidetzen digun birtualizazio metodoa da. Sortu berri den sistemak, guk hautatutako sistema eragile irudia, guk diseinatutako osagai "fisikoak"edo hardwarea, eta sareak izateko aukera ematen digu. Hau dena lehen esan bezala gure "Host"makinako hardwarea erabiliz dena simulatzeko.

Deskribapen orokor bezala, esan daiteke, gure ordenagailu fisikoak bere gainean beste ordenagailu guztiz ezberdin bat simulatzeko erreminta bat dela makina birtuala.

- **Abantailak**
  - Gure makinan arriskua izan dezaken ataza bat gauzatzeko ingurune seguru bat sortzen du.

  - Edozein arazoren aurrean sistemaren berreskurapen erraz eta azkarra izatea ahalbidetzen du.
  - Ingurune fisiko batean (gure ordenagailua adibidez), hainbat makina exekutatzea ahalbidetzen du.

- **Desabantailak**

  - Hainbat makina simulatzeak sistema fisiko berdinean, makina fisikoaren errendimendua modu esanguratsu batean jaistea ekar dezake.

Edukiontziak

Edukiontzien erabilera simulazio arin bat egitean datza. Simulazio hauetan, programak, funtzioak, ataza konkretuak, ... birtualizatzean oinarritzen da, mikrozerbitzuak bezala ere ezagutzen direnak eremu simulatu batean jaurtitzen.

Makina birtualetan ez bezala, edukiontzi batean ez da sistema eragile oso bat simulatzen, "irudi"izeneko kodeetaz baliatzen da aldiz. Irudi hauek, edukiontzi barruan erabili nahi den mikrozerbitzuaren bertsio oso, estatiko eta exekutagarri bat daramate barnean.

Irudien erabilerari esker, edukiontzien portabilitatea ia edozein sistemara ahalbidetzen da. Mikroprozesua exekutatzeko beharrezkoa den guztia irudi barnean dagoenez, ez da kanpoko bestelako zerbitzuren laguntzarik behar.

- **Abantailak**

  - Edukiontzietan birtualizatzeak, sorturikoa guztiz eramangarri eta bateragarri egiten du edozein sistemetan.
  - Baliabideen kudeaketa eraginkorra.
  - Mantenua eramaten erraza eta oso eskalagarria.

- **Desabantailak**

  - Segurtasun aldetik edukiontziek isolamendu arina dute host makinako sistema eragiletik edo sistema bereko beste edukiontziekiko.

4.2.2 Docker

Docker kode libreko motore bat da [Turnbull, 2014], aplikazioak modu automatizatu batean edukiontzitan sartzea ahalbidetzen duena. Aplikazioa Docker Inc taldeak diseinatua da eta Apache 2.0 lizentziapean izan zen argitaratua.

Docker-ek edukiontzien birtualizazio ingurunearen gainetik aplikazioak zabaltzeko motore bat gehitzen du. Diseinu honekin, pisu gutxiko ingurune azkar bat jartzen zaigu eskura edozein kode modu erraz eta eraginkor batean exekutatzeko. Modu berean, diseinu honekin kodearen portabilitatea errazten eta azkartzen da.

Docker hurrengo erraztasunak emateko diseinatu zen:

- **Errealitatea modelatzeko modu erraz eta arin bat.**

  Makina birtualen hiperbisorearen zatia kenduta, edukiontzien efizientzia asko handitzen da eta hauetako askoren erabilera ahalbidetzen du gure host makinan.

- **Funtzioen bereizketa logiko bat.**

  Docker edukiontzien arteko koherentzia bermatzeko dago diseinatua. Honela programatzaileak edukiontzi barnean sartu beharreko programak kodetzeaz bakarrik arduratu behar dira.

- **Garapen azkar eta eraginkorreko bizi-zikloa.**

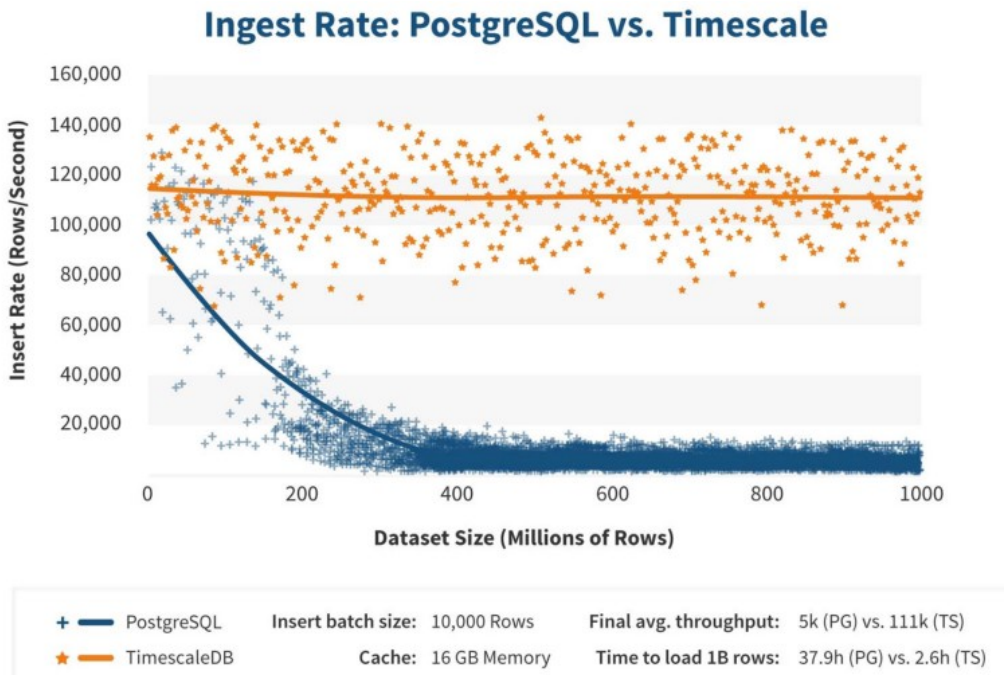
  Docker edukiontzien diseinua, kodeaketa, proba eta zabaltzea ahalik eta azkarren egiteko diseinatu zen. Edukiontzien erabilera eta portabilitatea era erraz batean egiteko kontuan hartuta.

- **Zerbitzuetara bideratutako arkitektura sustatzen du.**

  Docker-ek edukiontzi bakoitzaren barnean ataza edo aplikazio bakarra sartzea gomendatzen du, mikrozerbitzuetan oinarritutako diseinuak egitera bultzatuz. Honela aplikazio banatuen sistema bat sortzen da non edukiontzien konexio sare bezala errepresentatzen den.

4.3 Datu basea

Proiektuan makinak sorturiko datuak tratatzeko lehenengo datu hauek nonbait gorde beharko dira. Horretarako datu base baten laguntza izango da. Proiektuan bestalde, kontuan



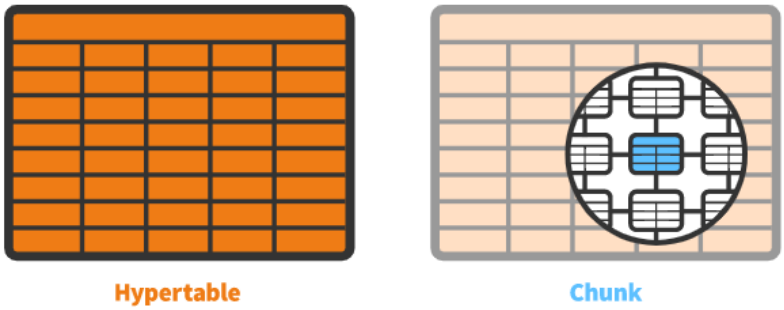
4.7 Irudia: PostgreSQL vs Timescale. Iturria: timescale.com

izan beharko da denbora errealeko datuak tratatu beharko direla, beraz, hauen tratamendu egoki bat egiteko moduko datu base bat aukeratu beharko da.

4.3.1 TimescaleDB

Denbora errealeko datuekin lan egiteko [tim,], datu asko jasotzeko gai den datu base bat erabiltzea da aukerarik onena, ahalik eta datu gehien jasotzeko. Gure kasuan, TimescaleDB datu basea izan da hautatua. TimescaleDB, PostgreSQL datu base lengoaian oinarritua dagoen datu base bat da. Lengoaia honetan oinarritua izatea ez da kasualitatea. Postgres aldibereko datu kantitate handia jasotzeko eginda dagoen lengoaia bat da. Bestalde, taulen aldibereko erabilera izatea ere ahalbidetzen du, taula batean idazten ari den bitartean kapaz da taula berberetik datuak eskuratzeko. Timescalek, Postgres-ek eskaintzen dituen ezaugarriak hartu eta optimizatu egin ditu oraindik ere errendimendu handiagoa izateko. 4.7 irudian ikus daiteken moduan, timescalek bere errendimenduan ez du aldaketa handirik jaso datu kopurua handitzen doan einean, PostgreSQL-ren kasuan den moduan.

Timescale aukeratzearen aldeko beste ezaugarri bat, bere egitura osatzen duten hipertaulak dira. Hipertaula hauek, erabiltzailearentzat gainontzeko taulak bezalakoak diren arren, egitura eta errendimendu aldetik oso ezberdinak dira. Taula hauek, denbora/espazio za-



4.8 Irudia: Hipertaularen egitura. Iturria: timescale.com

tikatze automatiko batean oinarritzen dira, honek datu kantitate handiak modu eraginkor eta optimizatu batean biltegiratzea ahalbidetzen du, erabiltzaileari inolako aldaketarik suposatu gabe taularen erabileran.

Hipertaula hauen funtzionamendua chunk edo zatietan oinarritzen da. Taulak jasotzen dituen datuak denbora tarteka zatitzen dira chunk hauek osatzeko. Zati hauek gero, taula txiki bezala gordeko dira hipertaularen barnean honen espazioa hobeto betetzeko, memoriaren erabilera nabarmen hobetuz, 4.8 irudian ikus daiteken modura. Behin denbora bat pasa denean, Timescalek duen erabilera algoritmoak chunk batzuk konprimatuko ditu memoriaren erabilera eta honen errendimendua are eta gehiago hobetzeko. Pixka bat laburbilduz, hipertaula bat azken finean, taula txikiz osaturiko taula bat dela esan daiteke. Datu konprimaketak datuen eskuratzea mantsotu dezakela imagina daiteke, baina lehen esan bezala, datu chunk hauek denborarekiko daude zatituta, beraz, datu hauetara heltzeak ez du bestelako zailtasunik sortzen eta konprimatuta egoteak aldiko kostu bat bakarrik suposatuko du, izan ere datu batek erabilera nabarmena duela igartzen bada honen konprimaketa atzeratzen baita.

Timescale datu basearen beste alde positibo bat, bere konpatibilitate handia da. Nahiz eta bere cloud zerbitzu propio eta espezializatua eskaini hileroko harpidetza baten truke, timescalek bere softwarea erabiltzeko lizentzia librea eskaintzen du. Datu basea zure maikan bertan docker bidez monta dezakezu honen irudi originalarekin edo librea den API bidez kontratatua daukagun cloud zerbitzarira konekta dezakegu, alderdi honetan ere konpatibilitate handia baitauka cloud zerbitze nagusienekin.

4.4 Datuen bistaratzea

Industria munduan sortzen, tratatzen eta erabiltzen diren datuen kopuruak asko egin du gora azken urteetan, sektorean erabiltzen diren makinaria guztia automatizatzeko nahiarekin. Eboluzio honek, makinek sorturiko datuen tratamenduan eragin handia izan du, datuei balio handia emanez.

Esan bezala, makinek sorturiko datuak biltegiratu ostean tratatu egin behar dira lortu nahi den emaitza lortzeko. Tratamendu honen errendimendua asko ari da aurreratzen machine learning, deep learning eta big data teknologiak aurrera egiten duten bezala. Hala ere, oraindik ia derrigorrezkoa da giza faktore bat sartzea datuek izaten dituzten emaitza hauen erabileran. Sektorean aditua den profesional batek analisien emaitzen berri izatea da normalena oraindik, eta horretarako datuak modu bisual batean ongi adieraztea ezinbestekoa da [Ware, 2019]. Horretarako, datuen informazio panelak erabiltzen dira datuen balioak zein analisien emaitzak erakusteko. Bertan gaian adituak makinaren egoeraren berri zein analisien emaitzak kontrolatuko ditu.

Proiektu hau datuen tratamendu eta analisira bideratuta egonda, datu bistaratzaile batek ezin zuen falta. Horretarako Grafana kode libreko softwarea erabili da.

4.4.1 Grafana

Proiektu hau edozein makinatan eta edozein erabiltzaileek bere sisteman zabaltzeko egina dagoenez, ahalik eta konpatibilitate handiena duen software bat izatea da egokiena. Premisa honen eskakizunak betetzen dituen kode libreko softwarea da Grafana. Konpatibilitate handia izatearekin batera, Grafanak proiektu edo enpresa batek izan ditzaken datu base ezberdinen datuak leku berberetik bistaratzeko aukera ematen du, datu-panel berdinean datu base ezberdinetatik jasotako datuak bistaratuz adibidez.

Konpatibilitate zabal batez gain, Grafanak bistaratze panelak nahi haina editatzeko ahalmena ematen digu modu erraz batean. Konfigurazio aldetik, sorturiko aginte panelak esportatzea ere ahalbidetzen digu Grafanak, behin sortu dugun panela beste erabiltzaile bati pasatzeko ahalmena edo software osoa beste makina batera pasatzerako garaian, hau, jada gure panelekin hasiz.

Aurretik aipatu dugun konpatibilitate zabalaren beste ezaugarri bat, daukan plugin liburutegi zabala da. Plugin hauek gure aginte paneletarako datu adierazle gehigarrietatik hasita,

kanpo programetarako komunikazioak izan daitezke. Aipaturiko guzti hau atzetik konpainia sendo eta komunitate zabal batek babestua.

Beste puntu garrantzitsu bat, datuen kontrolera bideratuta dagoena, gure datu paneletan alarmak sortzeko ahalmena da. Gure aginte panelean jasotzen ditugun datuetan zenbait errore edo jokabide aurreikusi daitezke, honi lotuta, datu hauen gainean alarmak sor ditzakegu jokabide hauek agertu ezkero Grafanak eskura uzten dizkigun bide ezberdinetatik. Datuak kontrolatzen ari den erabiltzaileari abisu bat bidali edo panelean bertan alarma bat agertuz.

Grafana kode libreko software bat izan arren, bere cloud propioa erabiltzeko aukera ematen du hileroko harpidetza baten truke. Proiektu honen kasuan, docker-erako daukan kode libreko irudi ofiziala erabiliko da, portabilitatea eta konfiguragarritasuna kontuan hartuta.

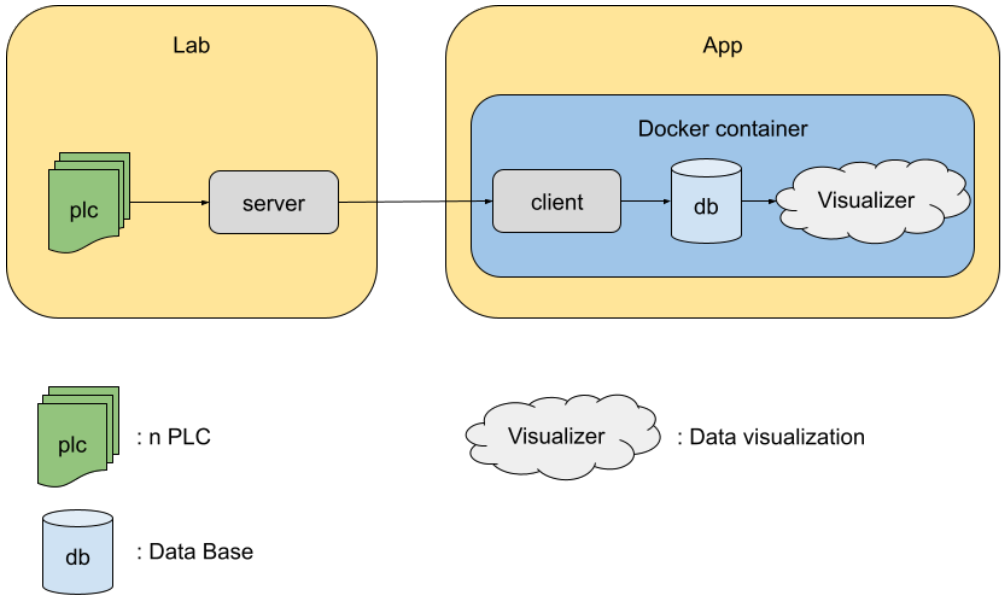
5. KAPITULUA

Inplementazioa

Proiektuaren plangintzan, ezerekin hasi aurretik proiektuak izango zituen helburuak erabaki ziren. Helburu hauek betetzeko planifikazio bat egin zen proiektua zati ezberdinetan banatuz, zati hauetako bakoitzari exekuzio denbora bat aurreikusi zitzaion bertan egin beharreko lana eta gerta zitezkeen arazoak kontuan hartuta. Aurreko atalean azaltzen joan garen moduan, proiektu honetan teknologia ezberdinak erabili dira, hauetako gehiengoa lehen aldiz erabiliz. Honek proiektuaren inplementazioarekin hasi aurretik ikasketa eta inbestigazio fase bat izatera behartu zuen. Behin teknologia hauek kontrolatzeko jakinduria barneratuta, proiektua burutzen hasteko garaia iritsi zen. Inplementazio egoki bat egiteko, garapenarekin hasi aurretik lehenengo diseinu bat egin behar izan zen, proiektua osatuko zuen softwareak izango zituen zati, modulu eta funtzionalitateak definituz. Diseinua behin garbi izanda, proiektua kodetzeko ordua iritsi zen. Atal honetan, aipatu berri den proiektuaren diseinua eta inplementazioaren inguruan hitz egingo da. Bertan izandako faseak, eta gainontzeko zatiak azalduz.

5.1 Diseinua

Proiektuan planteatu den softwarea inplementatzen hasi aurretik ezinbestekoa da honen diseinu bat egitea. Proiektua, simulazio batean oinarritutako zerbitzari/bezero komunikazioa oinarri duen programa bat izango da. Nahiz eta baliozkotze fasean bezeroaren aldea bakarrik inplementatu behar izango den, diseinu honetan komunikazioaren bi aldeak inplementatuko ditugu, protokoloak eskaintzen dituen ezaugarri guztiak frogatzeko.



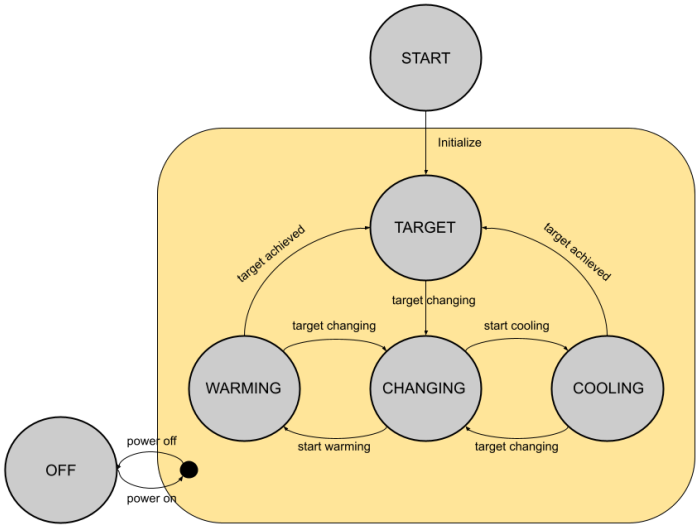
5.1 Irudia: Diseinu nagusia

Programaren helburu nagusia, bezero zerbitzari motako konexioa oinarri duen inplementazioa egitea izango zen. Zerbitzariak simulatzailetik datuak hartu eta bidaliko ditu, bitartean, bezeroak denbora errealean datu hauek eskuratu eta datu base batera igo beharko ditu biltegiratzeko. Azkenik datu hauekin denbora errealeko bistaratze bat diseinatu beharko da informazio horrekin egin nahi denaren arabera.

Inplementazio honetan, aurreko atalean aipatutako teknologiak probatzen zentratuko da. Oinarrizko konfigurazioetatik hasi eta gure proiekturako erabilgarrienak lortu arte. Horretarako lehenengo helburua simulatzaile bat sortzea izan zen.

Simulatzaileak ingurune kontrolatu bat simulatuko du, non bertan, tenperatura aldatzea ahalbidetzen duen klimatizagailu bat egongo den. Zerbitzariak inguruneko tenperaturaren balioak hartu eta bezeroari bidaliko dizkio. Bezeroaren aldetik datu hauek jasotzeaz gain, inguruneko klimatizagailuari tenperatura igo edo jaisteko aginduak bidali ahalko zaizkio.

Ingurune hori ahalik eta errealen simulatzeko, kodearen diseinua bi modulo guztiz bereiztutan osatua izan da. Lehen moduluan, zerbitzaria eta simulatzailea daude inplementatua. Bertan, datuak jasotzeko "ingurune kontrolatua"simulatuko da, tenperaturaren eta klimatizadorearen simulatzaileekin. Beste moduluan aldiz, bezeroaren aldea dago, aplikazio izena duena. Hemen datuen biltegiratzea eta bistaratzeko softwareak daude inplementatuak, datuak tratatzeko aplikazio moduan lana egingo duena. Programaren diseinu nagusia 5.1 irudian ikus dezakegu.



5.2 Irudia: Egoera makina

5.1.1 Laborategi modulua

Laborategi modulua, izenak dioen moduan, ingurune kontrolatu bat simulatzen duen modulu bat da. Bertan, tenperatura simulatuko duen simulatzaile bat eta honek sorturiko datuak bidaltzeko zerbitzaria egongo dira.

Simulatzailea hasterakoan inguruneak duen adierazitako tenperaturan hasiko da. Une batean bezerotik klimatizadorera helburu tenperatura bat iritsiko da, hau jasotzean, modu leun eta ahalik eta modu errealistenean tenperatura helburu horretara gerturatzen joango da. Bitartean, OPC UAko zerbitzaria simulazio hortatik sortzen diren datuak jaso eta bidaltzen joango da.

Simulatzailearen funtzionalitatea modu automatizatu batean egiteko egoera makina bat sortu dugu. Horretarako pythoneko *trasnitions* paketea erabili da. Egoera makina honek 6 egoera dauzka: START, OFF, TARGET, CHANGING, WARMING eta COOLING, 5.2 irudian marraztuta dagoen bezala.

5.1.2 Aplikazio modulua

Aplikazio moduluan, bezeroaren, Timescale datu basearen zein Grafana datu bistaratzaileen docker konfigurazioak aurkituko dira inplementatuta. Bezeroak, inplementazio ho-

netan bi lan izango ditu: zerbitzarira konektatzea honek bidalitako datuak jasotzea, eta datu hauek datu basera igotzea.

Diseinu aldetik, aplikazio modulu guzti hau, docker edukiontzi baten barnean joango da gordeta gero honen garraioa eta inplementazioa beste edozein makinatan erraza izan dadin.

5.2 Inplementazioa

Behin egin beharreko lanaren diseinua sortua izan denean, hau inplementatzen hasteko ordua iritsiko da. Proiektuaren plangintzan aipatu bezala, inplementazioa faseka joango da burutzen hasiera batean adierazitako moduluetan bereiztuta. Inplementazioan egin beharko diren atazak modu argi eta ordenatuan burutzeko.

5.2.1 Laborategi modulua

Proiektuaren inplementazioarekin hasterakoan, lehen lan nagusia, denbora errealean datuak sortuko dizkigun simulatzaile bat sortzea izango da.

Helburu honekin, 5.2 irudian ikus dezakegun egoera makina jarraituko duen termostatu bat inplementatuko da.

Egoeran makina hau *transitions* paketearen laguntzaz burutu da. Pakete honen laguntzaz, egoerak sortu eta hauen arteko loturak modu erraz batean diseinatzeko aukera ematen du. Pakete honen beste ezaugarrietako bat, egoera aldaketa espezifiko bakoitzaren ondoren funtzio konkretu bati deia egitea da. Ezaugarri honekin amaierako kodea modu txukunago eta ulerkorragoa izaten laguntzen du.

Egoera makina honela egongo da inplementatua:

```
STATESLIST = ['start', 'warming', 'cooling', 'off', 'changing', 'target']

machine = Machine(model=self, states=STATESLIST, initial ='start')
machine.add_transition(trigger='initialize', source='start', dest='target', after=
    'select_temp_change')
machine.add_transition(trigger='start_cooling', source=['changing', 'target', 'off
    '], dest='cooling', after='temperature_change_init')
machine.add_transition(trigger='start_warming', source=['changing', 'target', 'off
    '], dest='warming', after='temperature_change_init')
```

```
machine.add_transition(trigger='target_changing', source='*', dest='changing',
    after='select_temp_change')
machine.add_transition(trigger='target_achieved', source=['warming', 'cooling'],
    dest='target', after='temperature_change_init')
machine.add_transition(trigger='power_off', source='*', dest='off', after='
    temperature_change_init')
machine.add_transition(trigger='power_on', source='off', dest='target', after='
    select_temp_change')
```

Egoera makinaren inplementazioa hurrengo modura dago sortua: lehenik *STATELIST* zerrendan, erabiliko diren egoerak sartuko dira. Zerrenda honekin egoera makinaren objetua sortuko da, jarraituko duen modeloa *self* motakoa jarri, bere burua izango dela jarraitu beharrekoa ezarri; makinak izango dituen egoerak *STATELIST* aldagaikoak izango direla, eta hasierako egoera *start* izango dela adieraziz.

Hau egin ostean, egoera makinari trantsizioak gehitu behar zaizkio. Trantsizioa sortzeko 3 derrigorrezko datu eman behar dira: *source* eta *dest*, aldaketa zein bi egoeren artean izango den, eta gero *trigger*-a, zein izango den trantsizio hau burutzeko deia. Kasu batzuetan egoera bat baino gehiagotatik iritis daiteke helmuga egoerara. Kasu honetan egoeren zerrenda bat sar daiteke edo * bat jarri daiteke egoera guztiak adierazteko.

Derrigorrezkoa ez den datua baina inplementazio honetan erabili dena, *after* datua da. Hemen trantsizioa burutu ezkero funtzio konkretu bat burutzeko deia egiten da.

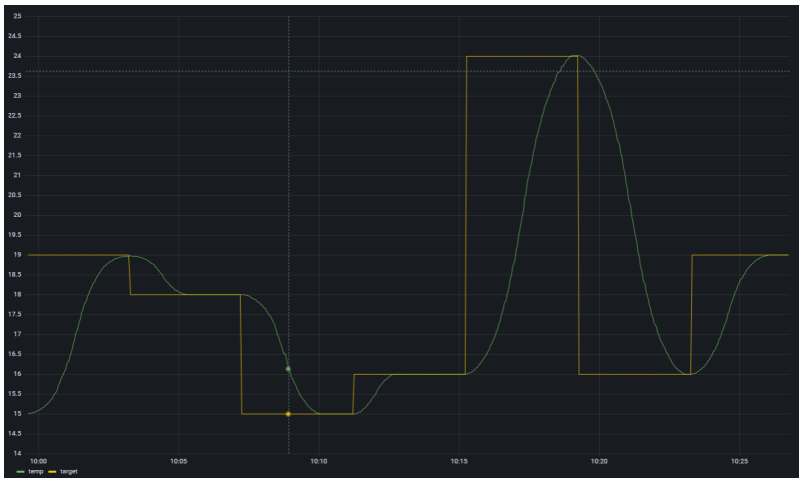
Behin egoera makina inplementatu dela, tenperatura aldaketaren kalkuluaren inplementazioa egin da.

Hasiera batean era konstante batean igo edo jaisten zen formula baten bidezko simulatzaile bat sortu zen. Ziklo bakoitzero gradu kantitate konstante bat gehitu edo kenduz orain arte zegoen tenperaturari.

Lehen bertsio bezala emaitza onargarri bat ematen zuen. Hala ere pauso bat hurrunago joan izan nahi da simulatzailean eta errealismo zantzu bat sartzen saiatu da. Hau lortzeko, zenbait kalkulu matematiko sinple gehitu dira kalkulura.

Kalkulua:

```
hurrengo_zikloa = 0.005
while tenperatura (< >) helburu_tenperatura:
  tenperatura (+−)= tenperatura_kalkulua(zikloa)
  if tenperatura > helburua + (helburua − tenperatura)/2:
```



5.3 Irudia: Tenperatura aldaketa

```
    zikloa += hurrengo_zikloa
  else:
    zikloa -= hurrengo_zikloa

def tenperatura_kalkulua(zikloa):
  if zikloa < 0:
    zikloa = 0
  return (zikloa**2.0)
```

Kalkulua burutzeko modu honekin, aldaketa leunago bat lortu dezakegu 5.3 irudian ikusi dezakegun modura.

Behin simulatzailea sortu dugula, makina-sistema komunikazioarekin hasiko gara. Horretarako, OPC UA zerbitzari bat sortuko dugu simulatzaileak sortzen dituen datuak jaso eta bidaltzen hasteko.

Pythonek erabilgarri daukan opcua paketea erabiliz zerbitzariaren funtzioak inplementatuko ditugu. OPC UA-ko zerbitzari bat sortzeko hiru lan nagusi daude: zerbitzariari IP eta portu bat esleitzea datuak bertara bidaltzeko, zerbitzariak izango duen segurtasuna hautatu eta konfiguratu, eta azkenik bidali nahi diren aldagaiak identifikatu eta hauekin nodo bat sortu.

Proba guzti hauek sistema lokal batean egiten ari garenez, zerbitzariari 0.0.0.0 helbidea esleituko zaio. Gero defektuz OPC UA-k 4840 portua dauka konfiguratuta, beraz zerbitzariaren URL-a honela geratuko zen: *opc.tcp://0.0.0.0:4840*

Segurtasuna konfiguratzeko garaian, defektuz, OPC UA-k hiru segurtasun maila konfigu-

razio eskaintzen ditu. Lehena eta sinpleena, segurtasunik gabeko komunikazioa da. Modu honetan edozein bezero konektatu ahalko da zerbitzarira eta komunikazio horretan bidalitako mezu guztiak enkriptatu gabe bidaliko dira. Bigarrena, konexioa sortzerakoan konektatu nahi den makinari bere ziurtagiri digitala eskatuko dio eta hirugarrengoa, ziurtari digitalaz gain, enkriptazio maila bat eskatuko du. Gure proba kasuetan lehena eta sinpleena erabili da, inongo segurtasunik gabe.

Azkenik, zerbitzariak simulatzaileko datuak bidali beharko ditu. Horretarako Nodo bat sortu beharko da simulatzaileak dituen datuekin. Lehen azaldu den bezala, OPC UA-k datuak Nodo izeneko datu egituratan bidaltzen ditu. Nodo hauek atributu bakar batetik nahi adinara izan ditzakete. Gure kasuan Thermostat izeneko nodo bat sortu dugu, non honen barruan, uneko tenperatura, helburu tenperatura, egoera makinaren uneko egoera, termostatoaren egoera(piztuta edo itzalita dagoen), eta ingurunearen tenperatura maximoa eta minimoa egongo diren.

```
node = server.get_objects_node()
custom_obj_type = node.add_object_type(id, "Thermostat")

temp = custom_obj_type.add_variable(0, "Temperature", 0.0)
temp.set_modelling_rule(True)
target = custom_obj_type.add_variable(1, "Target", 0.0)
target.set_modelling_rule(True)
target.set_writable()
state = custom_obj_type.add_variable(2, "State", 0.0)
state.set_modelling_rule(True)
power = custom_obj_type.add_variable(3, "Power", 0.0)
power.set_modelling_rule(True)
power.set_writable()
max_temp = custom_obj_type.add_variable(4, "Max_temp", 0.0)
max_temp.set_modelling_rule(True)
min_temp = custom_obj_type.add_variable(5, "Min_temp", 0.0)
min_temp.set_modelling_rule(True)
```

Zerbitzariko kode honetan ikus dezakegun bezala, *node.add _object_type()* agindua erabiltzen da nodo berri bat sortzeko.

Agindu honekin, guk esleitutako aldagaiak dituen nodo mota bat sortuko dugu. Modu honetara nodo honetatik nahi adina instantzia sortu ahalko dira, gerora zerbitzari beretik mota bereko beste termostatorik kontrolatu nahi bada.

Zerbitzaria abiarazi aurretik, sortu berriko nodo motaren instatzia bat sortu behar da.

Amaitzeko, begizta baten laguntzaz datuak bidaltzen jarriko dira. Zerbitzari honen kasuan, nodoko bi datu *writable* bezala sortu dira, honek, bezerotik datua aldatzea ahalbidetuko du. Horregatik, datuen bidalketa begiztan datu hauek bidali ordez, jaso egingo dira.

```
target_change = object.target
power_change = object.power
my_target.set_value(target_change)
my_power.set_value(power_change)

while True:
  my_temp.set_value(object.temp)
  my_state.set_value(object.state)
  my_max_temp.set_value(object.temp_max)
  my_min_temp.set_value(object.temp_min)

  if my_target.get_value() != target_change:
    target_change = my_target.get_value()
    my_target.set_value(target_change)
    object.target = target_change

  if my_power.get_value() != power_change:
    power_change = my_power.get_value()
    my_power.set_value(power_change)
    object.power = power_change
```

5.2.2 Aplikazio modulua

Aplikazio modulua inplememtatzerakoan, lehen lana, OPC UA bidez zerbitzariak publikatzen dituen datuak jasoko dituen bezero bat sortzea da. Hau egiteko, bezeroak zerbitzariaren IP helbidea eta hau konektatuta dagoen portua jakin behar ditu.

Sistema lokal batean lanean ari garenez, IP helbidea *127.0.0.0* izango da, eta portua zerbitzarian adierazi den bezala 4840.

Bezero/zerbitzari konexioa egin nahi denean, kontuan hartu beharko da zerbitzariak duen segurtasun maila. Konexioa ezartzerako garaian behar bada ziurtagiri edo sinadura digitala eskatuko baitigu. Gure kasuan ez da horrelako ezer eskatzen, beraz, zerbitzariaren IP helbidea eta portura konexio eskaera egin ezkero inolako arazorik gabe konexioa ezarriko da.

Konexioa ezartzerakoan, bezeroak zerbitzariak bidaltzen dituen datu guztiak izango ditu eskura, beraz momentu honetan interesatzen zaizkigun datuetara harpidetu ahalko gara.

Gure kasuan nodo bakarra bidaltzen du zerbitzariak eta bertan nahi ditugun datu gauztiak daudenez guztietara harpidetuko gara. Honetarako, datu bakoitzaren *ID* balioak jakin beharko ditugu.

```
temp = client.get_node('ns=2;s=V1.Temperature')
state = client.get_node('ns=2;s=V1.State')
target = client.get_node('ns=2;s=V1.Target')
power = client.get_node('ns=2;s=V1.Power')
```

Datuak jasotzerako garaian, **??** atalean azaltzen den bezala, OPC UAk datuak jasotzeko zenbait modu dauzka. Sinpleena, uneko datuak eskuratzea da, kodean begizta baten laguntzaz une oro nodoak duen balioa eskuratuz.

Begizta:

```
while True:
  temp_val = temp.get_value()
  state_val = state.get_value()
  target_val = target.get_value()
  power_val = power.get_value()
  insert_values(temp_val, state_val, target_val, power_val)
```

Datuak lortzeko beste metodo bat, nodoetara harpidetzea da. Metodo honekin, nodoak aldaketaren bat jasaten duenean guk jarritako ekintza bat burutuko du.

Kodea honela geratuko zen:

```
class DataHandler(object):
  def datachange_notification(self, node, val, data):
    if str(node) == 'ns=2;s=V1.Temperature':
      global temp_val
      temp_val = val
    elif str(node) == 'ns=2;s=V1.State':
      global state_val
      state_val = val
    elif str(node) == 'ns=2;s=V1.Target':
      global target_val
      target_val = val
    elif str(node) == 'ns=2;s=V1.Power':
      global power_val
      power_val = val
    insert_values(temp_val, state_val, target_val, power_val)

handler_temp = DataHandler()
sub_data = client.create_subscription(500, handler_temp)
handle_data = sub_data.subscribe_data_change(temp)
handle_data = sub_data.subscribe_data_change(state)
handle_data = sub_data.subscribe_data_change(target)
handle_data = sub_data.subscribe_data_change(power)
```

Bigarren metodo honekin, datuak datu basera beharrezkoak direnean bakarrik igoko dira, daturen baten balioa aldatzerakoan hain zuzen ere. Bestela, lehen metodoarekin, begiztaren iterazio bakoitzero datuak datu basera igoko dira naiz eta aldaketarik ez jaso.

Behin datuak jasota, datu basera bidaltzea geratuko zen. Horretarako TimescaleBD datu basea erabiliko da, eta honen gainean kontrol gehiago izateko bere docker-eko irudia erabiliko da. Datu basearekin batera Grafana datu bistaratzailearen irudia ere erabiliko da docker barruan.

Dena leku berdinetik kontrolatzeko, Docker-ek docker compose izeneko metodo bat dauka. Bertan, irudi guztien eraikuntza fitxategi beretik egin eta edukiontzi guztien arteko komunikazioak modu erraz batean egitea ahalbidetzen du.

Docker compose fitxategiak sintaxi berezi baten bidez sortzen da. Bertan zerbitzu moduan behar ditugun irudiak eta bakoitzaren datuak sartzen dira. Timescale-ren kasuan hurren-

goa izango zen bere sintaxia:

```
timescale:
  image: timescale/timescaledb:2.6.1-pg14
  container_name: "timescale"
  ports:
   - 5432:5432
  volumes:
   - ./client_files/docker/timescale/initialize.sh:/docker-entrypoint-initdb.d/init
    -user-db.sh
  environment:
   - POSTGRES_USER=postgres
   - POSTGRES_PASSWORD=password
   - POSTGRES_DB=thermostat
  networks:
   - gateway
```

Ikus daitezken hamahiru lerroak osatzen dute docker edukiontzi bat. Lehenik guk ulertzeko izen bat jarri behar zaio edukiontziari, gure kasuan aplikazioaren izen berdina jarri diogu: *Timescale*. Honen barruan, aplikazioaren irudia dago adierazia, **image**, kasu honetan ez denez irudiko ezer aldatu, zuzenean agertzen da. Irudia eta honen bertsioa: *timescale/timescaledb:2.6.1-pg14*

Hurrengo puntua, edukiontziaren izena da, **container_name**. Datu hau derrigorrezkoa izan ez arren, gero Docker barruko edukiontzien arteko komunikazioa baldin badago, lan asko errazten ditu. Izen honi erreferentzia egin ezkero beste edukiontzi batetik, honen IP helbidea automatikoki itzultzen baitu. Edukiontziaren izena: *timescale*.

Portuaren datua, **ports**, izenak berak dioen bezala, edukiontziari portu bat esleitzeko erabiltzen da. Dockerren kasuan, edukiontzi kanpoko : edukiontzi barneko portua. portua: *5423:5432*.

Bolumen datuarekin, **volumes**, gure makinako direktorio edo fitxategiak edukiontzi barrura kopiatzea ahalbidetzen digu. Timescale edukiontziaren kasuan, datu basea hasieratzean sortu behar dituen taulen aginduak pasatzen zaizkio, honela edukiontzia martxan jartzen den bakoitzean datu basea gure ezaugarriekin hasieratuko da automatikoki. Bolumena: *direktorio-lokala:edukiontzi-barneko-direktorioa.*

Environment atalean, edukiontziaren irudia eraikitzean onartzen dituen kanpo aldagaiak sartzen dira. Alderdi honetan kasu honetan, datu basearen kredentzialak eta izena sartu

ditugu.

Azkenik, edukiontzia docker compose barruan ea sare espezifikoren batean dagoen jar daiteke, **networks**. Gure kasuan edukiontzi guztiak *gateway* sarearen barruan egongo dira sartuak, elkarren arteko komunikazioak errazteko.

Docker-compose.yml fitxategia honela geratu da edukiontzi guztien informazioa sartuta:

```
services:
 timescale:
  image: timescale/timescaledb:2.6.1-pg14
  container_name: "timescale"
  ports:
   - 5432:5432
  volumes:
   - ./client_files/docker/timescale/initialize.sh:/docker-entrypoint-initdb.d/init
    -user-db.sh
  environment:
   - POSTGRES_USER=postgres
   - POSTGRES_PASSWORD=password
   - POSTGRES_DB=thermostat
  networks:
   - gateway

 grafana:
  build:
   context: ./client_files/docker/grafana
   dockerfile: Dockerfile
  ports:
   - 3000:3000
  volumes:
   - ./client_files/docker/grafana/provisioning/:/etc/grafana/provisioning/
  environment:
   - GRAFANA_DATA_HOST=timescale:5432
  networks:
   - gateway

 client:
  build:
   context: ./client_files/docker/cliente
   dockerfile: Dockerfile
  volumes:
```

```
    - ./client_files/docker/cliente/:/usr/app/src/
  networks:
    - gateway

networks:
 gateway:
```

Aurreko azalpenarekiko ikus daiteken diferentzia bat, *grafana* eta *client* edukiontzietan, irudia Docker-en zerbitzaritik zuzenean hartu beharrean, "eraiki"egiten dugu.

Kasu honetan, *image* atala izan beharrean, ***build*** atala daukagu. Honen barruan, *context* zatian, edukiontziaren irudia non aurkitzen den azaldu behar da, eta *dockerfile* zatian, nola izena duen irudia dagoen fitxategiak.

Docker edukiontzi baten deskribapena egiteko *Dockerfile* bat idatzi behar da. Fitxategi honek sintaxi espezifiko bat eskatzen du Docker-ek ulertzeko agindu eta deiekin.

```
FROM python:3.9

WORKDIR /usr/app/src

RUN pip install --no-cache-dir --upgrade pip && \
  pip install --no-cache-dir -r requirements.txt

CMD ["python", "./client.py"]
```

Hau izango zen gure bezeroak erabiliko duen irudia sortzeko kodea. Lehen agindua, **FROM**, zein iruditan oinarritu den jartzeko da, kasu honetan *python 3.9* instalatua duen linux ingurune minimo bat. Gero zein direktoriotan kokatu esaten zaio **WORKDIR** aginduarekin, kasu honetan gure bezeroaren fitxategiak utzi ditugun direktorioan. Ondoren, **RUN** aginduarekin, gure bezeoak behar dituen paketeen zerrenda pasaz, hauek *pip* bidez instalatzeko esaten diogu. Eta amaitzeko, **CMD** aginduaren bitartez, irudia exekutatzen den bakoitzean, zein agindu exekutatu adierazi diezaiokegu. Kasu honetan, gure bezeroaren python fitxategia martxan jartzea.

Grafanaren docker irudia honelakoa da:

```
FROM grafana/grafana-oss:8.4.0

ENV GF_AUTH_DISABLE_LOGIN_FORM "true"
```

```
ENV GF_AUTH_ANONYMOUS_ENABLED "true"
ENV GF_AUTH_ANONYMOUS_ORG_ROLE "Admin"
```

Kasu honetan, ingurune aldagaiak bakarrik pasatzen zaizkio **ENV** aginduarekin.

5.3 Inplementazioa MQTT bidez

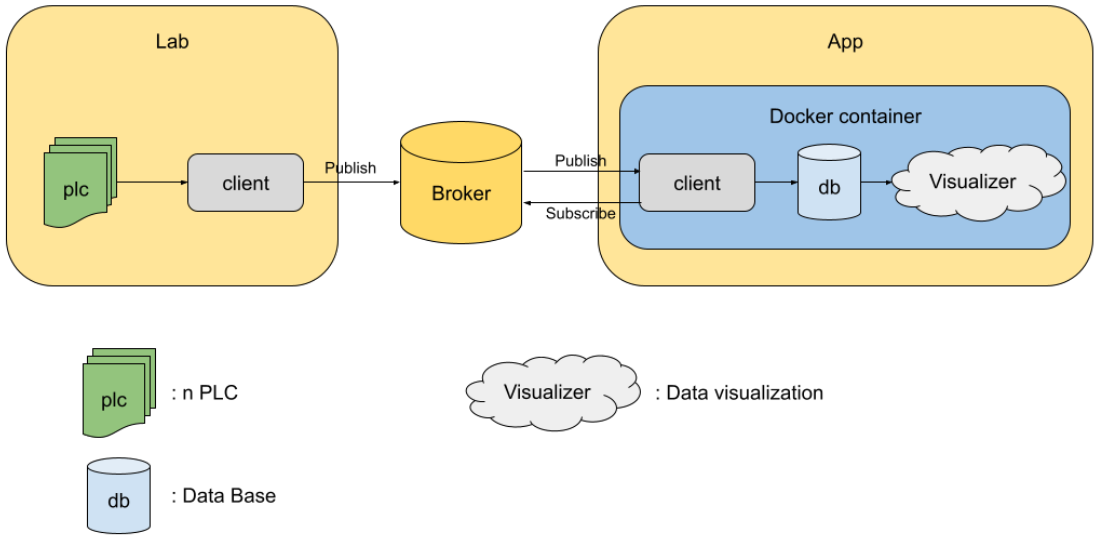
Orain arte, proiektuaren diseinu zein inplementazioa OPC UA bidez nola egin den azaldu da. Proiektuaren helburuak azaldu direnean, sektorean erabiltzen diren komunikazio protokolo ezberdinak konparatuko zirela adierazi da. Premisa honekin jarraituz, azkenaldian indar handia hartzen hari den MQTT komunikazio protokoloa erabili da proiektuaren bertsio berri bat egiteko.

5.3.1 Diseinua

Proiektuaren bertsio honetan, inplementazioak izango duen diseinua zerbait aldatu behar da OPC UAk erabiltzen duenarekiko. Aurreko bertsioan erabiltzen ziren bi moduluak hartu dira erreferentzi bezala, bakoitzeko elementuak bertan mantenduz. Lehen modulua, laborategi moduluak, egitura berbera izango du, protokolo honetan simulatzailera konektatzen den zerbitzari bat izan beharrean, bezero publikatzaile bat izanik. Bigarren moduluan aldiz, aplikazio moduluak, egitura berbera jarraituko du izaten, OPC UAko bezeroa izan beharrean MQTTko bezero harpidetu bat izanik bertan inplementatua egongo dena. Diseinuan benetan aldaketa suposatuko duen elementu bakarra, bi moduluen erdian kokatuko den Broker egiturarena izango da, bi moduluen arteko komunikazioa gauzatzea ahalbidetuko diguna, 5.4 irudian ikus daiteken bezala. Broker hau zerotik diseinatu eta inplementatu beharrean, kode libreko Eclipse Mosquitto broker-a erabiliko da non MQTT 5.0 bertsioa erabiltzen duen.

5.3.2 Inplementazioa

MQTT bidez eginiko inplementazio honetan, komunikazio protokoloa soilik aldatu da, diseinuan aipatu den bezela. Horregatik, aurreko bertsiotik simulatzailea, datu basea eta datu bistaratzailea berreskuratu dira. Modulu bakoitzean bezero berriak inplementatu dira



5.4 Irudia: Proiektuaren diseinua MQTT erabiliz

bakoitza bere rolarekin, laborategi moduluan bezero publikatzailea eta aplikazio moduluan bezero harpidetua hain zuzen ere.

Diseinua kontuan hartuta, laborategi moduluan simulatzailea eta bezero bat aurkitzen dira. Bezero honek simulatzaileak sorturiko datuak hartu eta broker-era bidaltzen ditu. Hau egiteko, bezeroan broker-era konexio bat ireki beharko da. Mosquito broker-a, Docker edukiontzi baten barnean dago martxan, beraz konexioa egiteko, edukiontzi honen helbidea eta irekita daukan portua behar ditugu konexio hau irekitzeko. Hurrengo agindu bidez irekitzen da konexio bat MQTT broker batekin pythonen:

```
client =mqtt.Client('stateMachine_client_publisher')
client.connect(broker_address, 1883, 60)
```

Behin broker-ak konexioa onartzen denean, datuak publikatzen hasi ahalko da bezeroa, horretarako aurretik aipatu den bezala bezeroa publish aginduak bidaltzen joango da berak dauzkan datuak bidaltzeko.

```
client.publish('TH/id', id)
client.publish('TH/temp', temp)
client.publish('TH/state', state)
client.publish('TH/target', target)
```

Laborategi modulu honetan bezeroak broker-arekin konexioa nola sortu eta datuak no-

la bidali diren ikusita, aplikazio modulua nola inplementatu den ikustera pasako gara. Modulu honetan aurrekoan bezala, gauza oso gutxi aldatu dira aurreko bertsioarekin konparatuta. Komunikazio sistema aldatu denez, moduluko komunikazio elementuak jasango ditu aldaketak. Aplikazioaren kasuan, bertan dagoen bezeroa da komunikazio sistemako elementua, eta MQTTren bezeroen artetik, bezero harpidetua izango da bertan inplementatuko dena.

MQTT-ko bezero harpidetu bat inplementatzeko, lehenengo konexio bat ireki behar da broker-arekin, hau, bezero publikatzaileak egiten duen moduan egiten du, bezeroari id bat eman eta honekin MQTT bezero bat sortuz lehenengo. Hau egin ostean broker-aren helbidea eta portua ezagututa bezeroa eta broker-aren arteko konexioa egin ahalko da.

Behin konexioa irekita, brokerreko datuetara harpidetzea izango da hurrengo pausoa. Horretarako, MQTT-k integratua duen ***on_connect*** funtzioa erabili beharko da. Funtzio hau broker-arekin konexioa egiten denean jaurtitzen da, beraz funtzio honen barruan egingo dira bezeroak behar dituen aldagaietara harpidetzak. Broker-ean aldagaiak bilatzeko garaian, kontuan izan behar da aldagai hauek zuhaitz eredua jarraituz gordetzen direla broker barruan, beraz, jakin egin beharko dugu guk nahi dugun aldagaia edo aldagai taldea non dagoen zehazki mapeatuta. Gure termostatuaren kasuan, aldagai denak ***TH*** aldagai gurasoaren umeak dira, beraz, TH aldagai osora harpidetu ahalko gera.

```
def on_connect(client, userdata, flags, rc):
  client.subscribe('TH/#')
```

Kodean ikus daiteken '#' sinboloarekin, TH aldagai barneko aldagai ume guztietara harpidetzen dela adierazten da.

Gure aplikazioak erabiliko dituen datuetara harpidetu ostean, hauen datuen aldaketak jasotzean egin beharko dena adierazi behar da. Horretarako ere MQTT-k funtzio bat dakar aurredefinitua ***on_message*** izenarekin. Funtzio hau, harpidetuta gauden aldagairen batean aldaketarik jasotzen badugu izango da deitua, beraz, funtzio honen barnean sartu beharko dira datu hauekin egin nahi diren aldaketa eta eragiketa guztiak. Gure aplikazioaren kasuan, datu hauek datu basera igoko dira. Hau egiteko, lehendabizi filtratu egin beharko dira aldagaiak eta gero datu basera igo. Filtro hau oso sinplea da, izan ere, broker-ak aldagaiaren datua String batean bidaltzen du zein aldagaietatik datorren aurre adierazlearekin. String honetatik datua lortzeko, filtro sinple bat aplikatu beharko da:

```
def on_message(client, userdata, msg):
```

```
    print('Message arrived')
    clasify_values(msg.topic, msg.payload)

def clasify_values(topic, payload):
    global data_list
    if topic == 'TH/id':
        x = str(payload).split("\'")
        data_list.id = x[1]
    elif topic == 'TH/temp':
        x = str(payload).split("\'")
        data_list.temp= x[1]
    elif topic == 'TH/state':
        x = str(payload).split("\'")
        data_list.state = x[1]
    elif topic == 'TH/target':
        x = str(payload).split("\'")
        data_list.target = x[1]
    insert_value(data_list.id, data_list.temp, data_list.state, data_list.target)
```

Harpidetutako aldagaian aldaketak jasatean zer egin behar duen adierazi ostean, aplikazoia hasieratzea bakarrik geratuko zen. Hasieran esan den moduan broker-arekin konexioa zabaltzeko bezero bat sortu behar da ID bat emanda bezero bakoitzari. Behin bezeroa sortu dela *on_connect* eta *on_message* funtzioak lotuko dizkiogu konexioa sortu aurretik.

6. KAPITULUA

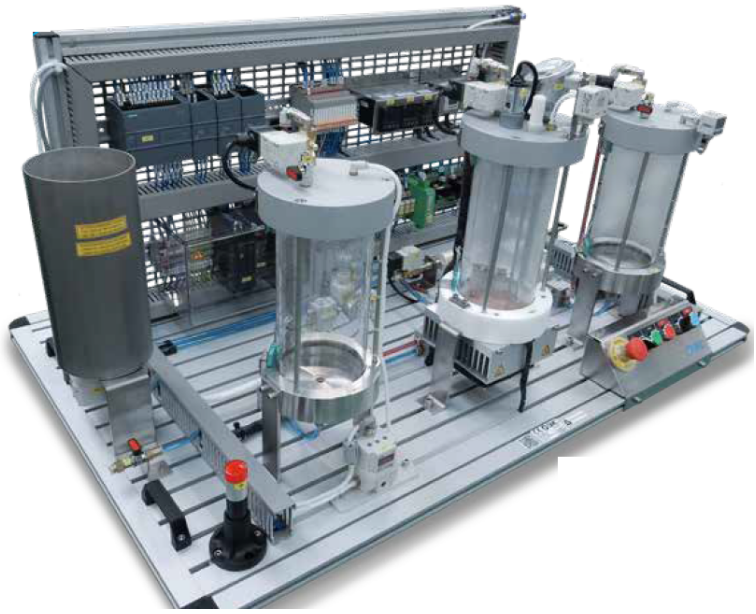
Baliozkotzea

Proiektu hau sortua izan zenean, bi fase izango zituela aurreikusi zen. Lehen fasea simulatzaile batek sortutako datuak erabiliz funtzionatzen duen software bat sortzen oinarritu da. Fase honetan, erabili ditugun teknologia guztien bertsio ezberdinak, funtzionalitateak, ... probatu ditugu ahalik eta software egokiena sortzeko guk nahi genuen helburua betetzeko. Bertsio ezberdinak pasa ostean, 5 fasean azaldu den diseinu eta inplementaziora iritsi da softwarea. Diseinu horrekin hasieran aipatutako lehen fase honen amaierara irtixi da proiektua. Hurrengo pausoa, sorturiko datuen biltegiratze eta bistaratze plataforma, industrian erabiltzen den makina erreal batean inplementatzea izango da. Fase hau aurrera eramateko Vicomtech-eko, Energia eta Industria Prozesuetarakoko Datu Adimen saileko laborategian dagoen SMC IPC-201C maquina erabili da.

6.1 SMC IPC-201C

Erabilitako makina hau SMC International Training®enpresak komertzializatzen du. SMC® enpresaren dibisio hau, enpresaren alde didaktikoaz arduratzen da. Bertan industrian erailtzen diren makinen bertsio txikiak egiten dituzte, gero hirugarren zentroetan ikasketa edo froga makina bezala erabiltzeko. Merkaturatzen dituzten makinak sektoreko edozein makinak izan ditzaken elementu berberak izaten dituzte, baina bertsio txikiago bat denez edozein arazori konponbidea bilatzeko lana eta kostua asko murrizten da, beraz perfektuak dira probak egiteko.

Proiektu honen baliozkotze faseari dagokionez, SMC IPC-201C makina erabili da. Ma-



6.1 Irudia: IPC-201C. Iturria: smc.eu

kina hau, hiru moduluz osatuta dagoen likido enboteilatze SMC IPC-200 linearen lehen modulua da. Modulu hau bi likido depositu, depositu hauetan dauden likidoak nahasteko hirugarren depositu zentral bat eta likido extra gordetzen duen deposito auxiliar batez osatzen da. Lau depositu hauetaz gain, likidoa mugitzeko bonba bat, presiopeko airearekin funtzionatzen duen bonba bat, deposituen maximo zein minimoak kontrolatzen dituen sentsoreak eta depositu zentralaren tenperatura, likidoaren sarrera/irteera isuria eta deposituaren maila kontrolatzeko sentsoreak ere ditu. 6.1 irudian ikus daiteke makina.

6.2 Inplementazioa

Orain arte eginiko lana ingurune erreal batean inplementatuko da, diseinuaren modularitatea eta egokitasuna balioztatzeko. Horretarako, lehenik inplementazioa egin nahi den sistemaren azterketa bat egin behar da. Azterketa honetan, sistematik jaso nahi den funtzionamenduaren eta honen sentsore edo aldagaiak aztertu beharko dira. Azterketa hau egiteko, makinaren dokumentaziora jo beharko da, aldagaien eta sentsoreen konexio helbideak lortzeko.

Gure softwareko bezeroa, datu basea eta datu ikuskatzailea hartuko ditugu baliozkotze hau aurrera eramateko. Makinak bere OPC UA zerbitzaria inplementatua darama, behin analizatu eta gorde nahi diren aldagaien helbideak jakinda, OPC UA-ko bezeroa moldatu

beharko da aldagai hauetara. Hau egin ostean, datu baserako konexioa berritu beharko da. Datu basearen aldean, lehenik eta behin datu base berri bat sortu beharko da nahi diren taulekin. Behin hau eginik, datu bistaratzaileak sarbidea izango du gordetzen ari diren datu hauetara.

Aurretik azaldu den bezala, IPC-201C makina likidoak nahasteko makina bat da. Honen harira, bere dokumentazioa irakurri ezkero bere funtzionamendua kontrolatzeko aldagaiak hurrengoak dira: likido balbulak, likido mailaren sentsoreak eta ur bonbak. Hiru sentsore hauekin, guk nahi dugun funtzionamenduaren kontrol guztia modu egoki batean egin ahalko da.

Behin analizatu nahi ditugun aldagaiak identifikatu ditugula, hauek jasotzeko datu base bat sortzea izango da lehen lana. Puntu honetan, jasotzen diren datuak 2 taulatan banatzea erabaki da. Batean, kontrolaren eta makinaren egoeraren informazioa sartzeko izango da eta beste taula bat bertatik lortzen diren datuak jasotzeko. Datu basearen egitura sortua izan dela, makinaren zerbitzariarekin konexioa sortzea izango da hurrengo lana. Ataza hau oso sinplea da behin eskuratu nahi diren aldagaiak zein diren jakinaren gainean daudenean. Aldagai bakoitzaren identifikadorea erabiliz, makinak barnean sartua duen OPC UA zerbitzarira konektatu eta aldagaietara harpidetzea bakarrik izango da.

Datuen bistaratzeari dagokionez, aplikazio guztiak Docker barnean daudenez, elkar konexioa jadanik eginak daude. Aldatu beharreko parametro bakarra datu base berriari emaniko izena izango da, modu honetan konexioaz gain zein datu base hartu beharko duen esango baitiogu. Puntu honetan, datuei zein analisi egin behar zaien analizatu beharko da datu bistaratzeileko taulak eta grafikoak sortzeko. Hau egiteko, datuen analisia egingo duen profesioanalarekin bat egitea da optimoena, bera izango baita gerora datu hauek ikusi eta analizatu beharko dituena.

7. KAPITULUA

Ondorioak eta etorkizunerako lana

Proiektuaren helbuuru nagusia guk sorturiko softwarearen laguntzaz hirugarren pertsonen lana erraztean oinarritzen zen, prozesu industrialetan sortzen ziren datu masiboen arazketa eta lehen analisi bat aplikatuz. Alderdi honetan, proiektuaren emaitza eta funtzionamendua ikusiz, helburu horretara iritsi dela esan daiteke, eginiko lana bertan martxan zebilen makina batean funtzionatzen geratu baita eta bertan lan egin behar duen hurrengoari lana erraztuaraziko baitio.

Helburu hauen barnean, proiektuaren modularitatea eta zabaltzen erraza izatea ere tarteko aurkitzen zen, softwarea sistema edo makina ezberdinetara konektatu eta funtzionatzen jartzea ahalik eta errazena izan zedin. Puntu honetan, Vicomtech-ek industrian duen esperientziaz baliatuta, industria makinen komunikazio protokolo ezberdinetara moldatu da proiektua, softwareak ahalik eta makina gehienetan erabili ahal izateko. Bertan OPC UA eta MQTT protokoloetara moldatu dugu proiektua. Bestalde, sistema instalatua joango den makinei erreparatuta, Docker Container-ei esker, gure proietua edozein motako sistemetan jarri ahalko da funtzionatzen, aldatu beharko den zati bakarra makinatik jaso nahi diren datuak izanik.

Azkenik, proiektuaren helburu eta lan nagusiena, sorturiko softwarea benetako sistema batean probatu eta balidatzea zen. Puntu honetan Vicomtech-en laborategietan zegoen industria makina baten maketa batean probatu eta testeatu ahal izan da software hau. Hasiera batean simulatzailean eginiko proba guztiak benetako eremu eta datuekin proban jarriz. Puntu honetan makinarekin arazoak izan arren softwarearen funtzionamendua egokia izan da eta jarritako proba eta lanei modu onean egin die aurre.

Hau dena azalduta, ikus daiteke proietu honetan ezarritako helburuak modu onean bete direla.

Proiektuarekiko ezarrita zeuden helburuez gain, pertsonalki informatikari bezala jakinduriak handitzeko aukera eman dit. Dimentsio handiko proiektu bat izanik garbi geratu zait ezinbesteko dela proiektu baten aurre prestaketa, buruan dagoen ideia martxan jarri aurretik nola egingo den eta bertan gerta daitezken arazoak aurretik identifikatuak izatea. Bestetik, egin beharreko kodea ongi estrukturatua egon behar duela ere ikasitako zati garrantzitsu bat izan da. Enpresa batean lanean ibilita, kode bera beste lankideek irakurtzeko eta ulertzeko ahalik eta errazena uztea ia kodeak ongi funtzionatzea bezain garrantzitsua da.

Hasieran aipatu bezala, proiektu hau Vicomtech Fundazioan egin da bertako Industria eta Energiarako datuen prozesamentu sailean. Proiektua enpresa bateko talde baten barne egin izanak, honen funtzionamendua nolakoa den ikasteko aukera eman dit, proiektuetan taldeka nola funtzionatu ikasi eta gainetik benetako enpresa baten funtzionamendua nolakoa den ikasteko.

Amaitzeko, helburuen ondorioetan aipatu bezala, softwarearen balidazioa ez zen nahi bezain egokia izan probak egiten ari ginen makinak izaniko arazoengatik. Horregatik, proiektu honek etorkizunean izan dezaken lan bat makina berdinean proba ezberdin gehiago sortzea izan daiteke.

Bibliografia

[tim,] Timescale docs. https://docs.timescale.com/timescaledb/latest/#welcome-to-the-timescaledb-documentation. Accessed: 2022-03-10.

[Cavalieri and Chiacchio, 2013] Cavalieri, S. and Chiacchio, F. (2013). Analysis of opc ua performances. *Computer Standards & Interfaces*, 36(1):165–177.

[Leitner and Mahnke, 2006] Leitner, S.-H. and Mahnke, W. (2006). Opc ua–service-oriented architecture for industrial applications. *ABB Corporate Research Center*, 48(61-66):22.

[Mahnke et al., 2009] Mahnke, W., Leitner, S.-H., and Damm, M. (2009). *OPC unified architecture*. Springer Science & Business Media.

[Soni and Makwana, 2017] Soni, D. and Makwana, A. (2017). A survey on mqtt: A protocol of internet of things(iot).

[Turnbull, 2014] Turnbull, J. (2014). *The Docker Book: Containerization is the new virtualization*. James Turnbull.

[Ware, 2019] Ware, C. (2019). *Information visualization: perception for design*. Morgan Kaufmann.