



REPUBLIKA SLOVENIJA
MINISTRSTVO ZA DIGITALNO PREOBRAZBO



NAČRT ZA
OKREVANJE
IN ODPORNOST



Financira
Evropska unija
NextGenerationEU

Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji

Tehnična in uporabniška dokumentacija

Naziv projekta	Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji
Status dokumenta	Prva verzija
Datum izdelave dokumenta	24. 4. 2024
Datum zadnje spremembe	10. 6. 2024
Verzija dokumenta	1.0
Avtor dokumenta	Marko Bajec, Jernej Cvek, Matjaž Pančur, Gregor Burger, Franc Drobnič, Andrej Kos

Zgodovina dokumenta

<i>Datum</i>	<i>verzija</i>	<i>avtor</i>
24. 4. 2024	0.1	Jernej Cvek
25. 4. 2024	0.9	Jernej Cvek
10. 6. 2024	1.0	Gregor Burger

Kazalo vsebine:

1	Video predstavitev tehničnega in uporabniškega vidika pilota ekosistema interneta stvari z algoritmičnimi orodji	7
2	Uvod	8
2.1	Namen dokumenta	8
3	Zahteve	8
3.1	Zahteve naročnika	8
3.1.1	Funkcionalnosti platforme	8
3.1.2	Osrednji sestav - Splošne tehnične zahteve pilota	9
4	Arhitektura/ Visokonivojski pogled	11
5	Arhitektura/ Gradniki	13
5.1	Chirpstack	13
5.2	Dodatne informacije	13
5.3	Context Broker	14
5.3.1	Predstavitev	14
5.3.2	Dodatne informacije	14
5.4	Grafana	14
5.4.1	Predstavitev	14
5.4.2	Dodatne informacije	15
5.5	IoT Agent Manager	15
5.5.1	Predstavitev	15
5.5.2	Dodatne informacije	15
5.6	IoT agenti	15
5.6.1	Predstavitev	15
5.6.2	Ultralight	16
5.6.3	LoRa	16
5.6.4	OPC-UA	16
5.6.5	JSON	16
5.6.6	Dodatne informacije	16
5.7	Keycloak	16
5.7.1	Predstavitev	16
5.7.2	Možnosti integracije	17
5.7.3	Dodatne informacije	17
5.8	Kong	17

5.8.1	Predstavitev	17
5.8.2	Različice Kong-a	18
5.8.3	Dodatne informacije	19
5.9	MongoDB	19
5.9.1	Predstavitev	19
5.9.2	Dodatne informacije	19
5.10	Mosquitto	19
5.10.1	Predstavitev	19
5.10.2	Dodatne informacije	20
5.11	Quantum Leap	20
5.11.1	Predstavitev	20
5.11.2	Dodatne informacije	20
5.12	Redis	20
5.12.1	Predstavitev	20
5.12.2	Dodatne informacije	20
5.13	TimescaleDB	21
5.13.1	Predstavitev	21
5.13.2	Dodatne informacije	21
6	Zunanji viri podatkov	22
7	Namestitev	23
7.1	Sistemske zahteve	23
7.1.1	Testno okolje	23
7.2	Seznam uporabljenih gradnikov	23
7.2.1	Seznam Docker slik	23
7.2.2	Seznam odvisnosti	25
7.2.3	Seznam ranljivosti	25
8	Konfiguracija omrežja	27
8.1	Seznam omrežnih vrat	27
8.2	Postopek namestitve	28
8.2.1	Predpogoji	28
8.2.2	Izvorna koda za namestitev platforme	29
8.2.3	Wildcard DNS zapis	29
8.2.4	Inicializacija platforme	29
8.2.5	Zagon platforme	30
8.2.6	Upravljanje platforme	30
9	Docker Swarm mode	31

10	Varnost	32
10.1	Varnost prenosnega sloja	32
10.1.1	Kriptografski sistemi	32
10.1.2	Kriptografija javnega ključa	32
10.1.3	Digitalno potrdilo javnega ključa	32
10.1.4	HTTPS	33
10.1.5	MQTTS	33
10.1.6	LoRaWAN	33
10.1.7	OPC-UA	34
10.2	Zaščita podprtih protokolov/ HTTPS	34
10.2.1	Avtomatska preusmeritev HTTP -> HTTPS	35
10.3	Zaščita podprtih protokolov / MQTTS	35
10.3.1	MQTTS v Kong-u	35
10.3.2	Primer konfiguracije	36
10.4	Zaščita podprtih protokolov / LoRaWAN	37
10.4.1	Varnost v omrežju LoRaWAN	37
10.4.2	Chirpstack Gateway Bridge	38
10.4.3	UDP	39
10.5	Zaščita podprtih protokolov / OPC UA	40
10.5.1	Varnost v OPC UA	40
10.5.2	Povezovanje OPC UA agenta IoT	40
10.6	Upravljanje digitalnih potrdil	41
10.6.1	Let's Encrypt	41
10.6.2	Self-signed certifikati	41
10.7	Upravljanje identitet in dostopov	46
10.7.1	Osnovni pojmi	46
10.7.2	Upravljanje identitet in dostopov (IAM)	47
10.8	Upravljanje identitet in dostopov	48
10.8.1	Kong + Keycloak	48
10.8.2	Kong pep-plugin	48
10.8.3	Keycloak	50
10.8.4	Zaščita dostopa do izpostavljenih gradnikov	52
11	Primeri uporabe	53
11.1	Onboarding	53
11.1.1	Onboarding naprav IoT	53
11.1.2	Registracija skupine naprav	53

11.1.3	Autoprovisioning	54
11.1.4	Registracija naprave IoT	54
11.1.5	Testna meritev	55
11.1.6	Ostale operacije	56
11.2	Onboarding zunanjega agenta IoT	56
11.3	Registracija zunanjega agenta IoT	56
11.4	Pridobivanje offline token-a	56
11.5	Nastavitev avtentikacije na agentu IoT	58
11.6	Uporaba offline token-a ob registraciji skupine naprav	59
11.7	Posodobitev offline token-a za skupino naprav	59
11.8	Preklic dostopa	60
11.9	Omejevanje dostopa skupine naprav	60
12	Uporaba orodja Grafana	61
12.1	Dostop do nadzorne plošče v orodju Grafana	61
12.2	Seznam nadzornih plošč	61
12.3	Nadzorna plošča	62
12.3.1	Interaktivnost nadzorne plošče	62
12.3.2	Podrobnejši prikaz podatkovnih vrednosti	63
12.3.3	Prehajanje med nadzornimi ploščami	63

1 Video predstavitev tehničnega in uporabniškega vidika pilota ekosistema interneta stvari z algoritmičnimi orodji



Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji

Datum: 4.6. 2024

Predavatelj: Franc Drobnič (UL FE) & Jernej Cvek (UL FRI)



Slika 1: Predstavitev tehničnega vidika pilota ekosistema interneta stvari z algoritmičnimi orodji.

Posnetek predstavitve tehničnega vidika pilota ekosistema interneta stvari z algoritmičnimi orodji se nahaja na povezavi:

https://unilj-my.sharepoint.com/:f/g/personal/gregorbfe_fe1_unilj_si/EiJn3OpDC9VJn62XV2auKNYBS-zjIMIFYdpG4wCGV7_pyA?e=NeGSjf



Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji

Datum: 7.6. 2024

Predavatelj: Franc Drobnič (UL FE) & Jernej Cvek (UL FRI)



Slika 2: Predstavitev uporabniškega vidika pilota ekosistema interneta stvari z algoritmičnimi orodji.

Posnetek predstavitve uporabniškega vidika pilota ekosistema interneta stvari z algoritmičnimi orodji se nahaja na povezavi:

https://unilj-my.sharepoint.com/:f/g/personal/gregorbfe_fe1_unilj_si/EiJn3OpDC9VJn62XV2auKNYBS-zjIMIFYdpG4wCGV7_pyA?e=NeGSjf

2 Uvod

2.1 Namen dokumenta

V dokumentu so podane **Smernice za energetska sanacijo zgradb**. Dokument nastaja v okviru projekta »*Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji*« (v nadaljevanju »platforma«). V strukturirani obliki so zbrani podatki o **podatkih**, ki jih je možno zbirati v okviru pilota.

3 Zahteve

3.1 Zahteve naročnika

V tehnični specifikaciji naročila za vzpostavitev platforme je naročnik podal naslednje zahteve:

3.1.1 Funkcionalnosti platforme

Platforma je zasnovana na osnovi gradnikov fundacije FIWARE in omogoča interoperabilnosti med sorodnimi sestavi. Uporabo odprtokodnih digitalnih gradnikov FIWARE in standardov podpira tudi Evropska komisija.

Izpostavljena sta predvsem dva gradnika:

- Standardizirani informacijski model in aplikacijski programski vmesnik NGSI-LD (ali vsaj NGSI-V2).
- CEF digitalni gradnik Context Broker. Uporaba odprtokodnega digitalnega gradnika je brezplačna, zagotovljeno je tudi dolgoročno vzdrževanje ter podpora pri uvajanju s strani Evropske komisije oz. CEF.

Glavne funkcionalnosti platforme, ki jih zasledujemo:

- podpora za prenos podatkov z in v naprave IoT preko različnih standardnih protokolov,
- možnost naročanja na podatke glede na njihov kontekst,
- shranjevanje podatkov časovnih vrst,
- upravljanje z napravami IoT, uporabniki in njihovimi dostopi,
- vizualizacija podatkov v obliki nadzornih plošč,
- varnost na vseh slojih.

3.1.1.1 Funkcionalne zahteve v okviru pilota za uporabnika DSP - Prvi uporabnik

Pilotni sestav mora omogočati spremljanje porabe energije v stavbah, tako agregirane kot tudi po posameznih porabnikih, zaradi zahtevanega energetskega knjigovodstva in spremljanja učinkovitosti rabe energije. Zagotovljeno mora biti:

- zbiranje podatkov o meritvah porabe električne energije,
- zbiranje podatkov o meritvah porabe toplotne energije,
- zbiranje podatkov o meritvah porabe plina,
- zbiranje podatkov o meritvah porabe vode.

3.1.1.2 Funkcionalne zahteve izven okvira pilotnega projekta

Pilotni projekt mora biti zasnovan tako, da bodo v prihodnosti mogoče njegove nadgradnje in razširitve. V nadaljevanju in izven okvira pilotnega projekta se predvideva vzpostavitev energetskega vodenja stavb z namenom energetske-ekonomske optimizacije procesov, ki so v stavbah povezani s porabo energije. Pilotni projekt mora biti zasnovan na način, da bo za potrebe energetskega vodenja zgradb omogočal dvosmerni podatkovni prenos med osrednjim sestavom in končnimi napravami interneta stvari v zgradbi (senzorji, števcji, ventili, aktuatorji itd.). Osrednji sestav mora omogočati povezavo z ločenim programskim gradnikom, ki bo razvit izven okvira projekta in bo udeležan krmilno-regulacijske funkcionalnosti, ki so potrebne za energetske vodenje zgradb.

3.1.1.3 Predvideni uporabniki pilota

Predvidena uporabnika pilota sta:

- MDP / Direktorat za informatiko in
- MJU / Direktorat za stvarno premoženje.

3.1.2 Osrednji sestav - Splošne tehnične zahteve pilota

3.1.2.1 Skladnost s standardi

Z namenom zagotavljanja čim večje splošnosti, standardizacije, interoperabilnosti in odpornosti na izzive, ki jih prinaša prihodnost, je zahtevana skladnost s specifikacijami NGSI-LD (ali vsaj NGSI-v2), t.j. standardiziranim informacijskim modelom in aplikacijskim programskim vmesnikom za objavljane, poizvedovanje in naročanje na informacije o kontekstu, katerega namen je olajšati odprto izmenjavo in souporabo strukturiranih informacij med različnimi zainteresiranimi stranmi.

V okviru pilotnega projekta mora biti na vseh ravneh omogočeno upravljanje celotnega življenjskega cikla informacij o kontekstu, vključno s posodobitvami, poizvedbami, registracijami in naročninami poenoteno in skladno s specifikacijami NGSI-LD (ali vsaj NGSI-v2).

3.1.2.2 Povezljivost s končnimi napravami interneta stvari - prenosni in podatkovni protokoli in smer podatkovnega toka

Osrednji sestav mora omogočati povezovanje s končnimi napravami interneta stvari po naslednjih prenosnih in podatkovnih protokolih:

- HTTP - Ultralight,
- HTTP - JSON,
- MQTT - Ultralight,
- MQTT - JSON,
- LoRaWAN - Cayenne LPP,
- LoRaWAN - CBOR,
- LoRaWAN - možnost določitve dekoderske funkcije za lastniški podatkovni format.

Osrednji sestav mora omogočati razširitve z dodatnimi prenosnimi in podatkovnimi protokoli ter povezovanje končnih naprav interneta stvari (to so viri podatkov (npr. senzorji), ponori podatkov (npr. aktuatorji) in naprav, ki so tako viri kot tudi ponori podatkov (npr. v zgradbah nameščeni koncentratorji)). Zagotavljati mora dvosmeren podatkovni tok.

3.1.2.3 Varnostne zahteve

Osrednji sestav mora biti navzven zavarovan z uporabo mehanizmov avtentikacije in granularne avtorizacije ter z uporabo kriptografskih postopkov pri prenosu podatkov navzven, kar omogoča zavarovanje na vseh ravneh. Tudi komunikacija med osrednjim sestavom in končnimi napravami interneta stvari mora biti, razen v izjemnih primerih, ko je to tehnično neizvedljivo, zavarovana z uporabo mehanizmov avtentikacije in avtorizacije ter z uporabo kriptografskih postopkov pri prenosu podatkov.

3.1.2.4 Skalabilnost

Sestav mora biti zasnovan na način, ki z naraščajočo obremenitvijo sestava omogoča strojno in programsko skalabilnost.

3.1.2.5 Uporabniki

Sestav mora omogočati več organizacijsko, več uporabniško in več najemniško delovanje.

3.1.2.6 Obdelava osebnih podatkov

Sestav ne obdeluje osebnih podatkov.

3.1.2.7 Pravni vidiki

Potreben je pregled in potrditev ustreznosti licenčnih pogojev vseh posameznih gradnikov sestava za uporabo v okolju državne uprave.

3.1.2.8 Finančni vidiki

Potrebna je analiza finančnih učinkov sestava za uporabo v okolju državne uprave.

4 Arhitektura/ Visokonivojski pogled

V tem poglavju je predstavljen visokonivojski pogled na arhitekturo platforme.

Za opazovanje in upravljanje stvari iz realnega sveta se uporabljajo različne naprave (senzorji in aktuatorji). Naloga prehodov na robu je, da iz množice senzorjev in z njimi povezanih protokolov pridemo do obvladljivega načina prenosa podatkov v platformo. Predvidena je uporaba TCP/IP prehodov in LoRa prehodov z nameščeno ustrezno programsko opremo. V primeru LoRa prehodov se podatki v platformo prenesejo po **protokolu UDP** do **Chirpstack Gateway Bridge-a**, od tam naprej pa po MQTT do Mosquitta, na katerega je povezan tudi Orion-LD Context Broker.

Prenos podatkov med preходом na robu in vstopno točko platforme IoT poteka po zaščiteni povezavi; za **TCP/IP** povezave se uporablja **protokol TLS** => HTTPS, MQTTS.

Ob vstopu v platformo IoT zahteve (sporočila) najprej prestreže API prehod **Kong**, ki poskrbi za njihovo ustrezno preusmerjanje (angl. routing) in omejevanje (angl. rate limiting), po potrebi tudi za beleženje dostopov (angl. access logging). V primeru komunikacije preko protokola HTTP izvede tudi odstranitev zaščitene povezave ("TLS termination") in preverjanje pristnosti.

Kong sporočila glede na aplikacijski omrežni protokol posreduje v zaledni del:

- HTTP, OPC-UA TCP in UDP sporočila posreduje direktno ustreznim agentom IoT,
- MQTT sporočila posreduje posredniku sporočil Mosquitto.

Posrednik sporočil MQTT **Mosquitto** poskrbi za preverjanje pristnosti. Nanj so preko nezaščitene MQTT povezave povezani tudi Orion-LD Context Broker, Chirpstack in agenti IoT.

Agenti IoT so zadolženi za (1) pretvorbo iz različnih formatov sporočil (npr. Ultralight 2.0) v standardni format NGSI in (2) pretvorbo iz različnih aplikacijskih omrežnih protokolov (npr. MQTT) v protokol HTTP.

Orion-LD Context Broker izpostavlja **NGSI-LD API**, preko katerega se nanj povežejo agenti IoT. Kong na tej točki med drugim poskrbi za nadzor dostopa.

Za shranjevanje in vizualizacijo podatkov je trenutno predvidena sledeča možnost. **QuantumLeap** se naroči na podatke Context Broker-ja in jih shranjuje v podatkovno bazo **TimescaleDB**, ki je vir podatkov za vizualizacije v **Grafani**.

Večina gradnikov, ki za svoje delovanje potrebuje shrambo podatkov, uporablja podatkovno bazo PostgreSQL. Podatkovna baza ni nujno del platforme, temveč je lahko zunanja storitev (DBaaS, ipd.). Podobno velja tudi za podatkovno bazo MongoDB, ki jo uporabljajo nekateri FIWARE gradniki.

The diagram illustrates the IoT Gateway Bridge architecture, showing the flow of data from IoT devices through various brokers and gateways to a central data processing stack.

IoT Devices and Gateways:

- HTTP Gateway:** Receives data from a user (represented by a person icon) via HTTP.
- OPC-UA Devices:** Connect to the **OPC-UA TCP** gateway.
- MQTT Gateway:** Connects to the **MQTT Broker (Mosquitto)** via MQTT.
- Lora Gateway:** Connects to the **UDP Packet Forwarder** via UDP.
- Lora Basics Station:** Connects to the **Chirpstack Gateway Bridge** via WSS.

Core Components and Data Flow:

- IoT agents (JSON / Ultralight / LoraWAN / OPC-UA):** Receives data from the **HTTP Gateway** and **OPC-UA TCP** via HTTP. It also receives data from the **Chirpstack Gateway Bridge** via MQTT.
- MQTT Broker (Mosquitto):** Connects to the **IoT agents** via MQTT.
- Chirpstack:** Connects to the **MQTT Broker (Mosquitto)** via MQTT. It also receives data from the **Lora Basics Station** via WSS.
- Chirpstack Gateway Bridge:** Connects to the **Chirpstack** via WSS and to the **IoT agents** via MQTT.
- Orion-LB Context Broker:** Connects to the **IoT agents** via HTTP. It also receives data from the **Chirpstack Gateway Bridge** via MQTT.
- QuantumLeap:** Connects to the **Orion-LB Context Broker** via MQTT.
- TimescaleDB:** Connects to the **QuantumLeap** via MQTT.
- Grafana:** Connects to the **TimescaleDB** via MQTT.

Authentication and Security:

- Keycloak OAuth2 authentication:** Connects to the **Grafana** and **Orion-LB Context Broker**.
- OIDC auth:** Connects to the **Chirpstack** and **Chirpstack Gateway Bridge**.
- custom mosquitto-gateway OAuth2 backend:** Connects to the **Chirpstack Gateway Bridge** and **Orion-LB Context Broker**.

Database and Storage:

- redis:** Connects to the **QuantumLeap** and **Orion-LB Context Broker**.
- mango DB:** Connects to the **Orion-LB Context Broker** and **Chirpstack**.

API Gateway:

- KONG API GATEWAY:** Connects to the **IoT agents** via HTTP.

Legend:

- MQTT:** Green arrow
- HTTP:** Red arrow
- HTTPS:** Red arrow
- OPC-UA TCP:** Grey arrow
- MQTTS:** Green arrow
- UDP:** Blue arrow
- WSS:** Red arrow

12

5 Arhitektura/ Gradniki

5.1 Chirpstack

Chirpstack je odprtokodni omrežni strežnik, ki služi za vzpostavitev omrežij po protokolu LoRaWAN. Z uporabo spletnega vmesnika je možno upravljati prehode, naprave in najemnike in tudi vzpostaviti povezave s ponudniki oblačnih storitev, podatkovnimi bazami in drugimi storitvami za obdelavo podatkov iz naprav.

Chirpstack je sestavljen iz treh osnovnih gradnikov:

- Network Server
- Application Server
- Gateway Bridge

5.2 Dodatne informacije

Chirpstack je odprtokodni omrežni strežnik, ki služi za vzpostavitev omrežij po protokolu LoRaWAN. Z uporabo spletnega vmesnika je možno upravljati prehode, naprave in najemnike in tudi vzpostaviti povezave s ponudniki oblačnih storitev, podatkovnimi bazami in drugimi storitvami za obdelavo podatkov iz naprav.

Chirpstack je sestavljen iz treh osnovnih gradnikov:

- Network Server
- Application Server
- Gateway Bridge

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Chirpstack Network Server	repo	docs	chirpstack/chirpstack-network-server	3
Chirpstack Application Server	repo	docs	chirpstack/chirpstack-application-server	3
Chirpstack Gateway Bridge	repo	docs	chirpstack/chirpstack-gateway-bridge	3
Postgres DB 15 (za Chirpstack)		docs	postgres	15-alpine

OPOMBA

Zaradi združljivosti s starejšimi storitvami predlagamo uporabo različice v3. Na voljo je že različica v4, vendar bi bilo treba njeno delovanje preveriti v ciljnem okolju.

5.3 Context Broker

5.3.1 Predstavitev

Orion je vzorčna implementacija Context Brokerja za upravljanje s podatki konteksta (Context Data Management). Je eden izmed gradnikov CEF Evropske komisije. Podpira API NGSI-LD v skladu s specifikacijami ETSI in tudi deloma NGSIv2.

Predlagamo uporabo različice Orion-LD, ki jo ponuja FIWARE. Vsi ostali gradniki bodo ustrezno nastavljeni za uporabo protokola NGSI-LD.

NGSI-LD Context Broker uporablja kontekst za razširitev in zgoščevanje kratkih imen, ki so del podatkov o koristnem tovoru ali ki so poslani kot parameter URI. Entiteta v NGSI-LD je poljubna, v celoti je odvisna od podatkovnega modela uporabnika.

S pomočjo konteksta se znotraj podatkov o koristnem bremenu razširijo/zgoščijo naslednje kratice:

- Tip entitete,
- Imena lastnosti (na vseh ravneh),
- Imena razmerij (na vseh ravneh),
- Izraz atribut se uporablja za sklicevanje na lastnosti, razmerja, lastnosti lastnosti itd.

Za spodbujanje interoperabilnosti je API NGSI-LD opredeljen z uporabo specifikacije JSON-LD, zato je temeljnega pomena za uporabo NGSI-LD dobro poznavanje JSON-LD in zlasti datotek @context.

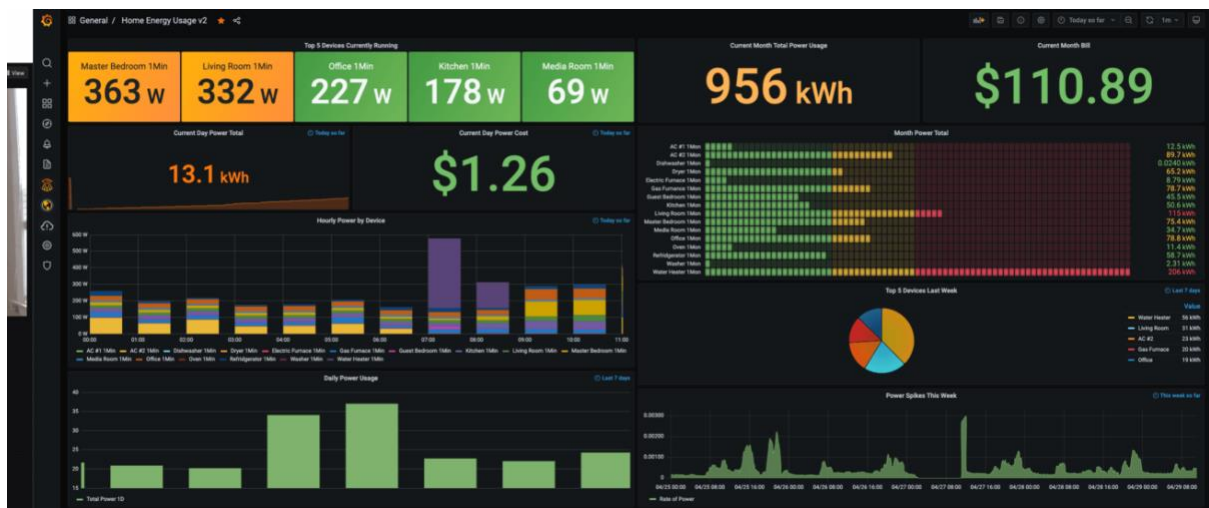
5.3.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Orion-LD Context Broker	repo	docs	fiware/orion-ld	1.4.0

5.4 Grafana

5.4.1 Predstavitev

Grafana je vodilna odprtokodna platforma za vizualizacijo in nadzorne plošče, ki omogoča poizvedovanje, vizualizacijo, opozarjanje in razumevanje podatkov ne glede na to, kje so shranjeni.



Slika 4: Primer nadzorne plošče v orodju Grafana

5.4.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Grafana	repo	docs	grafana/grafana	10.2.1

5.5 IoT Agent Manager

5.5.1 Predstavitev

Gradnik ni obvezen, ponuja zmožnost naslavljanja aktuatorjev po različnih protokolih in posreduje zahteve ustreznemu IoT Agentu glede na protokol.

5.5.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
IoT Agent Manager	repo		telefonicaiot/iotagent-manager	2.1.0-distoreless

5.6 IoT agenti

5.6.1 Predstavitev

IoT Agenti pretvarjajo sporočila senzorskih naprav, ki jih te pošiljajo po različnih protokolih, v obliko NGSI-LD, ki je primerna za vnos v Context Broker. IoT Agenti omogočajo tudi varnostne ukrepe (avtentikacija in avtorizacija kanala).

Več različnih IoT Agentov lahko deluje hkrati, morajo pa imeti nastavljena različna vrata na severni strani (v smeri proti Context Brokerju).

Vsi IoT Agenti podpirajo delovanje po specifikaciji NGSI-LD, kar dosežemo z nastavitvijo okoljske spremenljivke `IOTA_CB_NGSI_VERSION=ld`.

5.6.2 Ultralight

IoT Agent Ultralight podpira protokol Ultralight z uporabo HTTP/REST API-ja, ki zahteva zelo malo pasovne širine za prenos podatkov.

Vzorčna postavitve: <https://github.com/FIWARE/tutorials.IoT-Agent>

5.6.3 LoRa

IoT Agent LoRa podpira protokol LoRaWAN in lahko sprejema podatke v oblikah TheThingsNetwork v3 in Chirpstack.

5.6.4 OPC-UA

IoT Agent LoRa podpira protokol OPC UA (OPC Unified Architecture), ki je prenosljiva razširitev uveljavljenega protokola OPC.

5.6.5 JSON

IoT Agent JSON omogoča pošiljanje podatkov iz senzorsih naprav v osrednji sestav po protokolu HTTP(S), telo sporočila pa je v formatu JSON. V nastavitvah tega agenta definiramo preslikavo iz vhodnega formata JSON v lastnosti izbranega podatkovnega modela NGSI-LD.

5.6.6 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
IoT Agent LoRa	repo	docs	ioeairi/iotagent-lora	1.2.5
IoT Agent Ultralight	repo	docs	quay.io/fiware/iotagent-ul	2.0.0-distroless
IoT Agent OPC-UA	repo	docs	iotagent4fiware/iotagent-opcua	2.2.0
IoT Agent JSON	repo	docs	quay.io/fiware/iotagent-json	2.4.2-distroless

5.7 Keycloak

5.7.1 Predstavitev

Keycloak je odprtokodna rešitev za upravljanje identitet in dostopov (IAM), ki temelji na standardnih protokolih in zagotavlja podporo za OpenID Connect, OAuth 2.0 in SAML.

Izpostavljene funkcionalnosti:

- **Upraviteljska konzola:** Prek upraviteljske konzole lahko skrbniki centralno upravljajo vse vidike strežnika Keycloak.

- **Konzola za upravljanje računov:** V konzoli za upravljanje računov lahko uporabniki upravljajo svoje račune. Posodobijo lahko profil, spremenijo gesla in nastavijo dvofaktorsko preverjanje pristnosti.
- **Storitve avtorizacije:** Če avtorizacija na podlagi vlog (RBAC) ne pokriva potreb, ponuja Keycloak tudi storitve avtorizacije s fino zrnatostjo (npr. ABAC). Single-Sign On (SSO)
- **Posredovanje identitete in prijava z družabnimi omrežji:** Omogočanje prijave z družabnimi omrežji je enostavno dodati prek upraviteljske konzole.
- **Federacija uporabnikov:** Keycloak ima vgrajeno podporo za povezavo z obstoječimi strežniki LDAP ali Active Directory. Če se uporabniki shranjujejo v drugih shrambah, npr. v relacijski podatkovni zbirki, je možna tudi implementacija lastnega ponudnika.

5.7.2 Možnosti integracije

5.7.2.1 Kong

Integracija s Kong-om je podprta s pomočjo po meri izdelanega Kong vtičnika PEP-plugin. Vtičnik so razvili pri FIWARE, podpira delovanje s Kong različicami 2.x. Tekom razvoja platforme je bil vtičnik prilagojen za delo s Kong različicami 3.x.

5.7.2.2 Mosquitto

Za preverjanje pristnosti je na voljo vtičnik Mosquitto Go Auth, ki ponuja več različnih možnosti avtentikacije. Za OAuth 2.0 sicer ni na voljo direktne integracije z rešitvami IAM, se pa lahko implementira vtičnik po meri. Za primer se lahko vzame <https://github.com/gewv-tu-dresden/mosquitto-go-auth-oauth2>.

5.7.2.3 Chirpstack

Chirpstack ponuja integracijo z OpenID ponudniki, med katere se šteje tudi Keycloak.

5.7.2.4 Grafana

Grafano je moč integrirati s Keycloak-om s pomočjo t.i. Keycloak OAuth2 avtentikacije. Primer je opisan tukaj.

5.7.3 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Keycloak	repo	docs	bitnami/keycloak	22.0.5

5.8 Kong

5.8.1 Predstavitev

Kong ali Kong API Gateway je "cloud-native", platformno agnostični, skalabilni prehod API, ki ga odlikujeta visoka zmogljivost in razširljivost s pomočjo vtičnikov.

Med drugim podpira sledeče funkcionalnosti:

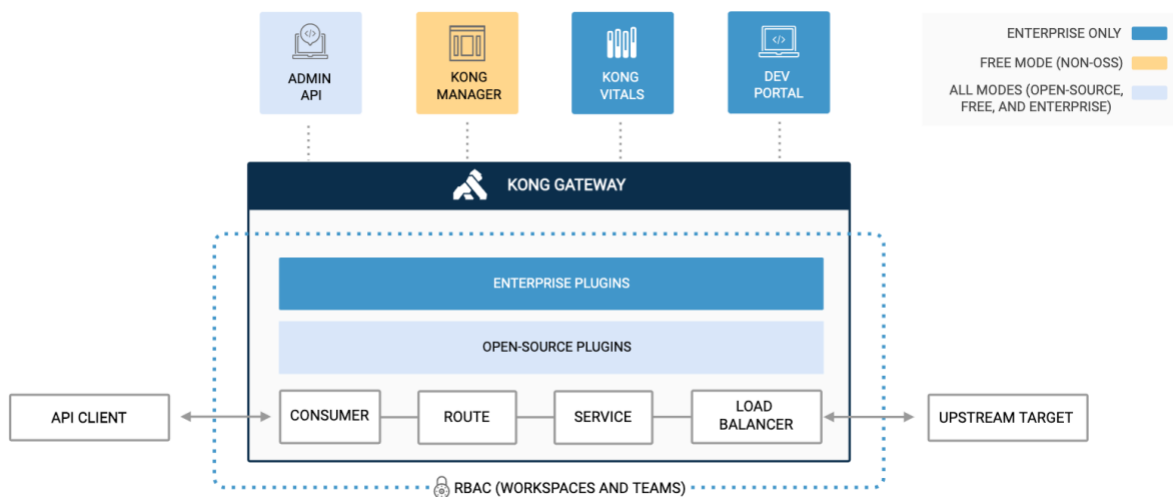
- preverjanje pristnosti in nadzor dostopa,
- napredne možnosti usmerjanja (angl. "routing"), uravnoteženja obremenitve (angl. "load balancing"),
- preoblikovanje oblike vsebine zahtevkov in odgovorov na zahteve (angl. "request/response transformations"),
- terminacija TLS,
- podpora za različne protokole na prenosnem (L4) in aplikacijskem (L7) omrežnem sloju.
- Kong deluje v sistemu Kubernetes zahvaljujoč uradnemu kontrolniku Kubernetes Ingress Controller.

5.8.2 Različice Kong-a

Kong se lahko namesti na dva načina: (1) v oblaku s storitvijo Kong Konnect in (2) lokalno (angl. "on-premises").

Na voljo so naslednje različice Kong-a:

- KONG OSS (open source)
 - Na voljo so zgolj nekatere funkcionalnosti in odprtokodni vtičniki.
 - Upravljanje je možno zgolj preko API-ja.
- Kong FREE
 - Vse iz Kong OSS.
 - Dodatno: upravljanje s pomočjo uporabniškega vmesnika upravljalne konzole Kong Manager
- Kong Premium (\$250 / storitev, na voljo zgolj za postavitve v oblaku (Kong Konnect))
 - Dodatne funkcionalnosti in plačljivi vtičniki.
 - Kong Enterprise
- Še več dodatnih funkcionalnosti:
 - Kong Dev Portal se uporablja za uvajanje novih razvijalcev in ustvarjanje dokumentacije API, ustvarjanje strani po meri, upravljanje različic API-ja in varen dostop razvijalcev.
 - Kong Vitals zagotavlja uporabne metrike o stanju in zmogljivosti vozlišč prehoda Kong ter metrike o uporabi API-jev.
 - RBAC model nadzora dostopa za uporabnike, ki upravljajo Kong,
 - Še več plačljivih vtičnikov.



Slika 5: Arhitektura KONG API Gateway.

5.8.3 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Kong Gateway	repo	docs	kong/kong-gateway	3.4.1.1

5.9 MongoDB

5.9.1 Predstavitev

MongoDB je odprtokodna NoSQL baza, ki shranjuje podatke v obliki JSON.

Uporabljajo jo različni gradniki za shranjevanje dinamičnih nastavitev, sezname registriranih naprav ipd.

5.9.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
MongoDB	repo	docs	mongo	4.4

5.10 Mosquitto

5.10.1 Predstavitev

Mosquitto je implementacija MQTT brokerja, ki jo razvija Eclipse Foundation in podpira verzije MQTT 5, 3.1.1 in 3.1.

Uporabljajo ga različni IoT Agenti kot začasno shrambo za podatke naprav (meritve senzorjev ipd.) in druge metapodatke.

5.10.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Mosquitto	repo	docs	eclipse-mosquitto	1.6.14

5.11 Quantum Leap

5.11.1 Predstavitev

QuantumLeap je storitev REST za shranjevanje, poizvedovanje in iskanje prostorsko-časovnih podatkov NGSI v2 in NGSI-LD (eksperimentalna podpora).

Med drugim podpira sledeče funkcionalnosti:

QuantumLeap pretvori polstrukturirane podatke NGSI v tabelarno obliko in jih shrani v podatkovno zbirko časovnih vrst, pri čemer vsak zapis podatkovne zbirke poveže s časovnim indeksom in, če je podatek prisoten, z lokacijo na Zemlji. Odjemalci lahko nato pridobijo entitete NGSI s filtriranjem nizov entitet prek časovnih razponov in prostorskih operatorjev. Funkcionalnost poizvedb, ki je na voljo prek vmesnika REST, je precej osnovna, za bolj zapletene poizvedbe morajo odjemalci običajno neposredno dostopati do podatkovne zbirke. Za shranjevanje podpira podatkovno zbirko CrateDB in TimescaleDB.

5.11.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
QuantumLeap	repo	docs	orchestracities/quantumleap	edge1

1

orchestracities/quantumleap@sha256:cfec3ebfba1f091ec541c67102581d0f0f315318924fcb6429030930ccf71191

5.12 Redis

5.12.1 Predstavitev

Redis je odprtokodna shramba parov ključ - vrednost.

Uporabljen je v več gradnikih kot začasna shramba v pomnilniku, medpomnilnik ali sporočilna vrsta.

5.12.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
Redis	repo	docs	bitnami/redis	7.2.2

5.13 TimescaleDB

5.13.1 Predstavitev

TimescaleDB je odprtokodna podatkovna zbirka časovnih vrst, zasnovana za hiter vnos, zapletene poizvedbe in preprosto uporabo. Navzven je videti kot PostgreSQL (pravzaprav je pakirana kot razširitev), kar pomeni, da je podedovala zanesljivost, orodja in obsežen ekosistem PostgreSQL-a.

V primerjavi s PostgreSQL ima TimescaleDB za časovne vrste:

20-krat večje število vstavitev (konstantno tudi pri milijardah vrstic),
od 1,2-krat do več kot 14.000-krat hitrejša poizvedba,
2000-krat hitrejša brisanje, kar je ključnega pomena za izvajanje politik hrambe podatkov,
nove časovno usmerjene funkcije, ki še olajšajo manipulacijo časovnih vrst v jeziku SQL.

5.13.2 Dodatne informacije

Gradnik	Github	Dokumentacija	Docker slika	Verzija
TimescaleDB	repo	docs	timescale/timescaledb-ha	pg15-ts2.10

6 Zunanji viri podatkov

Platforma (osrednji sestav) omogoča priključitev posameznih naprav (senzorjev in aktuatorjev) neposredno na vstopne točke, ki so povezane z IoT Agenti znotraj platforme. Možna pa je tudi drugačna povezljivost, to je uporaba zunanjih elementov, ki združujejo več naprav in se v skupno platformo povezujejo hkrati. Platforma trenutno omogoča dva načina povezovanja zunanjih koncentradorjev:

- Zunanji IoT Agent;
- Zunanji Context Broker.

Oba zunanja koncentradorja tečeta na računalniku, ki je nameščen blizu vsem napravam, ki so nanj povezane, in vsebuje vse potrebne strojne elemente (vmesnike) za pravilno komunikacijo s temi napravami. Pri prvem uporabniku naj bi bil nameščen v posamezni stavbi, za katero prvi uporabnik zbira podatke. Z osrednjim sestavom pa potem ta računalnik komunicira po omrežju TCP/IP in praviloma po protokolu HTTPS.

Zunanji IoT Agent nastavimo, da za svoj Context Broker uporablja kar Context Broker osrednjega sestava. Vse ostalo je nastavljeno enako, kot če bi uporabljali IoT Agent znotraj osrednjega sestava.

Zunanji Context Broker pa je treba povezati v osrednji sestav z nastavitvijo t.i. federacije Context Brokerjev. Pripravili smo primer povezave, ko na notranjem Context Brokerju vzpostavimo naročnino na spremembe podatkov v zunanjem Context Brokerju.

Uporaba zunanjih koncentradorjev ima za prednost, da je na omrežju odprta samo ena pot med zunanjim koncentradorjem in osrednjim sestavom, po njej pa teče promet po samo enem protokolu.

7 Namestitev

7.1 Sistemske zahteve

Sistemske zahteve so odvisne od številnih parametrov, ki vplivajo na potrebo po računskih virih. Med ključne parametre spadajo:

- število in vrsta naprav, ki bodo priključene oziroma bodo v komunikaciji s platformo,
- obseg podatkov, ki se bodo izmenjevali s platformo,
- intenzivnost komunikacije (frekvenca pošiljanja/prejemanja podatkov posameznih priključenih IoT naprav),
- arhitektura platforme, to je komponente, ki jo sestavljajo ter odvisnosti med njimi - pomemben vidik, ki bo vplival na sistemske zahteve, je vezan tudi na K8 okolje, ki ga bo zagotovil naročnik,
- zahtevane nefunkcionalne zahteve platforme, kot so na primer razpoložljivost, varnost, zanesljivost, propustnost ipd.
- Minimalne sistemske zahteve bomo podali, ko bodo znani naštetih parametri.

7.1.1 Testno okolje

V času razvoja je testiranje platforme potekalo na virtualnem računalniku s sledečimi specifikacijami:

- 16 vCPU
- 32 GB RAM
- 300 GB disk
- Ubuntu 20.04 LTS
- Docker 24.0.2
- Ker so vsi gradniki kontejnizirani, ne predvidevamo težav tudi v primeru, da bi se uporabil kakšen drug OS.


Za testno okolje predlagamo uporabo računalnika s primerljivimi specifikacijami.

7.2 Seznam uporabljenih gradnikov

V tem poglavju so predstavljeni konkretni gradniki, ki tvorijo platformo.

7.2.1 Seznam Docker slik

Gradnik	Github	Dokumentacija	Licenca	Ustrezno st licence	Docker slika	Verzija
Chirpstack Network Server	repo	docs	licenca	✓	chirpstack/chirpstack-network-server	3
Chirpstack Application Server	repo	docs	licenca	✓	chirpstack/chirpstack-application-server	3
Chirpstack Gateway Bridge	repo	docs	licenca	✓	chirpstack/chirpstack-gateway-bridge	3
Postgres DB 15.x		docs	licenca	✓	postgres	15-alpine
Grafana	repo	docs	licenca	✓	grafana/grafana	10.2.1
IoT Agent Manager	repo		licenca	✓	telefonicaid/iotagent-manager	2.1.0-distrol ss
IoT Agent LoRa	repo	docs	licenca	✓	ioeair/iotagent-lora	1.2.5
IoT Agent Ultralight	repo	docs	licenca	✓	quay.io/fiware/iotagent-ul	2.0.0-distrol ss
IoT Agent OPC-UA	repo	docs	licenca	✓	iotagent4fiware/iotagent-opcua	2.2.0
IoT Agent JSON	repo	docs	licenca	✓	quay.io/fiware/iotagent-json	2.4.2-distrol ss
Kong Gateway	repo	docs	licenca	✓	kong/kong-gateway	3.4.1.1
Keycloak	repo	docs	licenca	✓	bitnami/keycloak	22.0.5
MongoDB	repo	docs	licenca	✓	mongo	4.4
Mosquitto	repo	docs	licenca	✓	eclipse-mosquitto	1.6.14
Orion-LD Context Broker	repo	docs	licenca	✓	fiware/orion-lid	1.4.0
QuantumLeap	repo	docs	licenca	✓	orchestracities/quantumleap	edge ¹
Redis	repo	docs	licenca	✓	bitnami/redis	7.2.2

Gradnik	Github	Dokumentacija	Licenca	Ustrezna licence	Docker slika	Verzija
Timescale DB	repo	docs	licenca		timescale/timescaledb-ha	

1

orchestrations/quantumleap@sha256:cfec3ebfba1f091ec541c67102581d0f0f315318924fcb6429030930ccf71191
opomba

V končni različici postavitve v Docker okolju se bo uporabila zgolj ena instanca (tj. en Docker vsebnik) podatkovne baze PostgreSQL z najvišjo verzijo, ki je še združljiva z vsemi gradniki. V tej instanci bodo vsebovane podatkovne zbirke vseh gradnikov, ki za svoje delovanje potrebujejo podatkovno bazo.

7.2.2 Seznam odvisnosti

Spodnja tabela prikazuje medsebojne odvisnosti gradnikov.

Gradnik	Odvisnosti
Chirpstack Network Server	Redis, Mosquitto
Chirpstack Application Server	Postgres, Redis, Mosquitto, Chirpstack Network Server
Chirpstack Gateway Bridge	Mosquitto
IoT agent LoRa	MongoDB, Orion-LD Context Broker
IoT agent Ultralight	MongoDB, Mosquitto, Orion-LD Context Broker
IoT agent OPC-UA	MongoDB, Orion-LD Context Broker
IoT agent JSON	MongoDB, Mosquitto, Orion-LD Context Broker
Kong Gateway	Postgres
Keycloak	Postgres
Orion-LD Context Broker	MongoDB
QuantumLeap	TimescaleDB, Redis
Grafana	TimescaleDB

7.2.3 Seznam ranljivosti

V tem podglavju je navedeno število programskih ranljivosti za posamične gradnike.

Ranljivosti se lahko pojavijo zaradi programskih napak, nepravilne konfiguracije, slabih varnostnih praks med razvojem ali drugih vzrokov. Programske ranljivosti so napake, pomanjkljivosti ali slabosti v programski opremi, ki omogočajo napadalcem, da jih izkoristijo in pridobijo nepooblaščen dostop, povzročijo izpad sistema, ukradejo podatke ali izvedejo druge škodljive dejavnosti.

Za odkrivanje ranljivosti se uporablja odprtokodni varnostni skener [Trivy](#). Priporočljivo je, da se za aktualno stanje opravi ponovno skeniranje naštetih Docker slik.

Docker slika gradnika	Ranljivosti
chirpstack/chirpstack-network-server:3	Total: 32 (UNKNOWN: 0, LOW: 0, MEDIUM: 14, HIGH: 18, CRITICAL: 0)
chirpstack/chirpstack-application-server:3	Total: 32 (UNKNOWN: 0, LOW: 0, MEDIUM: 14, HIGH: 18, CRITICAL: 0)
chirpstack/chirpstack-gateway-bridge:3	Total: 4 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 4, CRITICAL: 0)
postgres:15-alpine	Total: 26 (UNKNOWN: 0, LOW: 0, MEDIUM: 12, HIGH: 14, CRITICAL: 0)
grafana/grafana:10.2.1	Total: 14 (UNKNOWN: 0, LOW: 0, MEDIUM: 12, HIGH: 2, CRITICAL: 0)
telefonicaid/iotagent-manager:2.1.0	Total: 33 (UNKNOWN: 0, LOW: 11, MEDIUM: 19, HIGH: 3, CRITICAL: 0)
ioeairi/iotagent-lora:1.2.5	Total: 77 (UNKNOWN: 1, LOW: 34, MEDIUM: 11, HIGH: 28, CRITICAL: 3) Total: 46 (UNKNOWN: 0, LOW: 2, MEDIUM: 18, HIGH: 21, CRITICAL: 5) CVE-2022-1664 CVE-2019-12900 CVE-2019-8457 CVE-2021-3807 CVE-2021-44906 CVE-2021-44906 CVE-2023-3696 CVE-2021-28918
quay.io/fiware/iotagent-ul:2.0.0-distroless	Total: 23 (UNKNOWN: 0, LOW: 11, MEDIUM: 8, HIGH: 4, CRITICAL: 0) Total: 9 (UNKNOWN: 0, LOW: 0, MEDIUM: 5, HIGH: 2, CRITICAL: 2) CVE-2023-3696 CVE-2022-0686
iotagent4fiware/iotagent-opcua:2.1.9	Total: 125 (UNKNOWN: 1, LOW: 89, MEDIUM: 10, HIGH: 24, CRITICAL: 1) Total: 4 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 1, CRITICAL: 1) CVE-2019-8457 CVE-2023-3696
quay.io/fiware/iotagent-json:2.4.2-distroless	Total: 33 (UNKNOWN: 0, LOW: 11, MEDIUM: 19, HIGH: 3, CRITICAL: 0)
kong/kong-gateway:3.4.1.1	Total: 71 (UNKNOWN: 0, LOW: 50, MEDIUM: 21, HIGH: 0, CRITICAL: 0)
bitnami/keycloak:22.0.5	Total: 165 (UNKNOWN: 0, LOW: 96, MEDIUM: 33, HIGH: 31, CRITICAL: 5) CVE-2023-23914 CVE-2023-23914 CVE-2019-8457 CVE-2024-24806 CVE-2023-45853
mongo:4.4	Total: 25 (UNKNOWN: 0, LOW: 25, MEDIUM: 0, HIGH: 0, CRITICAL: 0) Total: 4 (UNKNOWN: 0, LOW: 1, MEDIUM: 2, HIGH: 1, CRITICAL: 0)

Docker slika gradnika	Ranljivosti
eclipse-mosquitto:1.6.14	Total: 31 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 25, CRITICAL: 4) CVE-2021-36159 CVE-2021-3711 CVE-2021-3711 CVE-2022-37434
fiware/orion-ld:1.2.1	Total: 381 (UNKNOWN: 0, LOW: 144, MEDIUM: 219, HIGH: 18, CRITICAL: 0)
orchestracities/quantumleap:edge ¹	Total: 74 (UNKNOWN: 0, LOW: 2, MEDIUM: 17, HIGH: 44, CRITICAL: 11) CVE-2021-36159 CVE-2022-22822 CVE-2022-22823 CVE-2022-22824 CVE-2022-23852 CVE-2022-25235 CVE-2022-25236 CVE-2022-25315 CVE-2021-3711 CVE-2021-3711 CVE-2022-37434 CVE-2022-29361
bitnami/redis:7.2.2	Total: 116 (UNKNOWN: 0, LOW: 81, MEDIUM: 25, HIGH: 8, CRITICAL: 2) CVE-2019-8457 CVE-2023-45853
timescale/timescaledb-ha:pg15-ts2.10	Total: 163 (UNKNOWN: 0, LOW: 52, MEDIUM: 111, HIGH: 0, CRITICAL: 0)

1

orchestracities/quantumleap@sha256:cfec3ebfba1f091ec541c67102581d0f0f315318924fcb6429030930ccf71191
opomba

Tekom razvoja se bodo gradniki posodabljali na najnovejše združljive verzije.

8 Konfiguracija omrežja

To poglavje vsebuje informacije o konfiguraciji omrežja, ki je potrebna za pravilno delovanje platforme.

8.1 Seznam omrežnih vrat

Gradnik	Vrata vsebnika	Vrata gostitelja
Chripstack Network Server	8000/tcp	
Chripstack Application Server	8080/tcp, 8081/tcp, 8003/tcp	
Chripstack Gateway Bridge	1700/udp	
Postgres DB 15.x	5432/tcp	
Grafana	3000/tcp	
IoT agent LoRa	4041/tcp, 4061/tcp	

Gradnik	Vrata vsebnika	Vrata gostitelja
IoT agent Ultralight	4041/tcp, 4061/tcp, 7896/tcp	
IoT agent OPC-UA	4041/tcp, 9229/tcp	
IoT agent JSON	4041/tcp, 7896/tcp	
IoT agent manager	8082/tcp	
Kong Gateway	8000-8004/tcp, 8443-8447/tcp	80 (HTTP) ¹ , 443 (HTTPS), 1700 (UDP - Chirpstack GB), 1883 (MQTT) ²
Keycloak	8080/tcp, 8443/tcp	
MongoDB	27017/tcp	
Mosquitto	1883/tcp	
Orion-LD Context Broker	1026/tcp	
QuantumLeap	8668/tcp	
Redis	6379/tcp	
TimescaleDB	5432/tcp, 8008/tcp, 8081/tcp	

¹ Vrata 80 za HTTP se uporabljajo za [Let's Encrypt HTTP-01 challenge](#). Ob uporabi [DNS-01 challenge](#) se lahko ta vrata zaprejo.

² Vrata za MQTT se bodo kasneje spremenila iz 1883 (nekriptirana) v 8883 (kriptirana - MQTTS).

8.2 Postopek namestitve

To poglavje opisuje postopek začetne namestitve platforme na gostiteljski računalnik z uporabo Docker vsebnikov.

8.2.1 Predpogoji

Pred namestitvijo platforme je potrebno zagotoviti:

- gostiteljski računalnik z operacijskim sistem Ubuntu 20.04
- ustrezno nastavljen požarni zid
- javni IP naslov in domena z možnostjo urejanja DNS zapisov
- Docker 24.0.5
- [docker-compose v2.20.3](#)
- Python >= 3.6
- Python [pip](#)
- Python [virtualenv](#)

8.2.2 Izvorna koda za namestitev platforme

Izvorna koda za namestitev platforme se nahaja v Git skladišču.

git clone [git@github.com:mju-iot/mju-iot-platform.git](https://github.com:mju-iot/mju-iot-platform.git)

8.2.3 Wildcard DNS zapis

DNS poskrbi za mapiranje domene v IP naslov gostiteljskega računalnika.

Priporočljivo je uporabiti t.i. "wildcard" DNS zapis.

Tip DNS zapisa	Ime	IPv4 naslov
A	*	192.0.2.1

Zgornji primer bo preslikal domeno in vse njene poddomene (*) na podan IP naslov.

informacija

DNS zapis je potrebno dodati le v primeru produkcijskega načina uporabe.

8.2.4 Inicializacija platforme

Pred prvim zagonom platforme je potrebno izvesti inicializacijo.

Ob zagonu skripte `init.sh` se bo izvedel avtomatski postopek inicializacije. V korenskem direktoriju projekta je potrebno zagnati sledeča ukaza:

```
# Nastavitev pravic za zagon skripte
chmod u+x init.sh
# Zagon init skripte
./init.sh
```

Skripta uporabnika vodi čez postopek inicializacije platforme in med drugim poskrbi za:

- izbiro načina namestitve (Development, Production)
- [Production] nastavitve domene in admin e-poštnega naslova
- shranjevanje nastavitev v `settings.yml`
- ustvarjanje datoteke z okoljskimi spremenljivkami `.env`
- ustrezno konfiguracijo gradnikov
- [Development] dodajanje lokalne domene `mju-iot.local` in ostalih poddomen v `/etc/hosts`
- možnost zagona platforme po uspešno zaključeni inicializaciji

Za razvoj in/ali lokalni preizkus platforme priporočamo uporabo Development načina.

informacija

POMEMBNO: Platforma se zaenkrat zažene s privzetimi gesli in drugimi občutljivimi podatki.

Pred produkcijsko uporabo je potrebno z ročno konfiguracijo poskrbeti za ustrezno spremembo gesel.

8.2.5 Zagon platforme

Ob zagonu je s pomočjo healthcheck-ov (v docker-compose.yml) poskrbljeno, da se gradniki vzpostavijo zaporedno v pravilnem vrstnem redu, tj. najprej podatkovne baze, nato Kong API gateway, itd.

Tekom prvega zagona se avtomatsko opravi tudi inicializacija podatkovnih baz, Kong-a in drugih gradnikov.

Platforma je pripravljena za uporabo, ko imajo vsi gradniki status healthy, kar se lahko preveri z ukazom:

```
docker ps
```

8.2.6 Upravljanje platforme

Upravljanje platforme je možno s sledečimi ukazi, ki jih je potrebno zagnati v korenskem direktoriju projekta:

```
# Zagon platforme
docker compose up -d
# Spremljanje vseh dnevniških zapisov
docker compose logs -f --tail=10
# Spremljanje dnevniških zapisov določenega gradnika
docker compose logs -f --tail=1000 kong-gateway
# Začasna ustavitev platforme
docker compose stop
# Ponovni zagon platforme
docker compose restart
# Izbris platforme
docker compose down -v --remove-orphans
```

9 Docker Swarm mode

Zaradi tehničnih omejitev namestitvenega okolja pri naročniku se bo namesto predvidene implementacije na platformi Kubernetes za potencialno večvozliščno namestitev uporabilo Docker Swarm.

Docker Swarm je način uporabe Docker izvajalnega okolja, ki, podobno kot orkestrator Kubernetes, več strežnikov poveže v enotno platformo, na katero se lahko transparentno namešča večje število kontejnerjev. Namestitev in upravljanje Docker Swarm je preprostejše, kot to velja za Kubernetes, vendar pa je tudi sama zmogljivost in razširjenost Docker Swarm slabša. Še posebej opozarjamo na to, da se trenutno v Docker Swarm način uporabe za specifikacijo storitev uporablja specifikacijo Docker Compose V1, ne pa še naslednjo verzijo specifikacije Docker Compose V2. Po našem mnenju tudi to nakazuje, da samo podjetje Docker svojega lastnega Docker Swarm-a ne gleda več kot primarno platformo za orkestracijo, saj recimo v svojem glavnem produktu, IDE okolju za razvijalce Docker Desktop, zelo dobro podpira uporabo "konkurenčne" platforme Kubernetes.

Za pilotno namestitev priporočamo namestitev vseh komponent na (en) samostojen strežnik oz. virtualni računalnik, namenjen samo temu projektu. S tem se omeji območje vpliva pri napakah/hroščih v komponentah (t.i. "blast radius") samo na to aplikacijo, ne pa na celotno platformo. Pilotna namestitev tudi ne predvideva velikega števila uporabnikov, senzorjev in prometa, tako da je tehnično bolj zahtevna namestitev na večvozliščen način v tej fazi projekta manj smiselna.

Za namestitev aplikacijskega sklada se na Docker Swarm uporabi namesto docker compose kar ukaz:

```
docker stack deploy --compose-file docker-compose.yaml
```

Na samem strežniku je potrebno prej eksplicitno vklopiti Swarm način uporabe ("Swarm enabled host"), najprej za kontrolne strežnike ("manager node"), nato še za delavne strežnike ("worker nodes"). Če se na takšni Docker Swarm platformi zažene aplikacijski sklad z docker-compose ali docker compose (compose V2), se aplikacijski sklad namesti klasično, samo na trenutni strežnik, ne na celotno Swarm platformo.

10 Varnost

10.1 Varnost prenosnega sloja

Varnost prenosnega sloja (angl. *Transport Layer Security*, krajše **TLS**) je široko sprejet varnostni kriptografski protokol, zasnovan za zagotavljanje zasebnosti in varnosti podatkov med komunikacijo po medmrežju.

Protokol TLS v glavnem skrbi za:

- **šifriranje** (angl. *Encryption*), ki skriva prenesene podatke pred tretjimi osebami.
- **preverjanje pristnosti** (angl. *Authentication*), ki zagotavlja, da sta stranki, ki si izmenjujeta informacije, tisti, za kateri se izkazujeta,
- **integriteto** (angl. *Integrity*), ki preverja, da podatki niso bili ponarejeni ali prirejeni.

Več o protokolu TLS [tukaj](#)

10.1.1 Kriptografski sistemi

Kriptografski sistemi (angl. *cryptographic systems*) s pomočjo šifrnih algoritmov v kombinaciji s kriptografskimi ključi pretvarjajo golo vsebino sporočila (angl. *plaintext message*) v šifrirano obliko.

Šifrirni algoritem (angl. *cipher*) je serija dobro definiranih računskih korakov, ki se uporablja za šifriranje sporočila. Onemogočil (ali pa vsaj zelo otežil) naj bi obratno pridobivanje originalne vsebine iz šifrirane vsebine sporočila. Zaradi splošne dostopnosti informacij o delovanju šifrnih algoritmov, današnji kriptografski sistemi poleg njih uporabljajo tudi kriptografske ključe.

Kriptografski ključ je niz bitov, ki določa izhod šifrnega algoritma. Za ohranjanje varnosti torej ni več potrebna tajnost šifrnega algoritma, saj se lahko šifrirano sporočilo dešifrira le, če dešifrirni kriptografski ključ ustreza šifrnemu kriptografskemu ključu.

10.1.2 Kriptografija javnega ključa

Kriptografija javnega ključa (angl. *public-key cryptography*), poznana tudi kot asimetrična kriptografija (angl. *asymmetric cryptography*), je kriptografski sistem, ki za enkripcijo uporablja par kriptografskih ključev: **javni ključ**, ki je lahko javno znan, in **zasebni ključ**, ki je znan samo lastniku. Pošiljatelj s pomočjo javnega ključa, ki služi kot parameter šifrnemu algoritmu, zašifrira vsebino sporočila. Javni ključ se običajno uporablja kot parameter v šifrnem algoritmu, medtem ko se zasebni ključ uporablja za dešifriranje.

10.1.3 Digitalno potrdilo javnega ključa

Digitalno potrdilo javnega ključa (angl. *public key certificate*) je digitalni dokument, ki potrjuje povezavo med javnim ključem in neko osebo, institucijo ali strežnikom.

10.1.4 HTTPS

HTTPS (kratica za *Hypertext Transfer Protocol Secure*) je razširitev protokola HTTP. V protokolu HTTPS je komunikacijski protokol šifriran s protokolom TLS, zato je njegovo ime tudi "HTTP over TLS". Varnost protokola HTTPS torej zagotavlja protokol TLS, ki običajno uporablja dolgoročne javne in zasebne ključe za ustvarjanje kratkoročnega ključa seje, ki se nato uporablja za šifriranje podatkovnega toka med odjemalcem in strežnikom. Za preverjanje pristnosti strežnika (in včasih tudi odjemalca) se uporabljajo digitalna potrdila. Posledično so **organi za izdajo digitalnih potrdil** (angl.

certificate authorities, krajše *CA*) in digitalnega potrdila javnega ključa potrebni za preverjanje razmerja med digitalnim potrdilom in njegovim lastnikom ter za generiranje, podpisovanje in upravljanje veljavnosti digitalnih potrdil.

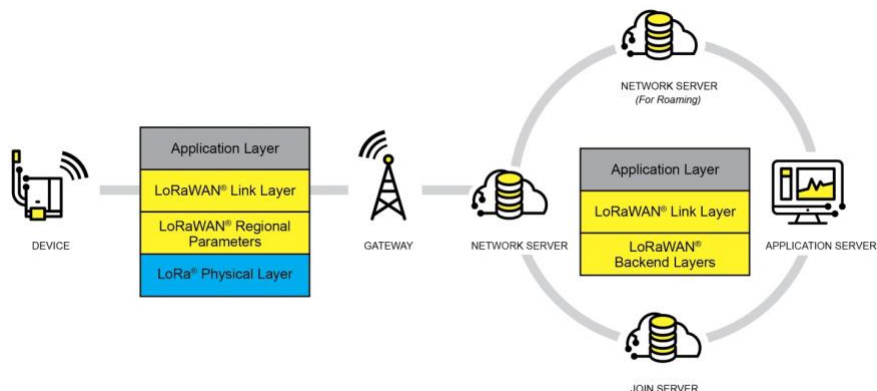
10.1.5 MQTTS

Podobno velja tudi za protokol **MQTTS**. Manjša slabost uporabe MQTTS je v tem, da varnost s seboj prinese dodatne obremenitve v smislu porabe CPE in dodatno potrebne komunikacije. Čeprav je dodatna poraba CPE zanemarljiva na infrastrukturi, kjer so običajno nameščeni **posredniki sporočil** (angl. *message brokers*), lahko postane problematična za **naprave z omejenimi viri** (angl. *constrained devices*), ki niso zasnovane za računsko intenzivna opravila. V primeru kratkoživih povezav, je vpliv TLS Handshake-a relativno velik, zato se priporoča uporaba dolgoživih povezav, v kolikor je to možno.

10.1.6 LoRaWAN

Specifikacija **LoRaWAN®** je omrežni protokol *LPWA* (*Low Power, Wide Area*), ki je zasnovan za brezžično povezovanje baterijskih "stvari" z internetom v regionalna, nacionalna ali globalna omrežja in je namenjen ključnim zahtevam interneta stvari, kot so dvosmerna komunikacija, celovita varnost, mobilnost in storitve lokalizacije.

Omrežna arhitektura LoRaWAN® se uporablja v topologiji "star-of-stars", v kateri prehodi posredujejo sporočila med končnimi napravami IoT in osrednjim omrežnim strežnikom. Prehodi so z omrežnim strežnikom povezani prek standardnih povezav IP in delujejo kot pregleden most, ki preprosto pretvarja RF pakete v IP pakete in obratno. Brezžična komunikacija izkorišča lastnosti fizičnega sloja LoRa® za dolge razdalje, kar omogoča povezavo z enim samim skokom med končno napravo in enim ali več prehodi.



Slika 6: LoRaWAN arhitektura.

LoRaWAN ima **tri različne razrede** končnih naprav, ki ustrezajo različnim potrebam, izraženim v široki paleti aplikacij.

10.1.7 OPC-UA

OPC pomeni *Open Platform Communications* in je eden najpomembnejših komunikacijskih standardov za industrijo 4.0 in internet stvari. Z OPC je dostop do strojev, naprav in drugih sistemov v industrijskem okolju standardiziran ter omogoča podobno in od proizvajalca neodvisno izmenjavo podatkov.

V tem kontekstu je pomen UA v **OPC UA** "Unified Architecture" ("enotna arhitektura") in se nanaša na najnovejšo specifikacijo standarda. Od predhodne specifikacije se razlikuje po tem, da je neodvisna od platforme, saj se odmika od COM/DCOM in prehaja na povsem binarni TCP/IP ali alternativno SOAP. Poleg številnih drugih izboljšav OPC UA podpira tudi semantični opis podatkov.

Več o OPC UA [tukaj](#).

10.2 Zaščita podprtih protokolov/ HTTPS

Sledeči gradniki platforme izpostavljajo možnost komunikacije preko protokola HTTP:

- vsi agenti IoT (JSON, Ultralight, LoraWan, OPC-UA),
- Orion-LD Context Broker,
- QuantumLeap,
- Chripstack,
- Kong, Keycloak, Grafana.

Med odjemalci (npr. naprave IoT) in vsemi naštetimi gradniki platforme je omogočena komunikacija po varni šifrirani povezavi - HTTPS.

Na vstopni točki (Kong) se podatki dešifrirajo; opravi se t.i. TLS termination. V notranjem (Docker) omrežju platforme prenos podatkov poteka po nešifrirani povezavi (HTTP).

10.2.1 Avtomatska preusmeritev HTTP -> HTTPS

V primeru produkcijske postavitve je priporočljivo, da se vključi avtomatska preusmeritev zahtevkov HTTP na uporabo varne povezave HTTPS.

Za avtomatsko preusmeritev HTTPS -> HTTPS je potrebno za izpostavljen gradnik platforme v Routes nastaviti, da je dovoljen protokol komunikacije zgolj https (odstrani se http). V tem primeru Kong avtomatsko zavrne zahteve, ki so izvedeni po nezaščiteni povezavi (HTTP).

Privzeto se ob zavrnitvi uporabi HTTP koda 426 Upgrade Required . Za avtomatsko preusmeritev je potrebno v Kong-u nastaviti, da se uporabi HTTP koda 301 Moved Permanently .

Primer konfiguracije:

```
routes:
  - hosts:
      - docs.mju-iot.local
    https_redirect_status_code: 301 # privzeto 426
    protocols:
      # - http <-- potrebno izbrisati
      - https
    # ostale nastavitve ...
```

Slika 7: Preusmeritev HTTP -> HTTPS

Konfiguracija se lahko prilagodi tudi s pomočjo Kong Manager-ja (Routes)

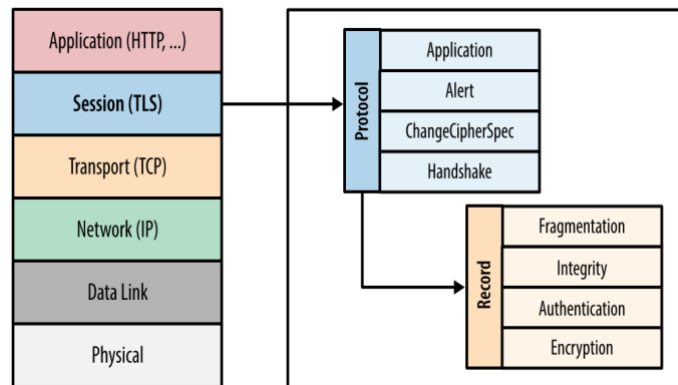
10.3 Zaščita podprtih protokolov / MQTTS

10.3.1 MQTTS v Kong-u

Kong na aplikacijski plasti (L7 v ISO/OSI modelu) podpira zgolj protokol HTTP.

Usmerjanje prometa aplikacijskega protokola MQTT je možno na nižjem nivoju transportne plasti (L4), torej protokola TCP.

Zaščita povezave je možna na način, da se nad TCP uporabi varnostni kriptografski protokol TLS.



Slika 8: Uporaba TLS protokola

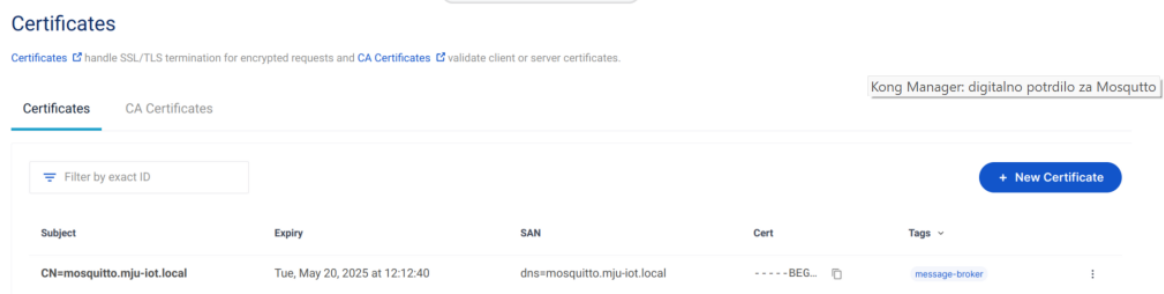
Pri tem se uporablja t.i. Server Name Indication (SNI):

An encrypted TLS tunnel can be established between any two TCP peers: the client only needs to know the IP address of the other peer to make the connection and perform the TLS handshake. However, what if the server wants to host multiple independent sites, each with its own TLS certificate, on the same IP address — how does that work? Trick question; it doesn't.

To address the preceding problem, the Server Name Indication (SNI) extension was introduced to the TLS protocol, which allows the client to indicate the hostname the client is attempting to connect to as part of the TLS handshake. In turn, the server is able to inspect the SNI hostname sent in the ClientHello message, select the appropriate certificate, and complete the TLS handshake for the desired host.

10.3.2 Primer konfiguracije

Spodaj je prikazan primer uporabe MQTTS za posrednika sporočil Mosquitto. V Kong Manager-ju se v razdelku Certificates doda digitalno potrdilo.



Slika 9: Konfiguracija MQTTS Mosquitto 1

V razdelku SNIs se digitalno potrdilo se poveže s SNI.

SNIs

An SNI object represents a many-to-one mapping of hostnames to a certificate. [Learn more](#)

Filter			+ New SNI
Name	SSL Certificate ID	Tags	
mosquitto.mju-iot.local	59c1f653-bca2-48d8-9ff9-ed496d13dd64	message-broker	

Kong Manager: SNI za digitalno potrdilo Mosquitto

Slika 10: Konfiguracija MQTTS Mosquitto 2

V razdelku Routes se za gradnik uporabi protokola tcp in tls ter ustrezno nastavi SNI.

SNI.

Routes > Mosquitto-MQTT-encrypted >

Mosquitto-MQTT-encrypted

Tags message-broker

Configuration

Analytics

Plugins

Configuration

ID	79cde43b-7f37-45cd-973b-cfe713962458
Name	Mosquitto-MQTT-encrypted
Gateway Service	Mosquitto-MQTT
Protocols	tcp tls
Tags	message-broker
Source(s)	
Destination(s)	{ "port": 8883, "ip": null }

Advanced

SNI(s)

mosquitto.mju-iot.local

Slika 11: Konfiguracija MQTTS Mosquitto 3

10.4 Zaščita podprtih protokolov / LoRaWAN

10.4.1 Varnost v omrežju LoRaWAN

Omrežja LoRaWAN podpirajo sodobne tehnologije zaščite podatkov:

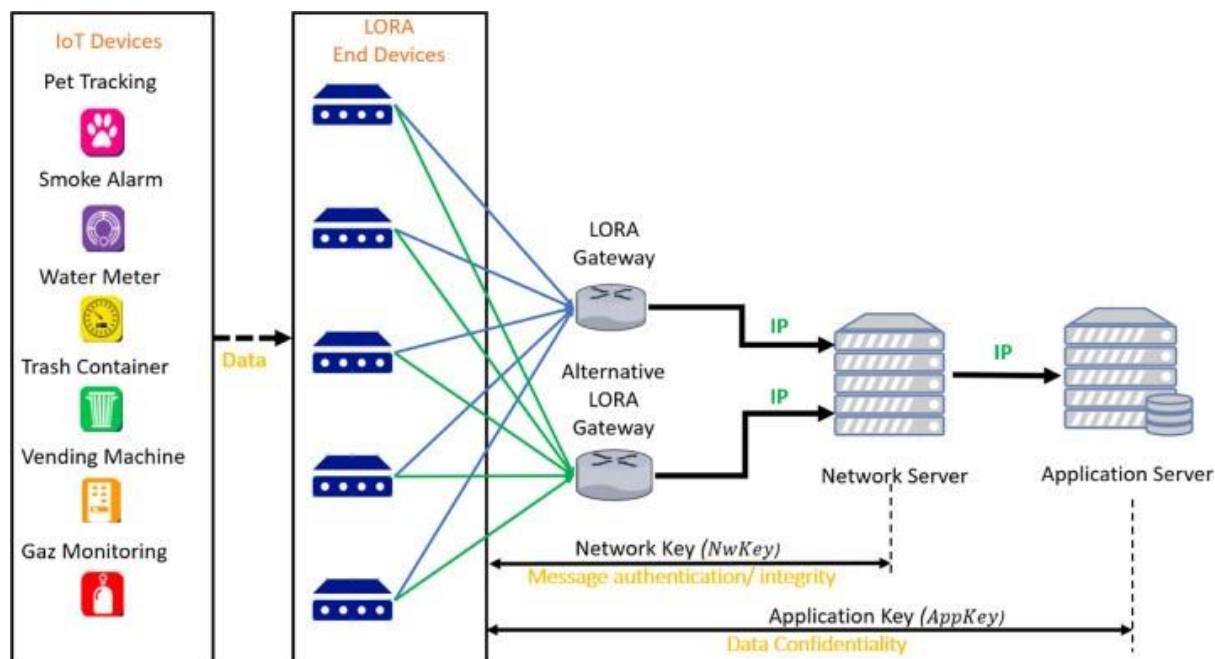
AES-128,

HMAC,

EUI-64 za identifikacijo (naprav ipd.),

Join Request/Join Accept protokol za zavarovanje pridruževanja naprav,

Frame Counter protokol za preprečevanje podvojitve prenosa podatkov. Za zaščito prenosa podatkov je v omrežjih LoRaWAN možnih več nivojev:



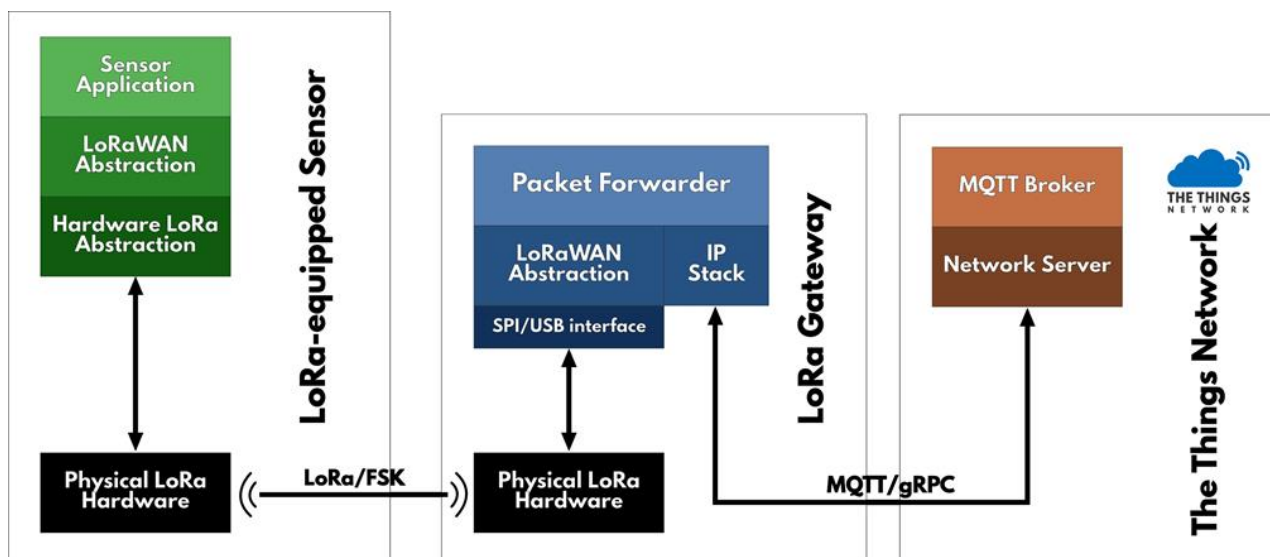
Slika 12: Shema zaščita protokola LoRaWAN

Avtentikacija in zaščita pred ponarejanjem je izvedena z uporabo omrežnega ključa (angl. Network Key), ki ščiti podatke med prenosom od LoRa končnih naprav (npr. senzorjev) do omrežnega strežnika, in to po brezžičnem prenosu LoRa in kasneje po internetnem protokolu (IP). Zaupnost vsebine pa omogoča uporaba aplikacijskega ključa (angl. Application Key), ki ščiti podatke še naprej, do aplikacijskega strežnika, in s tem onemogoča omrežnemu strežniku vpogled v podatke. Oba ključa sta po standardu AES-128. Varnost prenosa po omrežjih IP pa je lahko zagotovljena s standardnimi rešitvami za IP, npr. TLS, IPsec, požarne pregrade.

10.4.2 Chirpstack Gateway Bridge

Paketni posrednik (angl. Packet Forwarder) je program, ki teče na prehodu LoRaWAN (angl. LoRaWAN gateway) in komunicira:

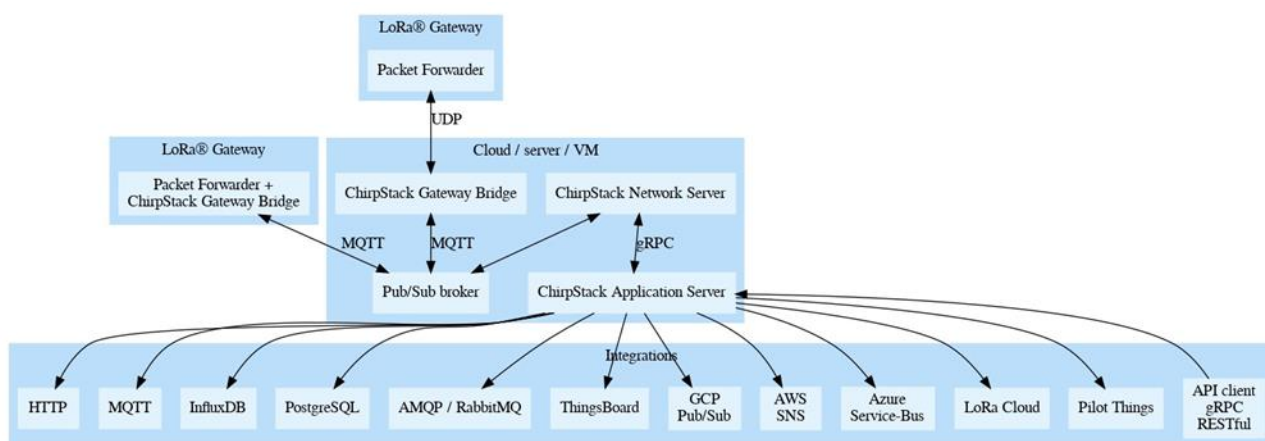
- s čipom LoRa za sprejemanje in oddajanje paketov LoRa,
- z omrežjem - za prenos teh paketov do aplikacij.



Slika 13: implementaciji paketnih posrednikov sta npr. *Semtech UDP Packet Forwarder* in *Semtech Basic Station Packet Forwarder*

Običajna implementaciji paketnih posrednikov sta npr. [Semtech UDP Packet Forwarder](#) in [Semtech Basic Station Packet Forwarder](#).

ChirpStack Gateway Bridge je storitev, ki skrbi za komunikacijo s prehodi LoRaWAN in ustrezno pretvorbo podatkov iz LoRa Packet Forwarder protokolov v formate (JSON in Protobuf), ki jih razume omrežni strežnik - **Chirpstack Network Server**.



Slika 14: *ChirpStack Gateway Bridge*

10.4.3 UDP

V trenutni različici platforme je komunikacija med prehodom LoRaWAN in Chirpstackom podprta tudi po protoklu UDP. Pakete prestreže Chirpstack Gateway Bridge, ki je preko Konga navzven izpostavljen na vratih 1700. Le-ta jih pretvori v ustrezno obliko (JSON) in jih v obliki sporočil preko protokola MQTT (brez zaščite, ker je v internem omrežju platforme) pošlje do posrednika sporočil (Mosquitto). Na ta sporočila se naroči LoRaWAN IoT agent, ki podatke posreduje v Context Broker.

Po potrebi bi se za dodatno zaščito prenosa podatkov preko protokola UDP lahko v prihodnjih različicah platforme raziskalo možnosti uporabe protokola DTLS.

10.5 Zaščita podprtih protokolov / OPC UA

10.5.1 Varnost v OPC UA

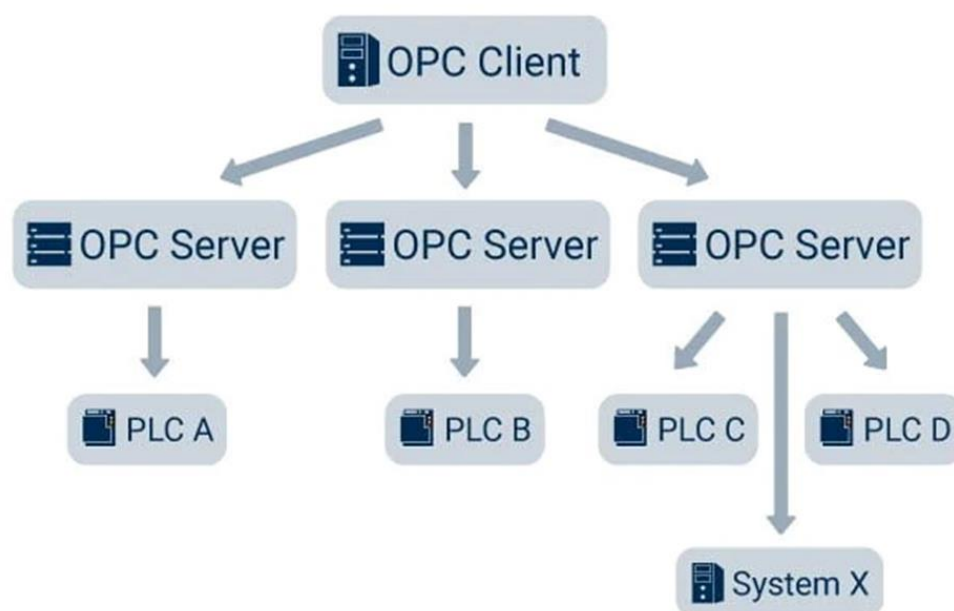
Pri razvoju standarda OPC UA je bila že od samega začetka upoštevana najvišja stopnja varnosti. V nasprotju s standardom OPC Classic je bil OPC UA razvit kot prijazen požarnim zidovom, kar pomeni, da ga je mogoče nadzorovati in krmiliti s standardnimi omrežnimi tehnikami. Na transportni plasti je podprtih več protokolov; med drugim je možna tudi uporaba TCP/IP.

Za varovanje podatkov med prenosom se uporablja šifriranje (128 ali 256 bitov), podpisovanje sporočil, zaporedje paketov in avtentikacija uporabnikov.

OPC UA za dodatno varnost uporablja izmenjavo potrdil, tako da se mora vsak odjemalec avtentificirati s potrdilom. Na ta način je mogoče nadzorovati, kateri odjemalec se lahko poveže s strežnikom.

10.5.2 Povezovanje OPC UA agenta IoT

Strežnik OPC UA je običajno odgovoren za pridobivanje podatkov o senzorjih iz tovarniških strojev. Te podatke nato izpostavlja OPC UA odjemalcu.



Slika 15: Povezovanje OPC UA agenta IoT

V platformi OPC UA agent IoT deluje kot OPC UA odjemalec. OPC UA IoT agent se torej povezuje iz platforme na strežnik OPC UA preko nižjenivojskega protokola

TCP. Agent OPC UA bo **samodejno ustvaril** pare ključev in digitalna potrdila za vzpostavitev varne povezave s strežnikom OPC UA.

Več o OPC UA agentu IoT **tukaj**.

10.6 Upravljanje digitalnih potrdil

10.6.1 Let's Encrypt

Let's Encrypt je globalni organ za izdajo digitalnih potrdil (*Certificate Authority* - CA). Ta digitalna potrdila se lahko uporabijo za omogočanje varnih povezav HTTPS. Način delovanja je opis **tukaj**.

Za (avtomatsko) pridobivanje in obnavljanje Let's Encrypt digitalnih potrdil je v Kong-u na voljo **vtičnik ACME**. Vklop/izklop tega vtičnika je najlažje izvesti v Kong Manager-ju (Plugins) oz. v sklopu vzpostavitvenega procesa, za kar poskrbi `init.sh` skripta.

10.6.2 Self-signed certifikati

Za namene testiranja varne povezave HTTPS lahko uporabimo lastno podpisana digitalna potrdila oz. t.i. self-signed certifikate.

10.6.2.1 Generiranje self-singed certifikatov

V spodnjem primeru bomo ustvarili svoj CA ("certificate authority") in iz njega ustvarili lastno podpisana potrdila. To pomeni, da bodo odjemalci morali imeti dostop do potrdila CA, da se omogoči preverjanje potrdil.

Ustvarjanje para digitalno potrdilo + ključ za naš CA.

```
openssl genrsa -out ca.key 4096
openssl req -new -x509 -days 3650 -key ca.key -out ca.pem
```

Slika 16: Ustvarjanje para digitalno potrdilo + ključ za naš CA.

Ustvarjanje ključa za željeno domeno (npr. `mju-iot.local`)

```
openssl genrsa -out mju-iot.local.key 2048
```

Slika 17: Ustvarjanje ključa za željeno domeno.

Ustvarjanje zahteve za podpis digitalnega potrdila (angl. certificate signing request, krajše CSR):

```
openssl req -new -key mju-iot.local.key -out mju-iot.local.csr
# Country Name (2 letter code) [AU]:SI
# State or Province Name (full name) [Some-State]:Osrednjeslovenska
regija
# Locality Name (eg, city) []:Ljubljana
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:IRI UL
# Organizational Unit Name (eg, section) []:FRI & FE
# Common Name (e.g. server FQDN or YOUR name) []:mju-iot.local
# Email Address []:admin@mju-iot.local

# A challenge password []:
# An optional company name []:IRI UL
```

Slika 18: Zahteva za podpis digitalnega potrdila (angl. certificate signing request,

Ustvarjanje datoteke mju-iot.local.ext z nastavitvami:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = *.mju-iot.local
```

Slika 19: Ustvarjanje datoteke mju-iot.local.ext z nastavitvami.

Podpisovanje digitalnega potrdila s (predhodno ustvarjenim) CA

```
openssl x509 -req -in mju-iot.local.csr -CA ca.pem -CAkey ca.key -
CAcreateserial -out mju-iot.local.pem -days 1825 -sha256 -extfile
mju-iot.local.ext
```

Slika 20: Podpisovanje digitalnega potrdila s (predhodno ustvarjenim) CA.

Končno stanje:

- ca.key - privatni ključ CA
- ca.pem - digitalno potrdilo CA
- ca.srl - serijska številka digitalnega potrdila CA
- mju-iot.local.csr - zahteva za podpis digitalnega potrdila
- mju-iot.local.ext - nastavitve za podpisovanje digitalnega potrdila
- mju-iot.local.key - privatni ključ digitalnega potrdila
- mju-iot.local.pem - digitalno potrdilo določene domene.

10.6.2.2 Uvoz lastnih digitalnih potrdil v Kong

V primeru uporabe lastnih digitalnih potrdil (npr. iz prejšnjega poglavja) je potrebno v Kong-u izvesti naslednje korake.

Prilagoditev docker-compose.yml

V glavnem docker-compose.yml je potrebno prilagoditi nastavitve za konggateway :

```
services:
  # ...
  kong-gateway:
    environment:
      # dodati sledeče okoljske spremenljivke in jih ustrezno
      nastaviti
      KONG_SSL_CERT: /certs/mju-iot.local.pem
      KONG_SSL_CERT_KEY: /certs/mju-iot.local.key
    volumes:
      # ...
      # mount mape, kjer so certifikati
      - ./kong-gateway/certs/local:/certs
```

Slika 21: Prilagoditev docker-compose.yml

Uvoz digitalnega potrdila CA

V Kong Manager-ju je potrebno uvoziti digitalno potrdilo CA:

New CA Certificate

CA Certificates verify the validity of a client or server certificate. [Learn more](#)

The screenshot shows the 'New CA Certificate' form in Kong Manager. It has three main sections: 'Public Certificate', 'General Information', and 'Tags'. The 'Public Certificate' section contains a 'Cert' input field with a red arrow pointing to it from the text 'ca.pem' written in red. The 'Cert' field contains a long base64-encoded string. Below it is a 'Cert Digest' field. The 'General Information' section has a 'General Information' label and a text area. The 'Tags' section has a 'Tags' label and a text area with a hint 'Enter a list of tags separated by comma' and an example 'e.g. tag1, tag2, tag3'.

Slika 22: Uvoz digitalnega potrdila CA.

Uvoz digitalnega potrdila strežnika

V Kong Manager-ju je potrebno uvoziti digitalno potrdilo in privatni ključ strežnika (domene), primer:

New Certificate

Certificates handle SSL/TLS termination for encrypted requests. [Learn more](#)

SSL Key Pair
The PEM-encoded public certificate chain and private key of the SSL key pair and the alternate.

Cert *

Key *

Cert Alt

Key Alt

Tags

General Information
General information will help identify and manage this key.

Slika 23: Uvoz digitalnega potrdila strežnika.

Nastavitev SNI

Digitalno potrdilo je potrebno s pomočjo SNI povezati z določeno (poddomeno), npr. *.mju-iot.local, primer:

New SNI

SNI are used to map hostnames to a certificate. [Learn more](#)

General Information
General information will help identify and manage this SNI.

Name *

Tags

Certificate
Map an existing Certificate object to hostnames

SSL Certificate ID *

Cancel **Save**

Slika 24: Nastavitev SNI.

Testiranje

Primer ukaza za testiranje uporabe digitalnih potrdil za vzpostavitev varne povezave HTTPS:

```
curl --cacert ./kong-gateway/certs/local/ca.pem -v https://kong-manager.mju-iot.local
```

Slika 25: Ukaz za testiranje uporabe digitalnih potrdil za vzpostavitev varne povezave HTTPS.

Opomba: Ker gre v zgornjem primeru za *self-signed* certifikat, je potrebno za uspešno preverjanje veljavnosti digitalnega potrdila strežnika dodati stikalo `--cert <lokacija-ca.pem-datoteke>` z digitalnim potrdilom CA.

Ob uspešni vzpostavitvi HTTPS povezave se izpiše:

```
* processing: https://kong-manager.mju-iot.local
*   Trying 127.0.0.1:443...
* Connected to kong-manager.mju-iot.local (127.0.0.1) port 443
* ALPN: offers h2,http/1.1
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
*   CAfile: ./ca.pem
*   CApath: /etc/ssl/certs
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN: server accepted http/1.1
* Server certificate:
*   subject: C=SI; ST=Osrednjeslovenska regija; L=Ljubljana; O=IRI
UL; OU=FRI & FE; CN=mju-iot.local; emailAddress=admin@mju-iot.local
*   start date: May 28 09:12:35 2024 GMT
*   expire date: May 27 09:12:35 2029 GMT
*   subjectAltName: host "kong-manager.mju-iot.local" matched cert's
"*.mju-iot.local"
*   issuer: C=SI; ST=Some-State; L=Ljubljana; O=IRI UL; CN=mju-
iot.local; emailAddress=admin@mju-iot.local
*   SSL certificate verify ok.
* using HTTP/1.1
> GET / HTTP/1.1
> Host: kong-manager.mju-iot.local
> User-Agent: curl/8.2.1
> Accept: */*
>
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=UTF-8
```

```
< Transfer-Encoding: chunked
< Connection: keep-alive
< Date: Tue, 28 May 2024 14:14:30 GMT
< Cache-Control: no-store, no-cache, must-revalidate, proxy-
revalidate, max-age=0
< X-Frame-Options: sameorigin
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< X-Permitted-Cross-Domain-Policies: master-only
< X-Kong-Upstream-Latency: 0
< X-Kong-Proxy-Latency: 1
< Via: kong/3.4.1.1-enterprise-edition
...
```

Slika 26: Uspešna vzpostavitev HTTPS povezave.

10.7 Upravljanje identitet in dostopov

10.7.1 Osnovni pojmi

10.7.1.1 Preverjanje pristnosti (avtentikacija)

Preverjanje pristnosti ali avtentikacija (angl. authentication) je v računalništvu postopek, s katerim se strežnik prepriča, da je uporabnik zares tisti uporabnik, za kogar se predstavlja, da je. Zgled preverjanja pristnosti je vpis uporabniškega imena in gesla pri npr. vpisu v sistem.

10.7.1.2 Nadzor dostopov (avtorizacija)

Nadzor dostopa ali avtorizacija (angl. authorization) je bistven element varnosti, ki določa, kdo lahko dostopa do določenih podatkov, aplikacij in virov – in v kakšnih okoliščinah. Podobno kot ključi in predhodno odobreni sezname gostov varujejo fizične prostore, pravilniki nadzora dostopa varujejo digitalne prostore.

Poznamo več modelov za nadzor dostopa, spodaj so na kratko predstavljeni trije izmed tradicionalnih modelov.

10.7.1.2.1 RBAC

Nadzor dostopa na podlagi uporabniške vloge (angl. Role-Based Access Control, krajše RBAC) dodeli uporabniku dostop do zaščitenega vira na podlagi njegove uporabniške vloge, na katero je vezan nabor pravic.

10.7.1.2.2 ABAC

Nadzor dostopa na podlagi atributov (angl. Attribute-Based Access Control, krajše ABAC), ovrednoti nabor pravil in politik za upravljanje pravic dostopa glede na specifične attribute, kot so informacije o okolju, sistemu, objektu, in/ali uporabniku. Uporablja enostavno logiko za odobritev ali zavrnitev dostopa, ki temelji na vrednotenju atributov in relacij med njimi.

10.7.1.2.3 PBAC

Nadzor dostopa na podlagi politik (angl. Policy-based Access Control, krajše PBAC). Rešitve, zgrajene po tem načinu, so običajno sestavljene iz sledečih gradnikov.

Policy Enforcement Point (PEP)

PEP prestreže uporabnikov zahtevke za dostop do določenega vira ter ga odobri ali zavrne. PEP ne sprejema odločitev; jih zgolj uveljavlja. Po potrebi lahko prilagodi vsebino zahtevka ali odgovora na zahtevek.

Policy Decision Point (PDP)

PDP na podlagi definiranih politik in dodatnih podatkov sprejme odločitev, ali se zahtevek odobri ali zavrne.

Policy Administration Point (PAP)

PAP je zadolžen za upravljanje politik, ki jih uporablja PDP.

Policy Information Point (PIP)

PIP je kateri koli vir podatkov (notranji ali zunanji), ki vsebuje attribute, pomembne za sprejemanje odločitev v skladu z definiranimi politikami.

Policy Retrieval Point (PRP)

PRP je glavni vir politik - shranjuje politike, ki jih uporablja PDP, upravlja pa ga PAP.

10.7.2 Upravljanje identitet in dostopov (IAM)

Upravljanje identitet in dostopov (angl. identity and access management, krajše IAM) je rešitev, ki zajema upravljanje uporabniških poverilnic (angl. user credentials), preverjanja pristnosti uporabnikov in nadzora dostopov glede na definirane uporabniške pravice. Njen glavni namen je varno hranjenje podatkov o uporabnikih ter omogočanje "pravih uporabnikom dostop do pravih virov ob pravem času iz pravih razlogov".

10.7.2.1 Dostopni žeton

Za avtentikacijo in avtorizacijo se pogosto uporablja preverjanje pristnosti na podlagi žetona (angl. token-based authentication). Odjemalec (npr. uporabnik, naprava IoT,...) v zameno za uporabniške poverilnice od strežnika za avtorizacijo (angl. authorization server) prejme dostopni žeton (angl. access token), ki mu omogoča dostop do zaščenega vira na strežniku virov (angl. resource server).

10.7.2.2 OAuth 2.0 in OpenID Connect

OAuth 2.0 je odprt standard za avtorizacijo, ki opisuje ogrodje za avtorizacijo (angl. authorization framework), znotraj katerega so definirani protokoli in potek avtorizacije. OpenID Connect (krajše OIDC) je odprt standard za avtentikacijo v obliki dodatnega sloja nad OAuth 2.0 protokolom. Namenjen je avtentikaciji uporabnikov, omogoča tudi t.i. single sign-on.

10.8 Upravljanje identitet in dostopov

10.8.1 Kong + Keycloak

V tem poglavju je na kratko predstavljen način preverjanja avtentikacije in avtorizacije s pomočjo Kong-a in Keycloak-a.

10.8.1.1 Vloge in proces

Vloge gradnikov za avtentikacijo in avtorizacijo so sledeče:

Vloge gradnikov za avtentikacijo in avtorizacijo so sledeče:

PEP - Kong z vtičnikom **pep-plugin**

PDP, PAP, PIP - Keycloak (glej tudi **arhitekturo**)

Proces preverjanja avtentikacije in avtorizacije:

1. Odjemalec s Keycloak-om izmenja uporabnikove oz. svoje uporabniške poverilnice za dostopni žeton.
2. Odjemalec pošlje HTTP zahtevek na platformo. V glavo zahtevka (HTTP header) pripne dostopni žeton; Authorization: Bearer <dostopni žeton> .
3. Kong prestreže zahtevek. Na podlagi predefiniranih poti (**Routes**) in storitev (**Gateway Services**) določi, kateri storitvi je potrebno posredovati zahtevek. Pred tem preveri, če je za določeno storitev omogočena instanca pep-plugin vtičnika (**Plugins**) oz. kateri drug vtičnik za avtentikacijo (**Basic Authentication** , **Key Authentication**).
4. V kolikor je avtentikacija za dostop do zahtevane storitve omogočena, pepplugin posreduje zahtevek za preverjanje pravic na Keycloak.
5. Keycloak na podlagi definiranih politik dostopa sprejme odločitev o odobritvi ali zavrnitvi zahtevka ter odločitev v obliki odgovora na zahtevek vrne pep-plugin - u.
6. Kong (s pomočjo vtičnika pep-plugin) na podlagi vsebine prejetega odgovora na zahtevek odobri (preusmeri zahtevek do določene storitve) oz. zavrne zahtevek odjemalca.

10.8.2 Kong pep-plugin

10.8.2.1 Osnovne informacije

Kong pep-plugin je bil razvit v sklopu FIWARE Kong vtičnikov. Ponuja možnost uporabe Kong-a kot t.i. PEP proxy-a v navezavi z različnimi PDP, med drugim je podprt tudi Keycloak.

Kong pep-plugin je bil razvit v sklopu FIWARE Kong vtičnikov. Ponuja možnost uporabe Kong-a kot t.i. PEP proxy-a v navezavi z različnimi PDP, med drugim je podprt tudi Keycloak.

10.8.2.2 Gradnja

Pred uporabo je vtičnik potrebno zgraditi. Vtičnik se zgradi in doda uradni Kong Docker sliki. Dopolnjena Docker slika se uporabi za namestitev platforme. Postopek in ukaz za gradnjo vtičnika je definiran v kong-gateway/Dockerfile . Ob karkšnihkoli spremembah izvirne kode vtičnika je potrebno na novo zgraditi Docker sliko za Kong.

10.8.2.3 Delovanje

Kong vtičnik pep-plugin izvede tri glavne korake:

1. Na podlagi poverilnic (`clientId` in `clientSecret`) **pridobi dostopni žeton**

Keycloak odjemalca storitve, do katere se želi z zahtevkom dostopati.

[keycloak.go getServiceAccountToken]

2. Opravi **poizvedbo za Keycloak vire** (angl. *resources*), ki so povezani s potjo, definirano v zahtevku (npr. `/iot/devices`). **[keycloak.go:**

getResourcesFromKeycloak]

3. **Pridobi dovoljenja** (angl. *permissions*) in **posreduje odločitev Keycloak-a**, ali naj se dostop do zahtevanega vira odobri ali zavrne. **[keycloak.go:**

checkPermission]

Ob pridobivanju dovoljenj se v *query parametrih* zahtevka, ki ga pep-plugin opravi na Keycloak , vključi tudi `claim_token` . T.i. **Claim token** (base64 zakodiran niz znakov) predstavlja dodatne zahteve, ki jih mora PDP (Keycloak) upoštevati ob evalvaciji dovoljenj za dostop do določenega vira.

S pomočjo *claim token*-a vtičnik Keycloak-u sporoči podatke o poti (`http.uri`) in metodi (`http.method`) prestreženega HTTP zahtevka. Poleg tega se lahko dodajo tudi drugi podatki, ki so definirani v nastavitvah vtičnika (`keycloakadditionalclaims`).

10.8.2.4 Uporaba in nastavitve

Predpogoj za uporabo pep-plugin vtičnika je pravilno skonfiguriran Keycloak odjemalec storitve, ki jo želimo zaščititi.

Za zaščito določene storitve je potrebno prilagoditi konfiguracijo Kong-a (npr. v `konggateway/deck/kong.yaml.j2`), primer:

```

services:
  - name: IoT-Agent-LoRa
    # ...
    plugins:
      - name: pep-plugin
        instance_name: "pep-plugin_iot-agent-lora-north"
        config:
          authorizationendpointtype: Keycloak
          authorizationendpointaddress: http://keycloak.{
mjuIot.domain }}
          keycloakrealm: mju-iot
          keycloakclientid: iot-agent-lora-north
          keycloakclientsecret: "t7W3jWmWD0DEaQaabxeJmCz9Z1YE0D8f"
          #keycloakadditionalclaims:
          # "http.fiware-service": "fiware-service"
          decisioncacheexpiryins: "20"
          keycloakresourcecacheexpiryins: "120"
        protocols:
          - http
          - https

```

Slika 27: Konfiguracija Kong-a za zaščito določene storitve.

Pri tem je potrebno ustrezno popraviti vsaj keycloakclientid in keycloakclientsecret ter instance_name .

Nastavitve določene instance vtičnika (npr. vklop/izklop) je možno upravljati (tudi) znotraj Kong Manager-ja pod Plugins :

Plugins

Plugins allow you to extend Kong's capabilities with features like rate limiting, authentication, and logging. [Learn more](#) [CF](#)

Filter					+ New Plugin	
Name	Applied To	Status	Ordering	Tags		
pep-plugin_iot-agent-lora-north pep-plugin	Service	Enabled <input checked="" type="checkbox"/>	Static	-		

Slika 28: Nastavitve določene instance vtičnika.

10.8.3 Keycloak

10.8.3.1 Odjemalci

Odjemalci (angl. clients) so entitete, ki lahko od Keycloak-a zahtevajo preverjanje pristnosti uporabnika. Najpogostejše so odjemalci aplikacije in storitve, ki želijo Keycloak uporabljati za lastno varnost in zagotavljanje rešitve enotne prijave. Odjemalci so lahko tudi entitete, ki želijo le zahtevati podatke o identiteti ali žeton za dostop, da lahko varno kličejo druge storitve v omrežju, ki so zaščitene s Keycloak-om.

V trenutni različici so v realm-u mju-iot za zaščito aplikacij vzpostavljeni naslednji odjemalci:

- `iot-agent-json-north` - zaščita Provisioning API-ja za agenta IoT JSON
- `iot-agent-lora-north` - zaščita Provisioning API-ja za agenta IoT LoRa
- `iot-agent-opcua-north-agent01` - zaščita Provisioning API-ja za agenta IoT
- OPC UA
- `iot-agent-ultralight-north` - zaščita Provisioning API-ja za agenta IoT
- Ultralight
- `orion` - zaščita API-ja za Context Broker Orion-LD
- `quantum-leap` - zaščita API-ja za QuantumLeap
- `grafana` zaščita API-ja za Grafano
- `external-iot-agents` - zaščita dostopov do aplikacij preko zunanjih agentov
- IoT

Vsi odjemalci uporabljajo `confidential access` način, kar pomeni, da so namenjeni uporabi v drugih aplikacijah in storitvah, ki lahko varno shranijo poverilnice odjemalca (`clientId` in `clientSecret`) ter jih podajo ob zahtevku za pridobivanje dostopnega žetona.

10.8.3.2 Uporabniške vloge

- RBAC
- FRI & FE Varnost
- Kje se definirajo na odjemalcih

10.8.3.3 Avtorizacija

PBAC

kje in kako se definirajo na odjemalcih

10.8.3.4 Politike dostopa po meri

Nashorn Engine

Javascript EC5 syntax

Evaluation API

namestitev (izgradnja JAR datoteke - skripta -> lokacija)

10.8.3.5 Upravljanje uporabnikov

Ustvarjanje novega uporabnika

Credentials

Uporabniške vloge

10.8.4 Zaščita dostopa do izpostavljenih gradnikov

10.8.4.1 Keycloak

Zaščita dostopov do lastnih virov (API, upravljavski portal, itd.) je podprta s strani gradnika.

Za upravljanje Keycloak-a ter prijavo v upravljavski portal Keycloak-a se uporabi administratorski račun, katerega uporabniško ime in geslo se ob vzpostavitvi platforme nastavi s pomočjo okoljskih spremenljivk KEYCLOAK_ADMIN in KEYCLOAK_ADMIN_PASSWORD (`.env.example`).

11 Primeri uporabe

11.1 Onboarding

V tem poglavju je predstavljen postopek vključitve novih naprav IoT in zunanjih agentov IoT v platformo.

11.1.1 Onboarding naprav IoT

Onboarding naprav IoT je potrebno izvesti na ustreznem agentu IoT.

Agent IoT je gradnik, ki omogoča, da skupina naprav IoT z uporabo lastno podprtih protokolov pošilja (senzorji) oz. prejema (aktuatorji) podatke iz Context Broker-ja. Obenem skrbi za avtentikacijo (in avtorizacijo) naprav IoT.

V nadaljevanju je primer za onboarding naprav na agentu IoT za naprave, ki podatke pošiljajo v obliki JSON.

11.1.2 Registracija skupine naprav

Za naprave IoT s podobnimi značilnostmi se lahko ustvari skupna konfiguracija za skupino naprav oz. service, za katero se določi tudi t.i. API key.

Config groups provides a template configuration for the all devices that belong to them. This allows to provision a set of devices with a single operation. They are identified by an apikey and a resource and mapped to a particular entity type. Once a measure is received by the IoT Agent, the apikey and resource are used to identify the config group to which the device belongs. The config group is used to map the measure to a particular entity type and to provide the information needed to interact with the Context Broker.

Primer zahtevka za registracijo skupine naprav:

```
curl --location '<iot-agent-url>/iot/services' \
--header 'fiware-service: <fiware-service>' \
--header 'fiware-servicepath: <fiware-service-path>' \
--header 'Content-Type: application/json' \
--data '{
  "services": [
    {
      "apikey":      "<api-key>",
      "cbHost":      "<context-broker-url>",
      "entity_type": "<type>",
      "resource":     "/iot/json",
    }
  ]
}'
```

Slika 29: Zahtevki za registracijo skupine naprav.

11.1.3 Autoprovisioning

Privzeto je vključen t.i. autoprovizioning naprav IoT:

If devices are not pre-registered, they will be automatically created when a measure arrives to the IoT Agent - this process is known as autoprovizioning. The IoT Agent will create an empty device with the group apikey and type – the associated document created in database doesn't include config group parameters. The IoT Agent will also create the entity in the Context Broker if it does not exist yet.

Za izklop avtomatskega provisioning-a naprav IoT je potrebno konfiguraciji določene storitve (v prejšnjem koraku) dodati "autoprovizion": false . V tem primeru je potrebno eksplicitno izvesti postopek za registracijo naprave.

11.1.4 Registracija naprave IoT

Registracija naprave doda novo napravo v t.i. device registry agenta IoT.

A device contains the information that connects a physical device to a particular entity in the Context Broker. Devices are identified by a device_id , and they are associated to an existing config group based in apikey matching. The IoT Agents offer a provisioning API where devices can be preregistered, so all the information about service and subservice mapping, security information and attribute configuration can be specified in a per device way instead of relaying on the config group configuration. The specific parameters that can be configured for a given device are described in the Device datamodel section.

Primer zahtevka za registracijo naprave:

```

curl --location '<iot-agent-url>/iot/devices' \
--header 'fiware-service: <fiware-service>' \
--header 'fiware-servicepath: <fiware-service-path>' \
--header 'Content-Type: application/json' \
--data '{
  "devices": [
    {
      "device_id": "motion013",
      "entity_name": "urn:ngsi-ld:Motion:010",
      "entity_type": "Motion",
      "timezone": "Europe/Berlin",
      "expressionLanguage": "jexl",
      "attributes": [
        {
          "name": "location",
          "type": "geo:point",
          "expression": "{coordinates: [lon,lat], type:
'\''Point'\''}"
        },
        {
          "name": "TimeInstant",
          "type": "DateTime",
          "expression": "ts|toisodate"
        },
        {
          "name": "lat",
          "type": "Number"
        },
        {
          "name": "lon",
          "type": "Number"
        },
        {
          "name": "ts",
          "type": "Number"
        }
      ]
    }
  ]
}'

```

Slika 30: Zahtevek za registracijo naprave.

11.1.5 Testna meritev

Za pošiljanje podatkov testne meritve na IoT agenta sta obvezna dva podatka, ki se podata v *query* parametru zahtevka:

- *API Key* (določen ob registraciji skupine naprav),
- identifikator naprave IoT (enolično definiran za to skupino naprav).

Če je vključen autoprovisioning naprav, se ob prvi meritvi naprava IoT na podlagi identifikatorja (parameter *i*) avtomatsko doda v register naprav na agentu IoT. V nasprotnem primeru se uporabi identifikator, ki je bil določen ob registraciji naprave v prejšnjem koraku.

Primer zahtevka za pošiljanje testne meritve na IoT agenta:

```
curl --location '<iot-agent-url>/iot/json?k=<api-key>&i=<device-id>' \
--header 'Content-Type: application/json' \
--data '{"c": "1"}'
```

Slika 31: Zahtevki za pošiljanje testne meritve na IoT agenta.

11.1.6 Ostale operacije

Za ostale *CRUD operacije* se priporoča branje dokumentacije agentov IoT (primer za [JSON agenta IOT](#)).

11.2 Onboarding zunanjega agenta IoT

V primeru, ko je agent IoT nameščen izven platforme, je potrebno poskrbeti za varen prenos podatkov med agentom in Context Broker-jem znotraj platforme, kar zahteva prenos podatkov po šifrirani povezavi (HTTPS) in preverjanje pristnosti (avtentikacijo) zunanjega agenta IoT.

11.3 Registracija zunanjega agenta IoT

V Keycloak-u se ustvari novega uporabnika, ki predstavlja agenta IoT:

1. V spustnem seznamu se izbere *mju-iot realm*.
2. V meniju Users se klikne na gumb za dodajanje uporabnika *Add user*.
3. Izpolni se polje *username*, npr. *iot-agent-ul-lendava*, in označi potrditveno polje *Email verified*.
4. V zavihku *Credentials* se uporabniku nastavi geslo (*Temporary = off*).
5. V zavihku *Role mapping* se dodajo uporabniške vloge (pravice), ki so potrebne za dostop do Orion LD Context Broker-ja. V spustnem seznamu se izbere *Filter by clients* ter označi *orion producer* in *orion consumer*.

11.4 Pridobivanje offline token-a

FIWARE agenti IoT za avtentikacijo uporabljajo t.i. *refresh token* ([vir](#)):

In *oauth2* based authentication, two types of tokens can be used depending on the availability in the IDM to be used. On one hand, the trust associated to the device or deviceGroup is a *refresh_token* issued by a specific user for the Context Broker client. The authentication

process uses the refresh_token grant type to obtain an access_token that can be used to authenticate the request to the Context Broker.

V trenutni implementaciji agentov IoT se predpostavlja, da *refresh token* nikoli ne poteče, tj. gre za t.i. *offline token* (vir):

At the time being the assumption is that the refresh_token is a not expiring offline_token (we believe this is the best solution in the case of IoT Devices, since injecting a refresh token look may slow down communication. Still, the developer would be able to invalidate the refresh token on the provider side in case of security issues connected to a token).

Za pridobivanje refresh token-a je potrebno izvesti postopek avtentikacije zunanjega agenta IoT, za kar se potrebuje:

- uporabniško ime in geslo Keycloak uporabnika, ki predstavlja zunanjega IoTagenta (prejšnji korak),
- poverilnice (clientId in clientSecret) za Keycloak odjemalca external-iotagents .

Poverilnice za Keycloak odjemalca external-iot-agents so na voljo v Realm: mjuiot -> Clients -> 'external-iot-agents' -> Credentials .

Za pridobivanje t.i. offline token-a je potrebno v zahtevek dodati scope offlineaccess .

Primer zahtevka za pridobivanje offline refresh token-a:

```
curl --location 'http://keycloak.mju-iot.local/realms/mju-
iot/protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'username=<iot-agent-username>' \
--data-urlencode 'password=<iot-agent-password>' \
--data-urlencode 'client_id=external-iot-agents' \
--data-urlencode 'client_secret=<client-secret>' \
--data-urlencode 'scope=offline-access' \
--data-urlencode 'grant_type=password'
```

Slika 32: Zahtevek za pridobivanje offline refresh token-a.

V odgovoru na zahtevek se offline refresh token nahaja v polju refresh_token , ta žeton nikoli ne poteče (refresh_expires_in ima vrednost 0).

Primer zahtevka za testiranje avtentikacije s pomočjo refresh token-a:

```
curl --location 'http://keycloak.mju-iot.local/realms/mju-iot/protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'refresh_token=<refresh-token>' \
--data-urlencode 'client_id=external-iot-agents' \
--data-urlencode 'client_secret=<client-secret>' \
--data-urlencode 'grant_type=refresh_token'
```

Slika 33: Zahtevek za testiranje avtentikacije s pomočjo refresh token-a.

11.5 Nastavitev avtentikacije na agentu IoT

FIWARE agenti IoT interno uporabljajo **FIWARE IoT Agent Node.js Library**, ki med drugim ponuja SDK za integracijo s Context Broker-jem. Za avtentikacijo z uporabo Keycloak-a se uporablja OAuth2 način (**implementacija**).

Če želimo, da se agent IoT ob dostopu do Context Broker-ja avtentificira, je potrebno prilagoditi konfiguracijo.

Agente se lahko konfigurira s pomočjo nastavitev v datoteki config.js ali s pomočjo okoljskih spremenljivk. Za nastavev avtentikacije agenta se uporabljajo **IOTA_AUTH_* okoljske spremenljivke**.

Primer konfiguracije (za docker-compose.yml):

```
environment:
  # ... ostale okoljske spremenljivke ...
  # # # AUTH # #
  - IOTA_AUTH_ENABLED=true
  - IOTA_AUTH_TYPE=oauth2
  - IOTA_AUTH_HEADER=Authorization
  - IOTA_AUTH_URL=http://keycloak.mju-iot.local # potrebno
  spremeniti
  - IOTA_AUTH_CLIENT_ID=external-iot-agents
  - IOTA_AUTH_CLIENT_SECRET=<client-secret> # potrebno
  spremeniti
  - IOTA_AUTH_TOKEN_PATH=/realms/mju-iot/protocol/openid-
  connect/token
```

Slika 34: Konfiguracija (za docker-compose.yml).

11.6 Uporaba offline token-a ob registraciji skupine naprav

V sklopu onboarding-a naprav IoT je potrebno ob registraciji skupine naprav dodati polje `trust`, v katerem se kot vrednost zapiše `offline refresh_token`, pridobljen v prejšnjih korakih.

Primer vsebine telesa zahtevka POST `/iot/services` :

```
{
  "services": [
    {
      "apikey": "4jggokgpepnvsb2uv4s40d59ov",
      "cbHost": "orion.mju-iot.local",
      "entity_type": "Thing",
      "resource": "/iot/json",
      "autoprovision": "false",
      "trust": "<offline-refresh-token>"
    }
  ]
}
```

Slika 35: Vsebine telesa zahtevka POST `/iot/services`.

11.7 Posodobitev offline token-a za skupino naprav

V primeru, ko se za neko skupino naprav pridobi nov `offline token` (npr. ob ponovem ustvarjanju žetona po preklicu), je potrebna posodobitev atributa `trust`.

Primer zahtevka za posodobitev osvežitvenega žetona za skupino naprav:

```
curl --location --request PUT '<iot-agent-url>/iot/services?
resource=<resource>&apikey=<api-key>' \
--header 'fiware-service: <fiware-service>' \
--header 'fiware-servicepath: <fiware-service-path>' \
--header 'Content-Type: application/json' \
--data '{
  "trust": "<offline-refresh-token>"
}'
```

Slika 36: Zahtevke za posodobitev osvežitvenega žetona za skupino naprav.

11.8 Preklic dostopa

Možnosti:

- Administrators can revoke offline tokens for individual users in the Admin Console in the Consents tab.
- Administrators can view all offline tokens issued in the Offline Access tab of each client.
- Administrators can revoke offline tokens by setting a revocation policy.

Primer za preklic dostopa, tj. preklic *offline token*-a za določenega agenta IoT:

- Realm: mju-iot -> Users
- V seznamu poiskati uporabnika, ki predstavlja ustreznega zunanjega agenta IoT, npr. *iot-agent-ul-lendava*.
- V zavihku Consents za Keycloak odjemalca *external-iot-agents* odstraniti soglasje (tri pikice v najbolj desnem stolpcu -> Revoke).

11.9 Omejevanje dostopa skupine naprav

V zgoraj opisanem postopku je možno omejevanje dostopa (npr. s preklicem veljavnosti *offline token*-a) izvesti le na nivoju vseh naprav IoT, katerih podatke pošilja določen zunanji agent IoT.

Ob drugačni organizaciji uporabe Keycloak-a bi se lahko *offline token* uporabljal na nivoju skupine naprav (in ne za celotni zunanji agent IoT kot trenutno).

V tem primeru bi moral za vsak zunanji agent IoT obstajati svoj Keycloak odjemalec (in ne skupni - *external-iot-agents*). Za vsako skupino naprav IoT bi se ustvaril svoj uporabnik, za katerega bi se pridobil *offline token* in uporabil v atributu *trust*.

Kljub temu, da onboarding naprav IoT poteka na (zunanjem) agentu IoT (nad katerim platforma nima neposrednega nadzora), bi ta način omogočal, da bi bilo možno na platformi omejiti dostop glede na skupine naprav IoT. Ob tem seveda obstaja teoretična možnost, da bi nekdo na zunanjem agentu vse naprave iz blokiranih skupin prestavil v skupino naprav, ki ima še vedno dostop.

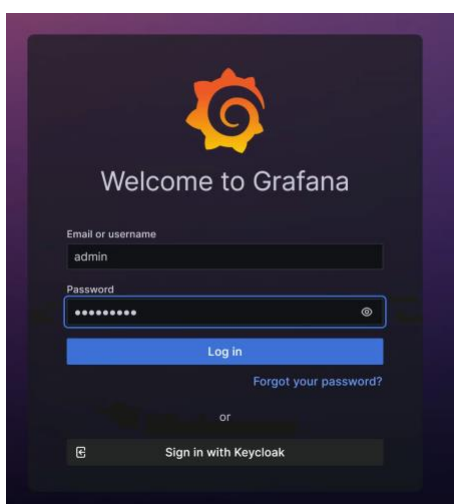
Naprednejši načini avtentikacije so predvideni v prihodnjih različicah platforme.

12 Uporaba orodja Grafana

Grafana je odprtokodna platforma za vizualizacijo podatkov in metrik zbranih iz različnih podatkovnih virov in aplikacij. Samo orodje je nameščeno v oblaku z možnostjo hitrega kreiranja podatkovnih nadzornih plošč, ki omogočajo pregledavanje in analizo vhodnih podatkov. Del funkcionalnosti orodja Grafana so tudi avtentikacija (angl. Authentication) in upravljanje z viri (angl. Provisioning).

12.1 Dostop do nadzorne plošče v orodju Grafana

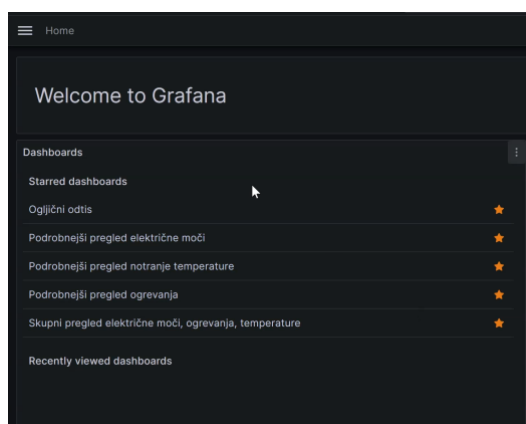
Uporabnika po zagonu orodja Grafana pričaka okno za vpisa uporabniškega imena in gesla. V primeru izvedene ustrezne konfiguracije je mogoče pravice za dostop do orodja Grafana urejati preko komponente Keycloak.



Slika 37: Okno za vpis uporabnikovih podatkov v orodju Grafana in dostop do nadzorne plošče.

12.2 Seznam nadzornih plošč

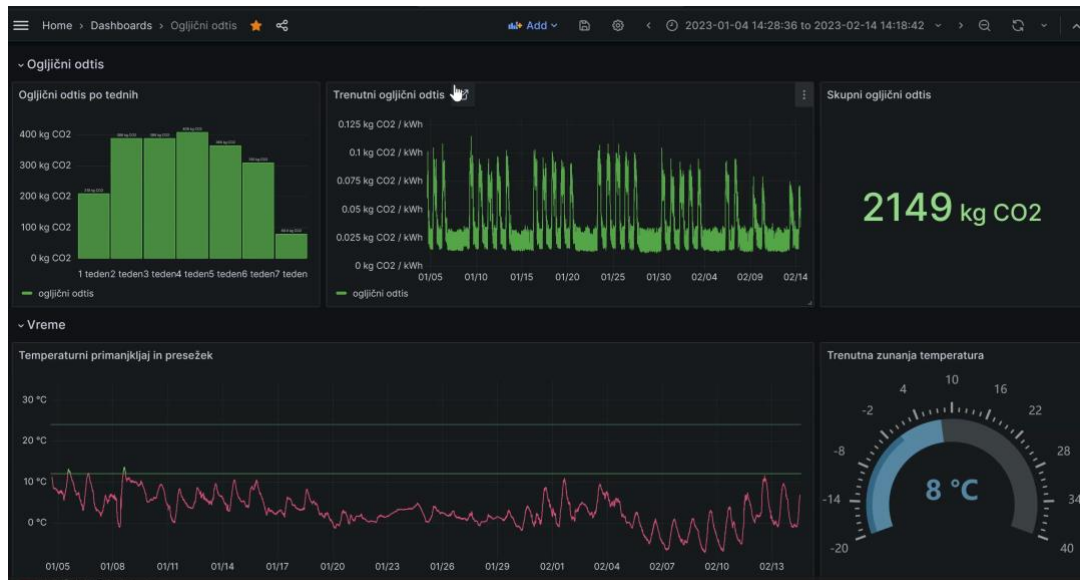
Po vpisu v orodje uporabnika pričaka seznam vseh nadzornih plošč kreiranih v orodju Grafana. Uporabnik izbere poljubno nadzorno ploščo za ogled ali pa izbere prvo nadzorno ploščo na seznamu in se preko povezav na nadzorni plošči premakne do drugih nadzornih plošč.



Slika 38: Primer nadzornih plošč v orodju Grafana.

12.3 Nadzorna plošča

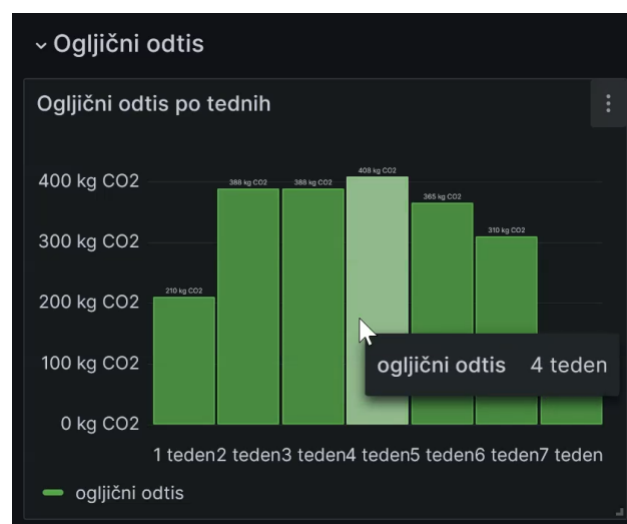
Po izbiri nadzorne plošče iz seznama se uporabniku izriše predhodno načrtovana nadzorna plošča njegove sistema. Prikazani elementi nadzorne plošče so bili določeni z namenom in zahtevami za prikaz podatkov.



Slika 39: Primer nadzorne plošče v orodju Grafana

12.3.1 Interaktivnost nadzorne plošče

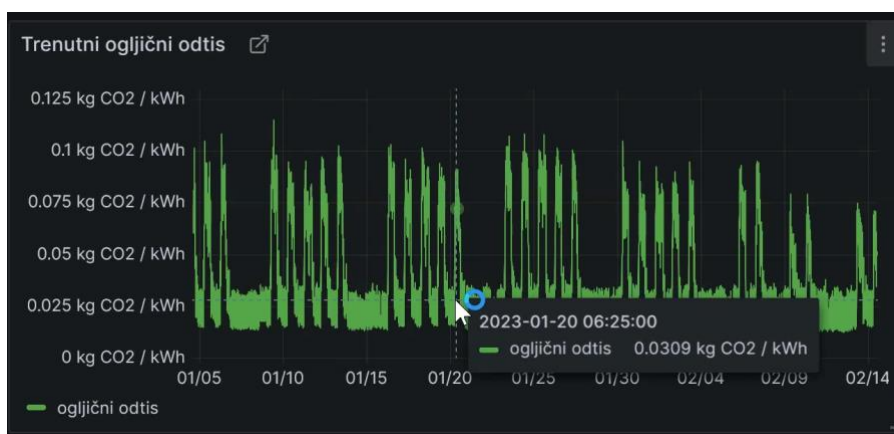
Vizualizacije podatkov na nadzorni plošči omogočajo interakcije z uporabnikom. Ob preletu s kazalcem računalniške miške se uporabniku izpišejo ali na drug način vizualizirajo dodatni podatki, komentarji ali vrednosti.



Slika 40: Interakcija grafov nadzorne plošče s komentarji.

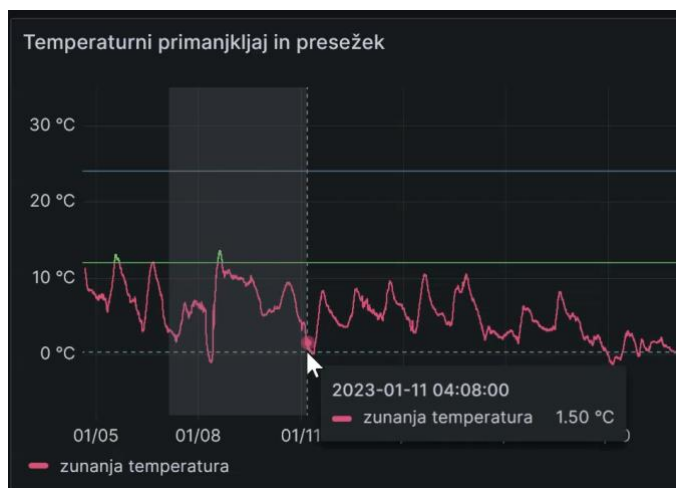
12.3.2 Podrobnejši prikaz podatkovnih vrednosti

Iz grafov ali drugih vizualizacije je mogoče razbrati podatke z večjo natančnostjo. Ob preletu s kazalcem računalniške miške preko grafa se sprotno izpisujejo točne vrednosti podatkov.



Slika 41: Podrobnejši prikaz podatkovnih vrednosti 1 .

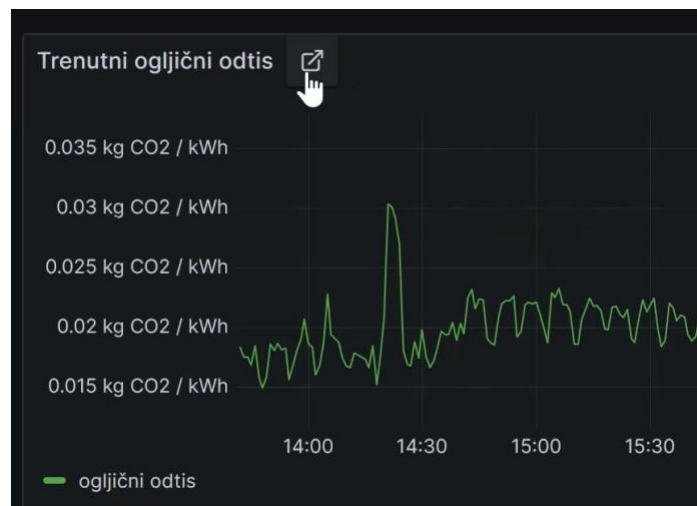
Vsebinsi grafov je mogoče pregledovati še na drug način. Če uporabnik z računalniško miško izbere del izrisanega grafa, se izbrani del, prikaže s povečano skalo. Postopek je mogoče ponavljati v primeru, da imamo na grafu izrisanih zadostno število podatkovnih točk.



Slika 42: Podrobnejši prikaz podatkovnih vrednosti 2.

12.3.3 Prehajanje med nadzornimi ploščami

Med nadzornimi ploščami je omogočeno tudi prehajanje, v primeru, da so na prvem grafičnem elementu prikazani agregirani podatki. Ob kliku na beli štirikotnik s puščico se pred uporabnikom odpre nova nadzorna plošča, ki vsebuje dodatne grafične elemente za prikaz podatkov.



Slika 43: Ikona za prehajanje med nadzornimi ploščami.

Prof. dr. Marko Bajec
