



# SADRŽAJ

<b>Popis slika</b>	<b>vi</b>
<b>Popis isječaka koda</b>	<b>vii</b>
<b>Popis tablica</b>	<b>ix</b>
<b>1. Uvod</b>	<b>1</b>
<b>2. Opis sustava i tehničkih zahtjeva</b>	<b>2</b>
2.1. Opis problematike u poljoprivredi - WIP naslov potpoglavlja . . . . .	2
2.2. Zahtjevi na sustav i opis predloženog rješenja . . . . .	2
<b>3. Razvojni sustav ESP32-C3-DevKitM-1</b>	<b>3</b>
3.1. Wi-Fi . . . . .	4
<b>4. Amazon Web Services (AWS)</b>	<b>9</b>
4.1. AWS IoT Core . . . . .	11
4.1.1. AWS IoT Fleet Provisioning . . . . .	14
4.1.2. AWS IoT Device Shadow . . . . .	18
4.1.3. AWS IoT Jobs . . . . .	21
4.1.4. AWS IoT OTA . . . . .	26
4.1.5. Ostale dostupne IoT usluge u sustavu AWS . . . . .	28
<b>5. Povezivanje razvojnog sustava i oblaka</b>	<b>31</b>
5.1. Programska potpora za mikrokontroler . . . . .	31
5.1.1. Dinamičko povezivanje mikrokontrolera na Wi-Fi . . . . .	32
5.1.2. Registracija u sustav AWS . . . . .	38
5.1.3. Očitavanje senzorskih mjerena . . . . .	43
5.1.4. Slanje očitanih podataka protokolom MQTT . . . . .	47
5.1.5. Sjena uređaja . . . . .	51

5.1.6. Ažuriranje softvera . . . . .	56
5.2. Programska potpora za oblak . . . . .	61
5.2.1. Dinamička registracija uređaja . . . . .	61
5.2.2. Obrada i pohrana dobivenih podataka . . . . .	63
5.2.3. Sjena uređaja . . . . .	69
5.2.4. Ažuriranje softvera . . . . .	72
<b>6. Web aplikacija</b>	<b>75</b>
6.1. Infrastruktura aplikacije . . . . .	75
6.2. Grafana . . . . .	79
<b>7. Zaključak</b>	<b>80</b>
<b>Literatura</b>	<b>81</b>

# POPIS SLIKA

2.1. Blok shema sustava . . . . .	2
3.1. Konfiguracija razvojnog sustava ESP32-C3-DevKitM-1 [14] . . . . .	3
3.2. Blok dijagram modula ESP32-C3 [12] . . . . .	4
3.3. Wi-Fi RF standardi [13] . . . . .	6
3.4. Primjer scenarija Wi-Fi povezivanja u načinu rada stanice [14] . . . . .	7
3.5. Primjer scenarija Wi-Fi povezivanja u načinu rada pristupne točke [14]	7
4.1. Odnos regija, zona dostupnosti i podatkovnih centara u AWS-u [4] . .	10
4.2. Arhitektura usluga AWS-a za IoT [3] . . . . .	11
4.3. Princip rada usluge AWS IoT Core [3] . . . . .	12
4.4. Komponente usluge AWS IoT Core [3] . . . . .	12
4.5. Različita stanja tijekom izvršavanja posla [3] . . . . .	24
5.1. Arhitektura unificiranog provizioniranja [11] . . . . .	33
5.2. Obavijest nakon skeniranja QR koda . . . . .	38
5.3. Odabir dostupne Wi-Fi mreže u blizini . . . . .	38
5.4. Tok registracije uređaja certifikatom zahtjeva [3] . . . . .	40
5.5. Razvojni sustav sa spojenom periferijom . . . . .	47
5.6. Pregled RMT odašiljača [14] . . . . .	52
5.7. Različite boje LED diode . . . . .	56
5.8. Popis politika dodijeljenih registriranom uređaju . . . . .	63
5.9. Sučelje za pregled podataka u bazi InfluxDB . . . . .	67
5.10. Sučelje za pregled stanja uređaja u bazi InfluxDB . . . . .	72
5.11. Popis dozvola uloge <i>OTARole</i> . . . . .	73
5.12. Stanja pokrenutog jednokratnog OTA posla . . . . .	74
6.1. AMI predložak i distribucija slika stroja na instance [4] . . . . .	76
6.2. Mrežna povezanost balansera i aplikacijske instance . . . . .	78

# POPIS ISJEČAKA KODA

4.1. Odjeljak <i>parametri</i> u predlošku za registraciju . . . . .	16
4.2. Odjeljak <i>resursi</i> u predlošku za registraciju . . . . .	17
4.3. Teme za sjene uređaja . . . . .	19
4.4. Poruke sjene uređaja . . . . .	19
4.5. Primjer formata tipa <i>cron</i> . . . . .	23
4.6. Teme za obavijesti o poslovima . . . . .	26
4.7. Teme za obavijesti o izvođenju poslova . . . . .	26
5.1. Stvaranje pristupne točke . . . . .	34
5.2. Generiranje QR koda iz pristupne točke . . . . .	36
5.3. Funkcija za prikaz QR koda na zaslonu . . . . .	36
5.4. Inicijalizacije memorije tipa SPIFFS . . . . .	41
5.5. Spajanje certifikatom zahtjeva i zahtjev za novim certifikatom . . . . .	42
5.6. Kontrolni zbroj poslanih bajtova s DHT11 . . . . .	44
5.7. FreeRTOS zadatak za očitanje senzorskih mjerena i slanje podataka . . . . .	46
5.8. Funkcije za pripremanje JSON objekta i njegovo slanje protokolom MQTT . . . . .	47
5.9. Funkcije za sinkronizaciju vremena s SNTP poslužiteljem . . . . .	50
5.10. JSON objekt za slanje na platformu . . . . .	51
5.11. Upravljanje LED diodom . . . . .	52
5.12. Proces ažuriranja sjene uređaja . . . . .	55
5.13. Rad OTA agenta . . . . .	59
5.14. Odjeljak <i>stvar</i> u predlošku za registraciju . . . . .	62
5.15. SQL upit rute za podatke s uređaja . . . . .	63
5.16. Podatak u bazi InfluxDB u formatu linijskog protokola . . . . .	66
5.17. Lambda funkcija za slanje podataka u InfluxDB . . . . .	68
5.18. SQL upit rute za sjene uređaja . . . . .	69
5.19. Poruka ažurirane sjene . . . . .	70
5.20. Lambda funkcija za preusmjeravanje poruka sjene uređaja . . . . .	70

6.1. Pokretanje Docker slike za Grafanu . . . . .	78
---	----

# POPIS TABLICA

4.1. Polja formata tipa <i>cron</i> [3] . . . . .	22
4.2. Stanja izvršenja posla [3] . . . . .	25
5.1. Povezivanje uređaja i LCD zaslona . . . . .	35
5.2. Spajanje uređaja i modula DHT11 . . . . .	43
5.3. Spajanje uređaja i senzora VMA303 . . . . .	43
5.4. Particije na uređaju . . . . .	57

# **1. Uvod**

Moj uvod.

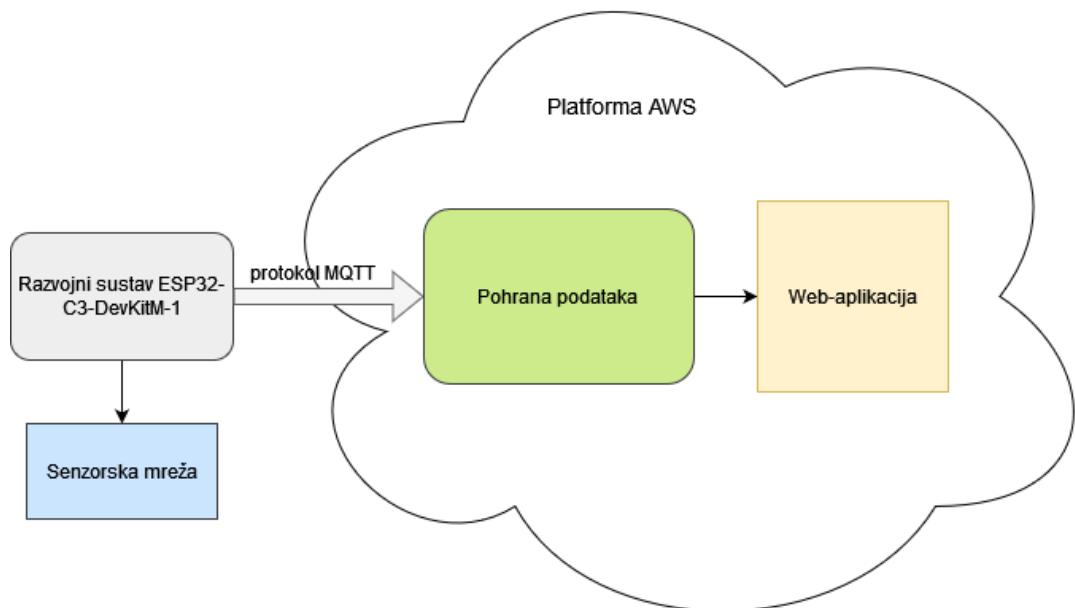
## 2. Opis sustava i tehničkih zahtjeva

### 2.1. Opis problematike u poljoprivredi - WIP naslov potpoglavlja

Moj problematični poljoprivredni sustav. Ovdje opisati sa čim se poljoprivrednici bore, šta im smeta, potkrijepiti to tako linkovima, onda naći rješenja koja su im pomogla i kako im IoT poljoprivreda boosta productivity i poboljšava usjeve i tako to.

### 2.2. Zahtjevi na sustav i opis predloženog rješenja

Ovdje je prikazana moja blok shema sustava koju trebam podrobnije opisati.

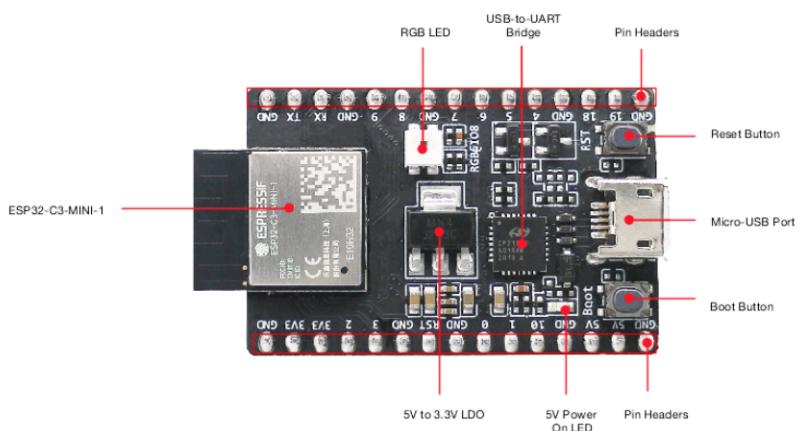


**Slika 2.1:** Blok shema sustava

## 3. Razvojni sustav

# ESP32-C3-DevKitM-1

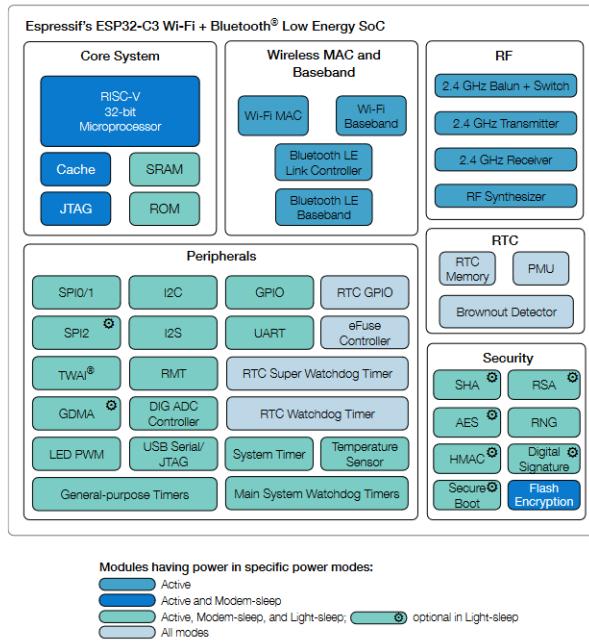
Razvojni sustav temelji se na modulu ESP32-C3-MINI-1. Modul je jedan u nizu ESP32-C3 serije SoC (engl. *System on Chip*) platformi tvrtke *Espressif*, te sadrži jednojezgreni 32-bitni procesor s RISC-V arhitekturom koji radi na frekvenciji do 160 MHz. Modul sadrži 400 KB memorije tipa SRAM (engl. *Static random-access memory*), od kojih je 16 KB rezervirano za priručnu memoriju (engl. *cache*), 384 KB memorije tipa ROM (engl. *Read-only memory*) te 4 MB memorije tipa *Flash*. Od periferije sadrži 22 programabilna GPIO pina (engl. *General Purpose Input Output*), te digitalna sučelja SPI, UART, I2C i I2S. Također sadrži upravljače za sučelja USB i JTAG koji se mogu koristiti za efikasnije otklanjanje pogrešaka u kodu (engl. *debugging*). Konfiguracija sustava prikazana je na slici 3.1. [12]



Slika 3.1: Konfiguracija razvojnog sustava ESP32-C3-DevKitM-1 [14]

Budući da modul ima funkciju RF (engl. *radio frequency*) primopredajnika, podržava bežično lokalno umrežavanje odnosno Wi-Fi, koji omogućava propusnost do 20 Mbps protokolom TCP te maksimalnu propusnost od 30 Mbps koristeći protokol UDP. Isto tako, podržava protokol Bluetooth s podrškom za velike udaljenosti.

Modul ESP32-C3-MINI-1 bežični je uređaj niske potrošnje energije (engl. *ultra-low-power*) primarno namijenjen razvoju aplikacija koje koriste Wi-Fi ili *Bluetooth Low Energy* (BLE) protokol. Na slici 3.2 nalazi se blok shema modula sa svim dostupnim značajkama.



**Slika 3.2:** Blok dijagram modula ESP32-C3 [12]

### 3.1. Wi-Fi

IEEE 802.11, skupina standarda za bežične lokalne mreže (engl. *WLANs*) [28], nudi nekoliko različitih načina bežične modulacije signala. Pojedini standardi označeni su slovima abecede. Za korisničke mreže postoje dva frekvencijska pojasa: 2,4 GHz i 5 GHz.

Prednosti pojasa od 2,4 GHz su veći doseg, bolje prolazanje kroz fizičke prepreke te bolja podrška jer više bežičnih uređaja koristi pojase od 2,4 GHz nego od 5 GHz. S druge strane, ovaj pojas ima manju propusnost i nudi manje kanala koji se ne preklapaju. Isto tako, može doći do zagruženja mreže jer kućni i *Bluetooth* uređaji koriste ovaj isti mrežni pojas.

Pojas od 5 GHz nudi brži protok, manje zagruženih kanala te ima više kanala koji se međusobno ne preklapaju. Ipak, ima kraći raspon u usporedbi s mrežama od 2,4 GHz jer teže prolazi kroz prepreke. [21]

U nastavku su opisani ključni standardi Wi-Fi tehnologije [7]:

- 802.11b - najsporiji i najjeftiniji standard, emitira u frekvencijskom pojasu od 2,4 GHz. Može prenijeti do 11 Mbps te koristi komplementarno šifriranje (engl. *complementary code keying - CCK*) radi poboljšanja brzine prijenosa.
- 802.11a - transmitira u pojasu od 5 GHz i može prenijeti do 54 Mbps. Koristi ortogonalno frekvencijsko multipleksiranje (engl. *orthogonal frequency-division multiplexing - OFDM*), što je efikasnija tehnika u odnosu na CCK koja dijeli radio signal u nekoliko podsignala prije slanja primatelju. Ova metoda značajno umanjuje interferenciju.
- 802.11g - poput standarda 802.11b, koristi frekvencijski pojas od 2,4 GHz. Međutim, može prenijeti do 54 Mbps jer koristi tehniku OFDM.
- 802.11n - kompatibilan je standard sa prethodno opisanim standardima. Nudi znatno poboljšanje u rasponu i brzini u odnosu na svoje prethodnike. Ovaj standard može prenijeti do četiri toka podataka, svaki maksimalno 150 Mbps, no većina usmjerivača (engl. *router*) dopušta dva ili tri toka.
- 802.11ac - radi isključivo u pojasu od 5 GHz, te je kompatibilan s prethodnim standardima. Manje je sklon interferenciji i brži je od prethodnih standarda s maksimalnim prijenosom od 450 Mbps jednim tokom.
- 802.11ax - najnoviji standard koji proširuje nekoliko ključnih mogućnosti svojih prethodnika. Usmjerivači koji podržavaju ovaj standard dopuštaju tok podataka do 9.2 Gbps, što je značajan porast u usporedbi s prethodnicima. Isto tako, moguće je postaviti više antena na jedan usmjerivač, čime je omogućen prihvati više veza odjednom bez usporavanja i interferencije.

Podsustav modula ESP32-C3 za Wi-Fi u skladu je sa standardom IEEE 802.11 te koristi nelicencirani pojas frekvencija od 2,4 GHz. U tom pojasu podržava propusnost od 20 i 40 MHz. Modul također podržava tehniku raznolikosti antena (engl. *antenna diversity*) za poboljšanje prijema i pouzdanosti signala korištenjem RF komutatora (engl. *switch*). Tim komutatorom upravljuju GPIO priključci i koristi se za odabir najbolje antene u kontekstu pouzdanosti i kvalitete signala. [13]

ESP32-C3 u potpunosti implementira protokol Wi-Fi na temelju standarda 802.11 b/g/n. Podržava osnovni skup (engl. *Basic Service Set - BSS*) operacija za značajke pristupne točke (engl. *software enabled access point - SoftAP*). Ovakve pristupne točke koriste softver kako bi omogućile uređajima kojima primarna svrha nije usmjeravanje prometa da postanu virtualni usmjerivač. [22] Također, upravljanje napajanjem odvija se automatski s minimalnom intervencijom domaćina kako bi se smanjila aktivnost uređaja.

Tvrtka *Espressif* također nudi biblioteke za povezivanje putem protokola TCP i IP te korištenje Wi-Fi *mesh* tehnologije. Pruža i podršku za protokole TLS 1.0, 1.1 i 1.2. Na slici 3.3 prikazani su Wi-Fi RF standardi koje koristi modul.

Name		Description
Center frequency range of operating channel <sup>1</sup>		2412 ~ 2484 MHz
Wi-Fi wireless standard		IEEE 802.11b/g/n
Data rate	20 MHz	11b: 1, 2, 5.5 and 11 Mbps 11g: 6, 9, 12, 18, 24, 36, 48, 54 Mbps 11n: MCS0-7, 72.2 Mbps (Max)
	40 MHz	11n: MCS0-7, 150 Mbps (Max)
Antenna type		PCB antenna and external antenna connector

**Slika 3.3:** Wi-Fi RF standardi [13]

ESP32 nudi nekoliko načina rada pri korištenju Wi-Fi tehnologije [29]:

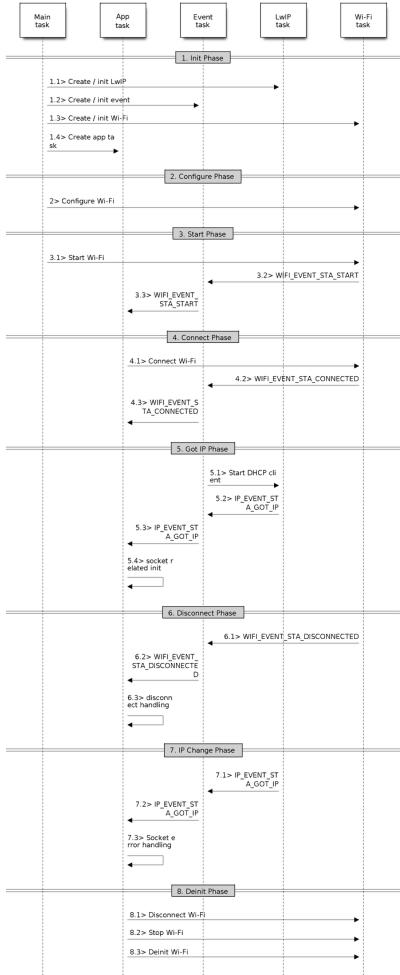
1. način rada stanice (engl. *station mode*) - ESP32 spaja se na točku pristupa,
2. način rada pristupne točke (engl. *SoftAP mode*) - druge se stanice spajaju na ESP32,
3. miješani - ESP32 radi kao stanica i pristupna točka spojena na drugu pristupnu točku.

U nastavku su opisani scenariji Wi-Fi povezivanja modula ESP32-C3 u načinu rada stanice i pristupne točke.

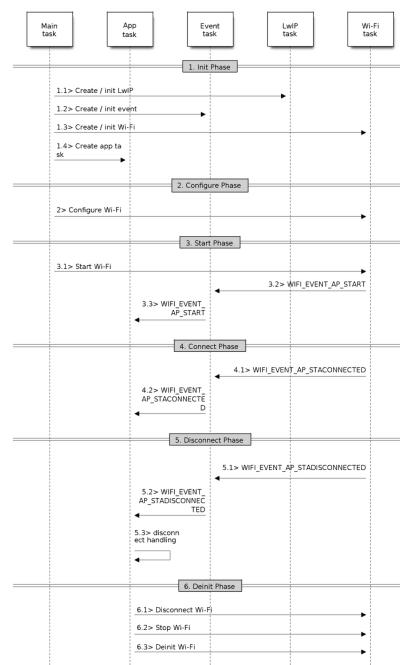
Na slici 3.4 prikazan je sekvenčni dijagram zadataka koje ESP32 obavlja u cijelom ciklusu spajanja i komunikacije s pristupnom točkom. Iz slike je vidljivo da se ciklus sastoji od osam faza. Prva faza služi za inicijalizaciju upravljačkih programa i pokretanje zadataka odnosno dretvi koje će obavljati zadatke vezane uz svoju dužnost. Glavni zadatak pokreće četiri različite dretve izvršavanja: aplikacijski zadatak, zadatak za događaje, zadatak za IP protokol, te zadatak za Wi-Fi. U drugoj fazi konfigurira se upravljački program za Wi-Fi. U sljedećoj se fazi pokreće upravljački program, nakon koje slijedi faza pretraživanja mreže i povezivanja na usmjerivač ili pristupnu točku. Nakon inicijalizacije DHCP klijenta, započinje faza dohvata IP adrese. Šesta faza odvija se nakon prekida Wi-Fi veze, čime se također uklanjuju i sve UDP i TCP konekcije. U aplikaciji se može omogućiti radno čekanje na ponovno uspostavljanje veze. Sedma faza pokreće se pri detekciji promjene IP adrese. Posljednja faza služi za programsko odspajanje s mrežom i zaustavljanje upravljačkog programa za Wi-Fi.

Slika 3.5 modelira slučaj u kojem ESP32 ima ulogu pristupne točke. Scenarij je vrlo sličan ranije opisanom slijedu događaja, no razlikuje se u dvije faze i događajima

koji su pohranjeni u sustavu. Ovaj način rada nema fazu detekcije promjene IP adrese, jer je u ovom načinu ESP32 upravo taj uređaj čija se IP adresa može promijeniti. Isto tako, ne postoji faza dohvata IP adrese.



**Slika 3.4:** Primjer scenarija Wi-Fi povezivanja u načinu rada stanice [14]



**Slika 3.5:** Primjer scenarija Wi-Fi povezivanja u načinu rada pristupne točke [14]

U modulu ESP32 stavljen je veliki naglasak na mehanizme uštade energije, što se također preslikava na korištenje Wi-Fi veze. Modul pruža načine uštade energije i pri radu kao stanica i pristupna točka, no neke značajke nisu podržane u pristupnoj točki. Modul pri neaktivnosti može otići u stanje mirovanja (engl. *sleep mode*). Postoje dva načina uštade energije u načinu rada stanice: minimalna i maksimalna ušteda. Pri minimalnoj uštedi stanica se budi iz stanja mirovanja nakon svakog DTIM intervala (engl. *Delivery Traffic Indication Message*). Ovim se načinom ne gube globalno emitirane poruke (engl. *broadcast*) jer se one prenose nakon DTIM intervala. Međutim, ova metoda ne štedi puno energije ako je pristupna točka na koju je spojen modul postavila

malen interval. Pri maksimalnoj uštedi moguće je znatno produžiti vrijeme mirovanja u odnosu na DTIM interval, no ovime se riskira gubitak globalno emitiranih poruka.

## 4. Amazon Web Services (AWS)

Amazon usluge za web (engl. *Amazon Web Services - AWS*) sveobuhvatna je platforma za računarstvo u oblaku koju pruža tvrtka Amazon te sadrži brojne usluge u oblaku, uključujući infrastrukturu (engl. *Infrastructure as a Service - IaaS*), platformu (engl. *Platform as a Service - PaaS*) i softver (engl. *Software as a Service - SaaS*). AWS usluge nude organizacijske alate kao što su računalna snaga, baza podataka i usluge isporuke sadržaja [25].

AWS je podijeljen u više različitih usluga koje se mogu pojedinačno konfigurirati na temelju korisničkih potreba. Neke od usluga koje nudi AWS su: pohrana, baze podataka, monitoriranje, sigurnost, analitika, umjetna inteligencija te razvoj mobilnih aplikacija.

AWS pruža usluge iz mnogo podatkovnih centara (engl. *data center - DC*) koji su raspodijeljeni po zonama dostupnosti (engl. *availability zone - AZ*) diljem regija cijelog svijeta. Jedna regija obuhvaća nekoliko fizički bliskih zona povezanih mrežom niske latencije. Geografskom raspodijeljenošću AWS pruža višestruke prednosti [1]:

1. niska latencija: budući da su regije skupovi fizičkih mrežno povezanih bliskih zona, korisnički podaci i aplikacije mogu biti smješteni bliže krajnjim korisnicima što poboljšava performanse aplikacija,
2. visoka dostupnost: u slučaju kvara podatkovnog centra, korištenjem više zona dostupnosti unutar regije podaci mogu i dalje ostati dostupni,
3. otpornost i oporavak od katastrofe: podaci i aplikacije mogu se replicirati između više zona ili regija, što omogućava brzi oporavak u slučaju prirodnih katastrofa, tehničkih problema ili napada,
4. skalabilnost: klijenti mogu dinamički povećavati ili smanjivati resurse u različitim regijama ovisno o potražnji,
5. poboljšana sigurnost: fizički odvojeni podatkovni centri smanjuju rizik od pojedinačnih točaka neuspjeha i omogućavaju implementaciju složenijih sigurnosnih

strategija.

Slika 4.1 prikazuje odnos regija, zona dostupnosti i podatkovnih centara unutar platforme AWS. Kao što je opisano, svaka regija ima više zona dostupnosti, i isto tako svaka zona ima nekoliko podatkovnih centara među kojima su podaci replicirani.



**Slika 4.1:** Odnos regija, zona dostupnosti i podatkovnih centara u AWS-u [4]

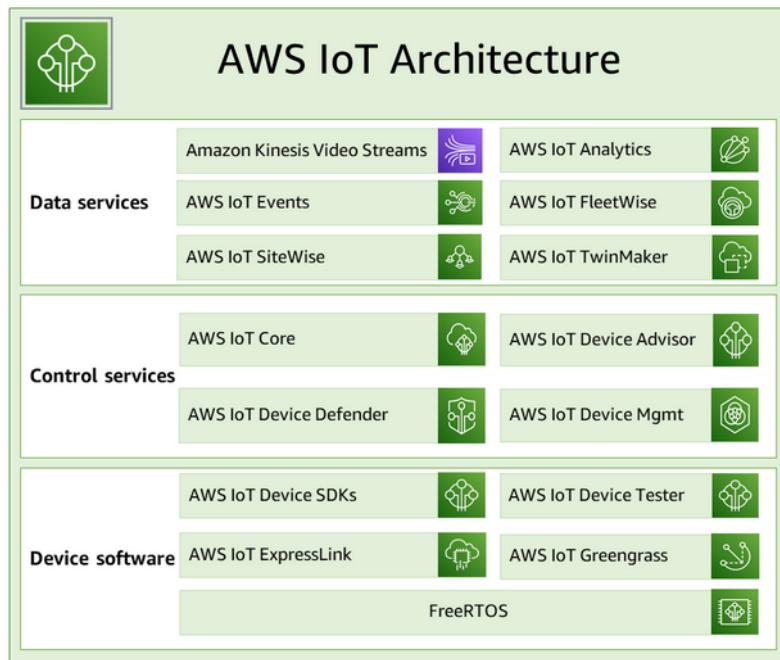
Također, nude se brojne mogućnosti za razvojne inženjere u sklopu AWS-a. Nudi alate naredbenog retka (engl. *command-line tools*) i pakete za razvoj programa (engl. *Software Development Kit - SDK*) za puštanje aplikacija u produkciju (engl. *deployment*) i upravljanje vlastitim uslugama i aplikacijama. Paketi za razvoj programa dostupni su u raznim programskim jezicima, uključujući programske jezike C++, Android, iOS, Java, Node.js, Python i Ruby.

AWS isto tako nudi brojne usluge za razvoj IoT sustava. Usluga AWS-a za IoT pruža platformu za upravljanje IoT uređajima te obradu podataka i njihovu pohranu na druge AWS usluge, poput baze podataka. AWS IoT pruža usluge u oblaku koje povezuju IoT uređaje s drugim uređajima i uslugama AWS-a u oblaku. Također pruža softver za uređaje, poput paketa za razvoj programa, za jednostavniju integraciju s uslugama AWS-a za IoT. Na slici 4.2 nalazi se prikaz arhitekture usluga koje AWS nudi za razvoj IoT sustava.

AWS podržava sljedeće komunikacijske protokole za IoT sustave:

- MQTT (engl. *Message Queuing Telemetry Transport*),
- HTTPS,
- LoRaWAN (engl. *Long Range Wide Area Network*),
- TLS.

U nastavku su opisane usluge koje nudi AWS za razvoj IoT sustava.

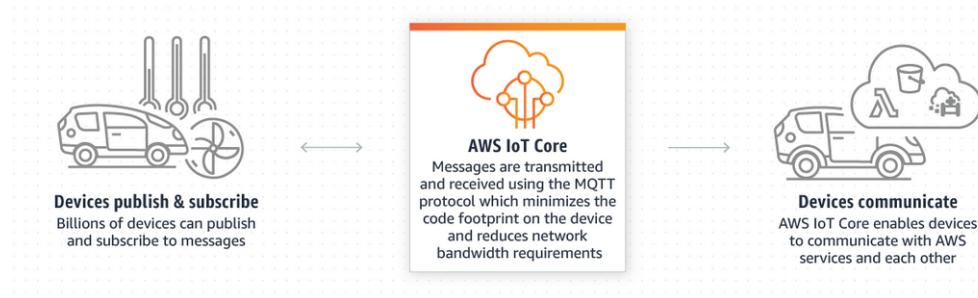


Slika 4.2: Arhitektura usluga AWS-a za IoT [3]

## 4.1. AWS IoT Core

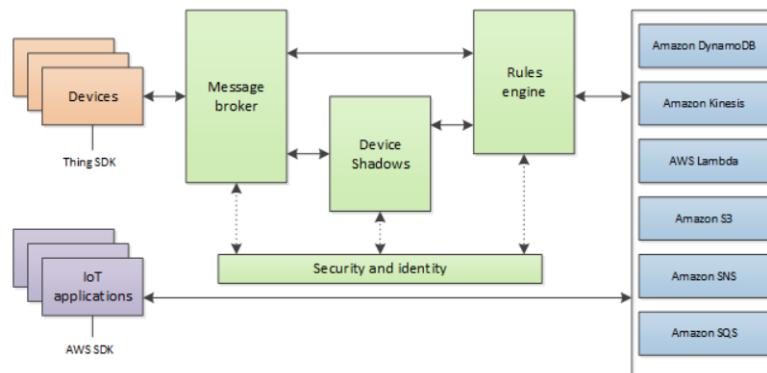
AWS IoT Core ključna je komponenta za integraciju oblaka i fizičkih uređaja. Omogućava povezivanje uređaja i preusmjeravanje poruka na usluge AWS-a. Koristi MQTT koji je standardni protokol za razmjenu poruka u IoT sustavima. To je lagan (engl. *lightweight*) protokol za prijenos poruka temeljen na objavi/preplati sustavu u kojoj glavnu ulogu ima broker kao posrednik, te je pogodan za povezivanje udaljenih uređaja uz minimalnu potrošnju [24]. AWS IoT Core pruža paletu značajki za razmjenu poruka temeljenih na protokolu MQTT, koje pomažu pri izradi prilagodljive i skalabilne IoT arhitekture [3]. Komponenta također ima ugrađenu podršku za upravljanje MQTT brokerom koji podržava trajne, uvijek uključene veze, napredna pravila zadržavanja poruka te obrađuje više uređaja i tema istovremeno. Isto tako, AWS IoT Core podržava najnoviji standard MQTT 5 i kompatibilan je s prethodnim standardom MQTT 3, što omogućuje učinkovito upravljanje heterogenim implementacijama s mnoštvom specifikacija. Nadalje, komponenta omogućava višestruke metode provjere autentičnosti i pristupne politike za zaštitu rješenja od ranjivosti. Štoviše, koristeći pomoć drugih usluga u sklopu platforme kao što je AWS IoT Core Device Advisor, moguće je pristupiti unaprijed izgrađenim paketima testova za provjeru MQTT funkcionalnosti uređaja tijekom faze razvoja.

Na slici 4.3 nalazi se pregled rada usluge AWS IoT Core.



**Slika 4.3:** Princip rada usluge AWS IoT Core [3]

AWS IoT Core pruža usluge koje povezuju oblak AWS-a s IoT uređajima kako bi se ostale usluge u oblaku i aplikacije mogle međusobno komunicirati s tim uređajima. Na slici 4.4 nalaze se svi segmenti usluge AWS IoT Core te kako oni komuniciraju s vanjskim dijelovima. Zelenom bojom označena je sama usluga, narančastom bojom fizički uređaji odnosno stvari (engl. *Things*), sivom bojom druge IoT aplikacije unutar AWS-a koje se mogu izravno spojiti na sustave u AWS-u, dok su plavom bojom označene ostale odnosno sustavi koji se nalaze u ekosustavu AWS.



**Slika 4.4:** Komponente usluge AWS IoT Core [3]

U nastavku su ukratko opisane ključne usluge koje pokriva AWS IoT Core [3].

### Usluge za slanje poruka

AWS IoT Core usluge za povezivanje pružaju sigurnu komunikaciju s IoT uređajima i upravlja porukama koje prolaze između uređaja i oblaka.

Prilazni uređaj omogućuje uređajima sigurnu i efikasnu komunikaciju sa sustavom AWS. Komunikacija je osigurana sigurnosnim protokolima koji koriste X.509 certifikate.

Broker za poruke pruža mehanizam uređajima i aplikacijama slanje i primanje poruka. Moguće je koristiti protokol MQTT ili direktno *WebSocket* za objavu i pretplatu na teme. Uređaji i klijenti koriste sučelje HTTP za objavu poruka brokeru. Broker zatim distribuira podatke na uređaje koji su se pretplatili na određene teme kao i na druge AWS aplikacije te usluge koje prate teme na brokeru.

AWS IoT Core za protokol LoRaWAN omogućava postavljanje privatne LoRaWAN mreže tako što poveže LoRaWAN te prilazne uređaje na AWS bez potrebe za razvojem mrežnog servera za LoRaWAN (engl. *LoRaWAN Network Server - LNS*). Poruke primljene od LoRaWAN uređaja šalju se na stroj za pravila (engl. *rules engine*) gdje se formatiraju i prosljeđuju ostalim AWS uslugama.

Stroj za pravila (engl. *rules engine*) povezuje podatke iz brokera s drugim AWS IoT uslugama za pohranu i dodatnu obradu. Primjerice, moguće je umetati ili pretraživati po podatkovnim tablicama ili pozvati određene definirane funkcije na temelju izraza definiranog u stroju. Isto tako, moguće je obraditi te podatke i proslijediti novostvoreni format poruka drugim uslugama ili bazama podataka.

## Upravljačke usluge

Upravljačke usluge AWS IoT Core komponente pružaju sigurnost uređaja te značajke za upravljanje i registraciju novih uređaja.

Moguće je definirati vlastite autorizatore radi upravljanja autentifikacijskim i autorizacijskim strategijama koristeći vlastiti servis za autentifikaciju i funkcije za računanje Lambda koje nudi AWS. Lambda je računalna usluga za slučajeve i aplikacije dinamičke skalabilnosti. Pokreće kod na infrastrukturi visoke dostupnosti i obavlja cje-lokupnu administraciju računalnih resursa, uključujući održavanje poslužitelja i operativnog sustava, osiguravanje kapaciteta i automatsko skaliranje ovisno o trenutnim potrebama sustava. Lambda funkcije korisne su za obradu datoteka, tokova te HTTP zahtjeva. Funkcije su pogodne za obradu podataka prije preusmjeravanja na drugu funkciju ili pohranu.

Usluga za registraciju uređaja u sustav (engl. *provisioning*) omogućava konfiguriranje i prijavu uređaja u AWS koristeći predložak koji opisuje resurse potrebne uređaju: stvar, certifikat i nekoliko politika. Stvar (engl. *thing*) je unos u registar koji sadrži atributе opisa uređaja. Uređaji koriste certifikate za autentifikaciju sa sustavom AWS. Politike određuju koje operacije uređaj može izvršiti. Svaka politika ima definirani dokument s izjavom koja se sastoji od učinka, akcije i resursa nad kojim se akcija izvršava. Učinak može biti *Dopusti* ili *Uskrati*, akcija se odnosi nad radnje s MQTT

brokerom i poslovima, a resurs se odnosi na regiju i račun u kojem politika vrijedi.

Isto tako, moguće je definirati grupe za lakšu kategorizaciju i upravljanje uređajima. Grupe također mogu imati podgrupe, i tako graditi hijerarhiju grupa. Sve akcije izvršene na roditeljima propagiraju se do najdubljih podgrupa. Dozvole dodijeljene grupi primjenjuju se na sve uređaje u toj grupi i svim njihovim podgrupama.

Usluga za poslove (engl. *jobs*) omogućava definiranje udaljenih operacija koje se pošalju i izvrše na fizičkim uređajima spojenih u oblak. Posao se može odnositi preuzimanje i instalaciju aplikacija, ažuriranje sustava, ponovno pokretanje, obnova certifikata i slično.

Usluga za sigurnost i identitet pruža dijeljenu odgovornost za sigurnost unutar AWS oblaka. Fizički uređaji moraju držati vjerodajnice na sigurnom kako bi se podaci mogli slati brokeru na siguran način. Značajke za sigurnost koriste i stroj za pravila kao i broker za poruke radi sigurnog prijenosa podataka drugim uslugama unutar AWS ekosustava.

### **Usluge za uređaje**

AWS IoT Core osigurava pouzdano aplikacijsko iskustvo iako uređaji nisu uvijek povezani.

Sjena uređaja (engl. *Device Shadow*) dokument je u JSON formatu koji se koristi za pohranu i dohvrat trenutnog stanja i informacija o uređaju. AWS nudi uslugu koja održava stanje uređaja (engl. *Device Shadow service*) kako bi aplikacije mogle komunicirati s uređajem bez obzira je li uređaj na mreži ili ne. Kada uređaj nije priključen na mrežu, usluga sjene uređaja upravlja podacima za povezane uređaje. Kada se uređaj ponovno spoji na mrežu, sinkronizira stanje sa sjenom uređaja koja se nalazi u oblaku. Uređaji također mogu objaviti svoje stanje usluzi u bilo kojem trenutku kako bi bilo na raspolaganju drugim aplikacijama i uređajima.

#### **4.1.1. AWS IoT Fleet Provisioning**

Kao što je ranije navedeno, AWS nudi uslugu registracije uređaja u sustav čime uređaj dobiva pristup ostalim uslugama unutar platforme, poput povezivanja na MQTT broker. Moguće je dinamički registrirati više uređaja pomoći privremenih ili trajnih certifikata, što se naziva provizioniranje flote (engl. *fleet provisioning*). Pomoći ove usluge AWS generira i sigurno dostavi certifikate te privatne ključeve na uređaje pri prvom povezivanju na platformu. AWS izdaje klijentske certifikate koji su potpisani certifikacijskim tijelom tvrtke Amazon (engl. *Certificate Authority - CA*) [3].

Tri su ključna koraka pri omogućavanju registracije više uređaja u sustav:

1. određivanje načina registracije,
2. definiranje upravljačke strukture nad uređajima,
3. kreiranje predloška za registraciju.

Određivanje načina registracije u sustav povezano je s odabirom vrste certificiranja koja će se koristiti. Uređaji trebaju jedinstveni certifikat za povezivanje s platformom AWS i korištenje njenih usluga. Postoje tri metode certificiranja uređaja te odabir ovisi o mogućnostima proizvodnje uređaja i ovlastima samih korisnika:

- registracija uređaja vlastitim jedinstvenim certifikatima,
- registracija od strane korisnika od povjerenja,
- registracija certifikatom zahtjeva.

Registracija uređaja vlastitim jedinstvenim certifikatima podrazumijeva postojanje verificiranih certifikata na samom uređaju pri prvom povezivanju u sustav. Ovaj se način još naziva i pravodobno provizioniranje (engl. *just-in-time provisioning*). Problem s ovim pristupom jest što instalacija certifikata nije uvijek moguća prije dostave uređaja korisniku. Isto tako, čak iako je instalacija certifikata moguća, nije uvijek moguće instalirati certifikate sigurno na uređaj, stoga je potrebno koristiti alternativne metode.

U slučaju kada rana certifikacija uređaja nije moguća, autorizirani ili krajnji korisnici (engl. *trusted users*) mogu uz pomoć aplikacije registrirati uređaje prije njihova spajanja na platformu. Ovdje je potrebno korisnicima pružiti aplikaciju kojom bi konfigurirali uređaj prilikom instalacije, te *firmware* uređaja mora podržavati ovaj način registracije.

Treća opcija jest registracija pomoću certifikata zahtjeva (engl. *claim certificate*). Certifikati zahtjeva privremeni su certifikati koji se mogu instalirati na više uređaja, te se pri prvoj registraciji u sustav zamijene s novim, jedinstvenim certifikatom. Ova metoda zahtjeva dodatne sigurnosne provjere zbog mogućnosti curenja privremenih certifikata zahtjeva jer više uređaja dijeli isti certifikat, no prijetnja se može ublažiti redovitim rotiranjem privremenih certifikata te kreiranjem više certifikata zahtjeva koji se mogu koristiti. Isto tako, privremenim certifikatima potrebno je dodijeliti minimalan skup dozvoljenih radnji potrebnih za registraciju uređaja. Dodatna sigurnosna provjera može se izvršiti i u obliku Lambda funkcije dodatnom provjerom primljenih parametara s uređaja.

Upravljačka struktura nad uređajima je važna radi lakše organizacije uređaja u samom sustavu. Povezani uređaji predstavljeni su u platformi kao stvari tj. resursi koji se mogu organizirati i održavati. Osim ranije spomenutih stvari i grupa stvari, moguće je uređajima dodijeliti atributi po kojima se može pretraživati. Atributi se isto tako mogu dinamički dodijeliti na temelju podataka s uređaja prilikom njegove registracije. Oni se definiraju u predlošku za registraciju.

Predložak za registraciju dokument je u formatu JSON koji opisuje resurse, politike i dozvole koje je potrebno dodijeliti uređaju kada je registriran. Odjeljak *parametri* u predlošku definira resurse u odjeljku *resursi* koje uređaj mora koristiti pri interakciji s platformom. Svaki parametar definira naziv, vrstu i zadanu vrijednost koja nije obavezna. Zadana vrijednost koristi se kada rječnik proslijeden s predloškom ne sadrži vrijednost za parametar. Odjeljak *parametri* izgleda na sljedeći način:

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "HR"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

**Isječak koda 4.1:** Odjeljak *parametri* u predlošku za registraciju

Odjeljak *resursi* u predlošku definira resurse koji su potrebni uređaju za daljnji rad i komunikaciju sa sustavom: stvar, certifikat, jedna ili više politika. Svaki resurs specificira naziv, vrstu i skup svojstava. Vrsta resursa može biti jedna od tri vrijednosti: AWS::IoT::Thing, AWS::IoT::Certificate, te AWS::IoT::Policy. Resursi tipa certifikat imaju posebna svojstva koja ih definiraju, ovisno o

tome koja je metoda registracije uređaja odabrana. Politike su vezane za već postojeće politike u sustavu AWS. Odjeljak *resursi* može izgledati na sljedeći način:

```
{
  "Resources" : {
    "thing" : {
      "Type" : "AWS::IoT::Thing",
      "Properties" : {
        "ThingName" : { "Ref" : "ThingName" },
        "AttributePayload" : { "version" : "v1", "serialNumber" : { "Ref" : "SerialNumber" } },
        "ThingTypeName" : "lightBulb-versionA",
        "ThingGroups" : [ "v1-lightbulbs", { "Ref" : "Location" } ]
      },
      "OverrideSettings" : {
        "AttributePayload" : "MERGE",
        "ThingTypeName" : "REPLACE",
        "ThingGroups" : "DO NOTHING"
      }
    },
    "certificate" : {
      "Type" : "AWS::IoT::Certificate",
      "Properties" : {
        "CertificateSigningRequest" : { "Ref" : "CSR" },
        "Status" : "ACTIVE"
      }
    },
    "policy" : {
      "Type" : "AWS::IoT::Policy",
      "Properties" : {
        "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
          \"Statement\": [{ \"Effect\": \"Allow\", \"Action\": [
            \"iot:Publish\" ], \"Resource\": [ \"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\" ] } ] }"
      }
    }
  }
}
```

```
    }  
}  
}
```

#### Isječak koda 4.2: Odjeljak *resursi* u predlošku za registraciju

Valja napomenuti kako je moguće u korisničkom sučelju platforme odabrati sve parametre, te kreiranje vlastitog dokumenta u JSON formatu nije potrebno, nego se automatski generira iz korisničkog odabira u sučelju.

### 4.1.2. AWS IoT Device Shadow

Usluga sjene uređaja (engl. *Device Shadow*) omogućava održavanje trenutnog stanja uređaja, čak i kada su uređaji privremeno nedostupni. AWS nudi uslugu koja održava stanje uređaja kako bi aplikacije mogle komunicirati s uređajem bez obzira je li uređaj na mreži ili ne. Kada uređaj nije priključen na mrežu, usluga sjene uređaja upravlja podacima za povezane uređaje. Kada se uređaj ponovno spoji na mrežu, sinkronizira stanje sa sjenom uređaja koja se nalazi u oblaku. Uređaji također mogu objaviti svoje stanje usluzi u bilo kojem trenutku kako bi bilo na raspolaganju drugim aplikacijama i uređajima. Ova funkcionalnost ključna je za slučajeve u kojima uređaji nemaju konstantnu povezanost na mrežu ili rade u udaljenim ili nepouzdanim mrežnim uvjetima.

Stvari (engl. *things*) u platformi nemaju nikakvu sjenu dok se eksplicitno ne kreiraju. Sjene se mogu stvarati, ažurirati i brisati pomoću sučelja, no i sami uređaji protokolom MQTT mogu slati poruke na rezervirane teme te tako mijenjati vlastitu sjenu. Budući da AWS pohranjuje sjene u oblaku, podaci o stanju uređaja mogu se slati drugim aplikacijama i uslugama u oblaku bez obzira je li uređaj povezan ili ne. Dok su uređaji, aplikacije i druge usluge u oblaku povezane s AWS IoT, mogu pristupiti trenutnom stanju uređaja i kontrolirati ga putem njegovih sjena. Traženjem promjene stanja na uređaju ažuriranjem sjene, AWS objavljuje poruku koja označava promjenu na uređaju, čime signalizira uređaju da ažurira svoje stanje sukladno zatraženom i objavljuje poruku s ažuriranim stanjem. Ukoliko dođe do pucanja internetske mreže na uređaju, pri ponovnom spajanju uređaj dohvata novo stanje te interno ažurira dosadašnje.

Usluga podržava imenovane i neimenovane, odnosno klasične, sjene. Stvar može imati više imenovanih, no najviše jednu klasičnu sjenu. Također može imati rezerviranu imenovanu sjenu koja radi slično kao i imenovana sjena, no nema mogućnost promjene imena. Pomoću imenovanih sjena moguće je stvoriti različite poglede na stanje uređaja odnosno stvari. Primjerice, stvar s puno parametara ili svojstava može se

podijeliti u sjene s logičkim skupinama parametara, a svaka sjena ima naziv prikladan skupini svojstava. Istom se metodom može ograničiti pristup pojedinim svojstvima uređaja tako što samo određene aplikacije imaju pristup imenovanoj sjeni, dok druge nemaju. Pristup imenovanim i neimenovanim sjenama razlikuje se u MQTT temi pomoću koje AWS i uređaj komuniciraju. U nastavku su navedene obje teme.

```
$aws/things/{thingName}/shadow  
$aws/things/{thingName}/shadow/name/{shadowName}
```

#### Isječak koda 4.3: Teme za sjene uređaja

Svaka sjena ima rezerviranu MQTT temu i HTTP URL koji podržava akcije dohvata, ažuriranja i brisanja sjene. Koriste dokumente u formatu JSON za pohranu i dohvat podataka. Dokument sadrži svojstvo stanja koji pobliže opisuje stanje te koja je svrha samog dokumenta. Tako se razlikuju tri svojstva stanja:

- željeno (engl. *desired*) - postavljaju aplikacije kako bi naznačile novo stanje uređaja koje žele,
- prijavljeno (engl. *reported*) - stanje koje je uređaj prijavio,
- delta - razlika između željenog i prijavljenog, određuje AWS.

Korištenje sjene uređaja u više aplikacija i drugih usluga zahtijeva konzistentnost i koordinaciju između usluga. Sve aplikacije koje koriste sjenu uređaja moraju redovito pratiti i ažurirati podatke na temelju tog stanja. Isto tako, budući da je samo stanje uređaja dinamično, potrebno je držati se određenih naputaka. Uređaji bi trebali jedino slati svojstvo prijavljenog stanja pri komunikaciji s uslugom, dok bi ostale usluge trebale koristiti isključivo svojstvo željenog stanja kako bi podnijeli zahtjev za promjenom stanja.

Nadalje, ne postoji garancija da će poruke dospjeti u AWS pravilnim vremenskim poretkom. Zato je važno da svaki dokument sjene uređaja sadrži vremensku oznaku (engl. *timestamp*) kako bi AWS mogao razlučiti koja je poruka ažurnija. Isto tako, svaki dokument stanja ima verziju, čime se jednostavno prati koja je poruka novija. Primjerice, ukoliko stignu dvije željene poruke jedna za drugom od različitih aplikacija, u AWS-u nastat će dva delta dokumenta, po jedan za svako željeno stanje uspoređeno s prijavljenim. Međutim, delta poruka s nižom verzijom bit će odbačena i ostat će samo novije željeno stanje koje će se ažurirati kao novo postojeće. U nastavku je dan primjer za prijavljenu, željenu i delta poruku.

```
// reported state  
{
```

```

"state": {
    "reported": {
        "color": "blue"
    }
},
"version": 9,
"timestamp": 123456700
}
// desired state
{
    "state": {
        "desired": {
            "color": "RED"
        }
    },
    "version": 10,
    "timestamp": 123456701
}
// the resulting delta message
{
    "state": {
        "color": "RED"
    },
    "version": 11,
    "timestamp": 123456702
}

```

#### Isječak koda 4.4: Poruke sjene uređaja

Isto tako, moguće je i postaviti klijenski token, koji mora biti jedinstven za svako ažuriranje sjene, no može se ponovno koristiti po završetku ažuriranja. Njegova je glavna funkcija omogućavanje praćenja i identifikacije specifičnih zahtjeva i odgovora između klijenta i usluge AWS. Tokenom se mogu povezati željene poruke s rezultirajućim stanjima. Također služe za praćenje odakle zahtjevi za promjenom dolaze.

### 4.1.3. AWS IoT Jobs

Pokretanje poslova u sklopu usluge AWS IoT Jobs, kao što je ranije spomenuto, služi za udaljene operacije na fizičkim uređajima povezanih s platformom.

Posao je udaljena operacija koja se šalje i izvodi na jednom ili više uređaja povezanih s uslugom AWS IoT. Primjerice, moguće je definirati posao koji upućuje skup uređaja da preuzmu i instaliraju aplikaciju ili pokreću ažuriranja, rotiraju certifikate te je moguće udaljeno rješavati nastale probleme na uređajima (engl. *remote troubleshooting*). Da bi se kreirao posao, najprije je potrebno kreirati dokument posla (engl. *job document*) koji je opis udaljenih operacija koje će uređaji izvesti. Dokumenti posla sadrže informacije koje su potrebne uređajima za obavljanje posla te su u formatu JSON. Dokumenti se mogu pohraniti na platformu i posebno dohvaćati, no isto tako mogu se definirati uz samu naredbu koja izvršava posao. Pri kreiranju posla, potrebno je specificirati odredišne uređaje (engl. *targets*) koji trebaju izvršiti naredbe. Moguće je odabrati pojedinačne uređaje, ali i grupu uređaja. Usluga zatim pošalje poruku svakom uređaju da postoji posao dostupan za izvršavanje. Nakon kreiranja posla, dokument posla se postavlja na udaljene uređaje koje je potrebno ažurirati. Ciljni uređaj preuzima taj certifikat i time započinje izvršavanje posla. Izvodi operacije opisane u dokumentu i o napretku obaveštava platformu. Broj izvršenja (engl. *execution number*) jedinstveni je identifikator izvršenja posla na određenom ciljnom uređaju. Usluga izvršavanja posla pruža naredbe za praćenje napretka izvršenja posla na uređaju kao i na svim uređajima.

Na temelju broja ponavljanja posla, razlikuju se dvije vrste poslova:

1. jednokratni posao (engl. *snapshot job*): posao se šalje svim odabranim ciljnim uređajima pri kreiranju posla, te nakon odrađivanja posla ili odgovora o nemogućnosti izvršenja, posao se smatra gotovim,
2. kontinuirani posao (engl. *continuous job*): definiran je na razini grupe stvari, te se pri dodavanjem svakog novog uređaja u grupu izvršava na dodanom uređaju.

Preporučljivo je poslove označiti kontinuiranim jer se tako omogućava izvršavanje posla na novododanim uređajima čak i nakon kreiranja posla.

Moguće je odrediti koliko se brzo ciljni uređaji obaveštavaju o zakazanom izvršenju posla. To omogućuje stvaranje postupnog uvođenja (engl. *staged rollout*) radi boljeg upravljanja ažuriranjima, ponovnim pokretanjem i drugim operacijama. Konfiguraciju uvođenja moguće je izraditi koristeći statičku ili eksponencijalnu stopu uvođenja. Za navođenje maksimalnog broja uređaja koji se mogu obavijestiti u minuti, potrebno je koristiti statičku stopu.

Raspored poslova omogućuje raspoređivanje vremenskog okvira uvođenja dokumenta posla na sve ciljne uređaje. Osim toga, moguće je stvoriti vremenski okvir održavanja sa specifičnim datumima i vremenima unutar kojeg se dokument posla šalje na sve uređaje. Taj se vremenski okvir održavanja provodi na proizvoljnoj vremenskoj bazi, primjerice tjedno, ili pak na određene prilagođene datume koji se odaberu pri inicijalnom postavljanju posla. Samo se kontinuirani poslovi mogu izvršavati periodički tokom održavanja budući da se oni mogu ponavljati. Maksimalno trajanje ponavljanja vremenskog okvira održavanja je 23 sata i 50 minuta. Jednokratni poslovi također se mogu uvrstiti u raspored poslova, no njima nije omogućeno izvršavanje unutar vremenskog okvira održavanja, nego jednokratno u definiranom vremenskom trenutku.

Za zakazane poslove koji se izvršavaju tijekom prozora održavanja s ručno postavljenom učestalošću izvršavanja, učestalost odnosno frekvencija definira se izrazom u formatu posla *cron*. U operacijskim sustavima tipa *Unix*, posao tipa *cron* jest zadatak koji se kreira pomoću alata *cron*, što je alat za zakazivanje i automatizaciju budućih poslova [20]. Pomoću njih automatizira se održavanje sustava, nadziranje diska, kao i pravljenje sigurnosnih kopija. Poslovi tipa *cron* imaju određenu sintaksu i strukturu koja omogućava da se izvršavanje skripti pravilno izvršava. Sintaksa tipa *cron* sastoji se od pet obaveznih polja međusobno odvojenih razmakom. Tablica 4.1 prikazuje polja koja se moraju definirati.

**Tablica 4.1:** Polja formata tipa *cron* [3]

Polje	Vrijednosti	Zamjenski znakovi
Minuta	0-59	, - * /
Sat	0-23	, - * /
Dan u mjesecu	1-31	, - * ? / L W
Mjesec	1-12 ili JAN-DEC	, - * /
Dan u tjednu	1-7 ili MON-SUN	, - * ? L #

U tablici se isto tako nalazi stupac za zamjenske znakove. Zamjenski znakovi (engl. *wildcards*) služe upravo kao zamjena sa konkretnu vrijednost za ona polja čija vrijednost nije bitna u kontekstu izraza. Zarez se koristi za odvajanje više vrijednosti, crtica označava opseg od prve do posljednje definirane vrijednosti, a zvjezdica je zamjena za sve vrijednosti polja. Kosa crta označava pojedinačne inkremente, a upitnik navodi jedno ili drugo. Slovo *L* označava posljednji dan u tjednu ili mjesecu, a *W* radni dan. Ljestve specificiraju pojedinačnu instancu unutar mjeseca, primjerice svaki drugi ponedjeljak u mjesecu. U nastavku je primjer posla u formatu *cron* koji se pokreće

svake minute između 15:00 i 15:59, svaki drugi dan u mjesecu, no samo u siječnju i veljači:

```
// min hr day month weekday  
* 15 2-30/2 JAN,FEB *
```

**Isječak koda 4.5:** Primjer formata tipa *cron*

Moguće je postaviti otkazivanje uvođenja posla na temelju postavljenih kriterija. Poslovi se mogu otkazati ako je previše uređaja vratilo negativnu potvrdu o izvršenju posla, ili pak previše uređaja nije poslalo nikakav odgovor do isteka određenog vremenskog perioda.

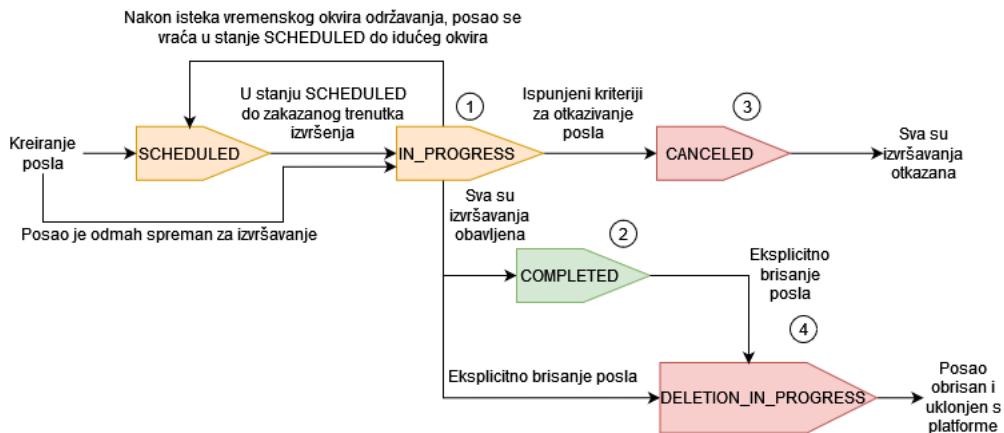
Pri isteku vremena (engl. *timeout*) za posao šalje se obavijest pri neočekivano dugom stanju uređaja bez ažuriranja promjene. Postoje dvije vrste mjerača vremena: aktivni mjerači (engl. *in-progress timers*) te mjerači koraka (engl. *step timers*). Aktivni mjerači vremena ne mogu se izmijeniti nakon njihova pokretanja, te služe za mjerenje izvođenja trenutno aktivnih poslova i postavljanje statusa izvršenja posla. Isto tako, ovaj mjerač vremena vrijedi za sva izvršenja istog posla, bez obzira na uređaj. Ako je potrebno pratiti odnosno ažurirati samo jedno izvršenje posla, onda se koristi mjerač koraka. On nema utjecaja na aktivni mjerač vremena. Moguće je postaviti novu vrijednost mjerača koraka pri svakom ažuriranju izvršenja posla, odnosno mjeriti trajanje svakog koraka pri izvršenju posla.

Isto tako, moguće je pokrenuti ponovni pokušaj izvršenja posla kada posao ne uspije ili pak istekne maksimalno vrijeme izvršavanja posla. Moguće je imati najviše deset ponovnih pokušaja za izvršenje posla te se svaka iteracija može nadzirati i pratiti napredak izvršenja.

Dijagram na slici 4.5 prikazuje stanja posla koja se mijenjaju tokom pokušaja izvršenja posla na uređaju. Jedan posao istovremeno ima više izvršenja poslova na različitim uređajima, te se ovisno o uspješnom ili neuspješnom izvršenju svih poslova stanje samog posla mijenja. Posao se može naći u sljedećim stanjima:

- *SCHEDULED*: ovo stanje odnosi se isključivo na poslove koji se kontinuirano izvršavaju. Kada se pokrene posao koji ima specificirano vrijeme pokretanja i završetka, status posla ažurira se u *SCHEDULED*. U trenutku početka vremenskog okvira održavanja, stanje posla promijenit će se u *IN\_PROGRESS*.
- *IN\_PROGRESS*: tokom ovog stanja, posao se postupno šalje na sve ciljne uređaje u grupi. Po završetku vremenskog okvira održavanja kontinuiranog posla, posao se vraća iz ovog stanja natrag u stanje *SCHEDULED*.

- *COMPLETED*: prijelazom u ovo stanje označava se kraj posla na ciljnom uređaju. Kontinuirani posao koji nema definirano početno i krajnje vrijeme izvršavanja nikad ne doseže ovo stanje, nego iz aktivnog stanja odnosno *IN\_PROGRESS* prelazi u stanje *SCHEDULED* gdje ponovno čeka sljedeću iteraciju izvršavanja. Kontinuirani poslovi s definiranim krajnjim vremenom po isteku tog vremena prelaze u stanje *COMPLETED*. Kod jednokratnih poslova, posao prelazi u *COMPLETED* kada sva izvršenja posla na svim uređajima dosegnu prekidno stanje.
- *CANCELED*: posao prelazi u ovo stanje namjernim otkazivanjem posla. Tijekom otkazivanja, AWS počinje poništavati izvršenja prethodno kreiranih poslova.
- *DELETION\_IN\_PROGRESS*: posao prelazi u ovo stanje pokretanjem brisanja iz konzole platforme. Pri brisanju posla, usluga briše sva ranije kreirana izvršavanja tog posla. Brisanjem se posao u potpunosti uklanja s platforme.



**Slika 4.5:** Različita stanja tijekom izvršavanja posla [3]

Postoje određena ograničenja na poslove i njihovo izvršavanje. Svi poslovi u stanju *IN\_PROGRESS* smatraju se aktivnim poslovima za koje postoji limit. Ovo uključuje poslove koji ili uvode nova izvršavanja poslova ili poslove koji čekaju da uređaji dovrše postojeće izvršavanje. Ograničenje se odnosi na kontinuirane i jednokratne poslove. Poslovi u tijeku i poslovi koji poništavaju izvršenja prethodno stvorenih poslova istovremeni su i ubraju se u ograničenje istovremenosti poslova. Usluga može pokretati i otkazivati izvršenje poslova brzinom od tisuću uređaja u minuti. Svaki posao je istovremen i ubraja se u ograničenje istovremenosti poslova samo kratko vrijeme. Nakon što su izvršenja uvedena ili otkazana, posao više nije istovremen i ne ubraja se u ograničenje istovremenosti poslova. Moguće je iskoristiti istovremenost za kreiranje

većeg broja poslova za vrijeme čekanja izvršenja postojećih poslova.

Isto tako, postoje različita stanja izvršenja samog posla. Izvršenje posla odnosi se na jednu instancu pokretanja posla na određenom ciljnom uređaju, te skup svih izvršenja poslova na svim uređajima određuje stanje samog posla. Tablica 4.2 prikazuje sva stanja u kojima se može naći izvršenje nekog posla. Isto tako, u tablici je navedeno koje stanje može okinuti uređaj, a koje usluga. Označeno je i koja su stanja prekidna, odnosno označavaju završetak izvođenja posla. Stupac *retry* odnosi se na mogućnost ponovnog pokušaja izvršenja posla nakon navedenog stanja.

**Tablica 4.2:** Stanja izvršenja posla [3]

Stanje	Pokrenuo uređaj?	Pokrenula usluga?	Prekidno?	Retry?
<i>QUEUED</i>	Ne	Da	Ne	-
<i>IN_PROGRESS</i>	Da	Ne	Ne	-
<i>SUCCEEDED</i>	Da	Ne	Da	-
<i>FAILED</i>	Da	Ne	Da	Da
<i>TIMED_OUT</i>	Ne	Da	Da	Da
<i>REJECTED</i>	Da	Ne	Da	Ne
<i>MOVED</i>	Ne	Da	Da	Ne
<i>CANCELED</i>	Ne	Da	Da	Ne

Kad usluga pošalje posao na ciljni uređaj, status izvršenja posla prelazi u *QUEUED*. Posao stoji u stanju čekanja u redu sve dok uređaj ne primi izvršenje posla, pokrene ga i promijeni status u *IN\_PROGRESS* koji zatim i pošalje na platformu. Isto tako, u slučaju da se posao otkaže ili se kriteriji otkazivanja posla ispune, status izvršenja posla prelazi u *CANCELED*. Pri uspješnom izvršenju posla, uređaj šalje obavijest na platformu o uspješnosti, čime se status mijenja u *SUCCEEDED*. Ako pak izvršenje posla bude neuspješno, prelazi u stanje *FAILED* odakle ima priliku ponovnog izvršenja ako je tako naznačeno pri konfiguraciji samog posla. Stanje *TIMED\_OUT* odnosi se na istek maksimalnog vremena izvođenja, i isto tako podržava mehanizam ponovnog izvršenja. Uređaj može promijeniti stanje izvršenja posla u *REJECTED* ako primi nevaljan ili nekompatibilan zahtjev od platforme. Stanje *MOVED* postavlja se ako uređaj više nije podoban ili valjan za izvršenje traženog posla.

Komunikacija između uređaja i platforme može se odvijati putem sučelja REST, odnosno protokolom HTTP, te protokolom MQTT. Preporučena je komunikacija pomoću protokola MQTT zbog ranije opisanih prednosti u brzini i jednostavnosti povezivanja. U tom slučaju, uređaj se mora pretplatiti na rezervirane teme kako bi primio

obavijest o nadolazećem izvršenju posla. Glavni je uvjet da razvojni sustav zna ime stvari u platformi kako bi mogao pratiti obavijesti namijenjene sebi. U nastavku su navedene teme na koje se uređaj mora preplatiti kako bi dobivao obavijesti o poslovima na čekanju. Tema `notify` označava da postoji jedan ili više poslova na čekanju, dok tema `notify-next` pruža uvid u detalje sljedećeg posla koji je potrebno izvršiti.

```
$aws/things/{thingName}/jobs/notify  
$aws/things/{thingName}/jobs/notify-next
```

**Isječak koda 4.6:** Teme za obavijesti o poslovima

Isto tako, postoje teme na koje uređaj objavljuje kako bi obavijestio AWS o stanju izvođenja traženih poslova. Teme su navedene u nastavku.

```
$aws/things/{thingName}/jobs/{ jobId}/get/accepted  
$aws/things/{thingName}/jobs/{ jobId}/update  
$aws/things/{thingName}/jobs/{ jobId}/update/accepted  
$aws/things/{thingName}/jobs/{ jobId}/update/rejected
```

**Isječak koda 4.7:** Teme za obavijesti o izvođenju poslova

#### 4.1.4. AWS IoT OTA

Usluga AWS IoT OTA (engl. *Over-The-Air*) usko je vezana za prethodno opisanu uslugu izvršavanja poslova. Ona omogućava bežično ažuriranje softvera na uređajima, čime se osigurava da uređaji rade na najnovijoj i najsigurnijoj inačici programskog koda. Usluga pruža visoku fleksibilnost i kontrolu nad procesom ažuriranja. Mogu se postaviti različite strategije ažuriranja, primjerice fazna ažuriranja, gdje se prvo ažurira manji dio uređaja kako bi se testirala stabilnost novog softvera prije nego se pošalje na sve uređaje. To omogućava brzo otkrivanje i ispravljanje potencijalnih problema bez utjecaja na sve uređaje odjednom. Osim toga, AWS pruža detaljno nadziranje i praćenje uspješnosti svakog OTA zadatka, što dodatno pomaže u upravljanju i dijagnostici. Korištenje ove usluge primarno se oslanja na uređaje koji podržavaju FreeRTOS.

Iako su OTA ažuriranja namijenjena za ažuriranje softvera uređaja, mogu se koristiti za slanje bilo kojih datoteka na uređaje registrirane u sustav AWS. Pri bežičnom slanju ažuriranja, preporučljivo je digitalno potpisati datoteke kako bi uređaji koji primaju datoteke mogli potvrditi da nisu mijenjane na putu. U tu svrhu može se koristiti usluga potpisivanja koda koju nudi AWS ili vlastite alate za potpisivanje koda.

AWS nudi i biblioteku za izvršavanje OTA ažuriranja u sklopu projekta FreeRTOS. Klijentska biblioteka nudi povezivanje putem protokola MQTT i HTTP te logički odvaja operacije ažuriranja softvera od ostatka aplikacije na uređaju. U sklopu biblioteke razvijen je OTA agent koji služi za pojednostavljenje količine koda koju je potrebno napisati za funkcionalnost OTA ažuriranja. Integracija se prvenstveno sastoji od inicijalizacije OTA agenta i stvaranja prilagođene funkcije povratnog poziva za odgovaranje na poruke događaja OTA agenta. Tijekom inicijalizacije agentu se proslijeđuju sučelja korištena za komunikaciju s platformom kao i operacijskim sustavom. Međuspremniči (engl. *buffers*) se također mogu inicijalizirati i proslijediti OTA agentu. Agent koristi protokol MQTT za sve kontrolne komunikacijske operacije koje uključuju AWS IoT usluge, ali ne upravlja MQTT vezom. Kako bi se osiguralo da OTA agent ne ometa politiku upravljanja vezom aplikacije, MQTT vezom mora upravljati glavna korisnička aplikacija. Datoteka se može preuzeti preko protokola MQTT ili HTTP. Protokol se može odabrati i pri samom kreiranju OTA posla. U slučaju odbira protokola MQTT, agent koristi istu vezu za kontrolne operacije i za preuzimanje datoteka.

Pri kreiranju OTA ažuriranja, usluga za upravljanje OTA ažuriranjima (engl. *OTA Update Manager*) stvara AWS posao kako bi obavijestila uređaje da je ažuriranje dostupno. Aplikacija na uređaju zatim stvara FreeRTOS zadatak koji se pretplaćuje na teme obavijesti za poslove i čeka poruke ažuriranja. Kada je dostupno ažuriranje, OTA agent objavljuje zahtjev za njim i prihvata ažuriranja putem protokola MQTT ili HTTP. OTA agent provjerava digitalni potpis primljene datoteke te, ako je potpis valjan, instalira ažuriranje. Cijeli je proces u suštini izvršavanje posla u kombinaciji s dohvatom i verifikacijom datoteke koju je zatim potrebno pokrenuti na uređaju radi ažuriranja. OTA ažuriranje je podatkovna struktura koju održava usluga za upravljanje ažuriranja. Objekt se sastoji od sljedećih komponenti:

- identifikator OTA ažuriranja,
- opis ažuriranja,
- popis ciljnih uređaja koje je potrebno ažurirati,
- vrsta OTA ažuriranja: kontinuirani ili jednokratni posao,
- protokol za izvođenje ažuriranja: HTTP ili MQTT,
- popis datoteka za slanje na uređaje,
- uloga (engl. *role*) koja dozvoljava pristup pohrani, poslu i usluzi potpisa koda.

Datoteka koja će se izvršiti na uređaju ključna je komponenta ažuriranja i ona mora

biti pohranjena na platformi kako bi se pri svakom izvršavanju posla dohvatala i poslala na ciljne uređaje. Pohrana datoteka omogućena je uslugom Amazon S3 pohrane (engl. *Amazon Simple Storage Service Bucket*). To je osnovna komponenta platforme AWS za skladištenje objekata i datoteka. Spremniči (engl. *buckets*) omogućavaju korisnicima pohranu i preuzimanje velike količine podataka putem skalabilnog i visoko dostupnog sučelja. Pohranjeni objekti mogu biti bilo koje vrste, uključujući slike, videozapise, dokumente, baze podataka i pričuvne kopije. Svaki objekt čuva se u spremniku s jedinstvenim ključem. Podaci u spremnicima automatski su replicirani preko više uređaja i objekata unutar jedne AWS regije, čime se osigurava otpornost na greške i visoka dostupnost podataka. Također, usluga podržava različite mehanizme za kontrolu pristupa i politike, čime se precizno definira pristup podacima. Isto tako, S3 podržava verzioniranje, što omogućava korisnicima da čuvaju više verzija istog objekta unutar jednog spremnika. Stoga, kako bi se datoteka ažuriranja mogla poslati putem posla na uređaj, potrebno je kreirati novi spremnik u S3 i unutra pohraniti željenu binarnu datoteku. Isto tako, radi potvrde o valjanosti datoteke potrebno je ili pri pohrani digitalno potpisati datoteku ili pak pri kreiranju posla ažuriranja definirati parametre potpisivanja kojima će se dokument potpisati, poput kriptografskog algoritma i potpisa u formatu PEM (engl. *Privacy-Enhanced Mail*). Prva je opcija preporučena.

#### 4.1.5. Ostale dostupne IoT usluge u sustavu AWS

Uz ranije opisanu glavnu komponentu IoT Core koju nudi AWS za stvaranje IoT aplikacija, u samom ekosustavu nalazi se još mogućnosti za jednostavniju integraciju oblaka i fizičkih uređaja. U nastavku su ukratko opisane ostale usluge koje se mogu integrirati uz jezgrenu uslugu AWS IoT Core.

Važno je napomenuti kako nisu sve usluge dostupne u svim regijama unutar platforme AWS.

##### IoT Analytics

AWS IoT Analytics automatizira korake potrebne za analizu podataka prikupljenih od IoT uređajima. Filtrira, transformira i obogaćuje podatke prije nego ih pohrani u vremensku bazu podataka za daljnju analizu. Moguće je postaviti uslugu da prikuplja podatke s uređaja samo koji su potrebni, vrši matematičke operacije i dopunjava podatke raznim metapodacima, primjerice o lokaciji. Zatim se podaci mogu analizirati koristeći ugrađeni sustav za pretraživanje koji koristi SQL sintaksu ili pak vršiti kompleksniju analizu koristeći usluge umjetne inteligencije. Isto tako, ova usluga nudi

vizualizaciju podataka integracijom s dodatnom uslugom Amazon QuickSight.

### **IoT Device Defender**

AWS IoT Device Defender potpuna je usluga koja pomaže pri osiguranju IoT uređaja. Kontinuirano revidira IoT konfiguracije radi provjere jesu li sve u skladu s najboljim sigurnosnim praksama. Također pruža kontinuirano monitoriranje sigurnosnih metrika s uređaja i usluge AWS IoT Core kako bi se detektirale anomalije u ponašanju pojedinih uređaja.

Ova usluga također omogućuje slanje alarma na konzolu AWS IoT sustava i na uslugu za monitoriranje Amazon CloudWatch. Koriste se ugrađene mitigacijske akcije kako bi se izolirali nesigurni uređaji.

### **IoT Events**

AWS IoT Events usluga služi za praćenje događaja u sustavu. Ova usluga prati ulazne podatke s više IoT uređaja i aplikacija radi prepoznavanja uzoraka i pokretanja prikladnih operacija na određene događaje. Moguće je pratiti ne samo fizičke uređaje, nego i druge AWS aplikacije integrirane u IoT sustav.

### **IoT FleetWise**

AWS IoT FleetWise jest usluga koja se koristi za prikupljanje podataka od vozila i njihovu organizaciju u oblaku. Prikupljeni se podaci mogu koristiti za poboljšanje kvalitete, performansa i autonomije vozila. Također podržava više različitih protokola i podatkovnih formata. Ova usluga pomaže pri transformaciji poruka niske razine (engl. *low-level*) u oblik čitljiv čovjeku i standardizira podatke radi lakše analize u oblaku. Moguće je također definirati vrstu podataka i trenutak u kojem se ti podaci šalju u oblak.

Kada su podaci o vozilu u oblaku, mogu se koristiti u aplikacijama koje analiziraju zdravlje vozila. Ove informacije mogu pomoći pri identifikaciji potencijalnih problema u održavanju i pri unapređenju naprednih tehnologija poput autonomne i asistirane vožnje integracijom strojnog učenja.

### **IoT Greengrass**

AWS IoT Greengrass jest usluga otvorenog koda (engl. *open source*) za računarstvo na rubu (engl. *edge computing*) i u oblaku koja pomaže pri izradi, objavi i upravljanju

IoT aplikacija na uređajima. Može se koristiti za omogućavanje uređajima lokalno reagiranje na podatke koje generiraju, pokretanje modela strojnog učenja za predikciju, te filtriranje i agregaciju podataka s uređaja. Omogućava uređajima da prikupljaju i analiziraju podatke ne u oblaku, nego ili na samom uređaju ili drugom mjestu koje je bliže izvorištu tih podataka. Također može komunicirati na siguran način s uslugom AWS IoT Core i izvoziti podatke u oblak. Karakteristika računarstva u rubu, koje omogućava ova komponenta, jest približavanje računanja izvorišnim uređajima, čime se poboljšava vrijeme odziva i štedi propusnost [5].

### **IoT Roborunner**

AWS IoT RoboRunner nova je usluga koja pruža infrastrukturu za optimizaciju robota iz jedne točke gledišta. Uz pomoć ove usluge moguće je izgraditi aplikacije za jednostavniji međusobni rad robota. Namijenjena je za industrijske robote i automatizirane sustave za olakšano upravljanje opremom. Pruža centralne rezervorije podataka za pohranu te podržava različite podatkovne formate od raznih robota i autonomnih sustava.

### **IoT TwinMaker**

AWS IoT TwinMaker usluga je za kreiranje operativnih digitalnih dvojnika fizičkih i digitalnih sustava. Stvara digitalne vizualizacije koristeći mjerenja i analize iz raznih senzora i kamera radi praćenja stvarnog stanja i uvjeta u kojima se objekt, zgrada ili kompleks nalazi. Podaci iz stvarnog svijeta se mogu koristiti za dijagnostiku i ispravljanje pogrešaka ili pak optimizaciju operacija.

Digitalni dvojnik (engl. *digital twin*) digitalna je reprezentacija sustava i svih njegovih fizičkih i digitalnih komponenti. Dinamički se ažurira primitkom novih podataka kako bi simulirao stvarno stanje i ponašanje sustava.

### **IoT SiteWise**

AWS IoT SiteWise jest usluga koja skalabilno prikuplja, modelira, analizira i vizualizira podatke iz industrijske opreme. Usluga pruža kreiranje web aplikacija za operativne korisnike radi prikaza i analize industrijskih podataka u stvarnom vremenu. Moguće je dobiti uvide u podatke i operacije konfiguriranjem i praćenjem raznih metrika, primjerice efektivnost i efikasnost opreme. Ovu je uslugu moguće koristiti jedino uz ranije opisan IoT TwinMaker.

# 5. Povezivanje razvojnog sustava i oblaka

Oblak koju pruža platforma AWS i razvojni sustav ESP32-C3 dva su odvojena sustava koja moraju međusobno komunicirati i razmjenjivati podatke. Za ostvarenje njihove veze razvijena su dva programska rješenja:

1. programska potpora za mikrokontroler, koja će omogućiti dinamičko povezivanje na Wi-Fi, spajanje na platformu AWS te slanje podataka u oblak,
2. programska potpora za platformu AWS, koja će ostvariti umrežavanje uređaja u sustav, ažuriranje softvera na uređaju, pohranu primljenih podataka s uređaja te prikaz tih podataka u web aplikaciji.

## 5.1. Programska potpora za mikrokontroler

Za razvoj programske potpore za mikrokontroler korišten je paket za razvoj softvera *ESP-AWS-IoT* (engl. *Software Development Kit - SDK*) tvrtke *Espressif*. To je repozitorij otvorenog koda temeljen na službenom programskom paketu tvrtke Amazon koji omogućava ugradbenim računalima razvijanih u programskom jeziku C komunikaciju s AWS-om. Korišteni razvojni paket omogućava uređajima temeljnim na ESP32 jezgri povezivanje sa uslugama platforme AWS. Pojednostavljuje integraciju uređaja s AWS ekosustavom nudeći gotova sučelja i programske primjere za sve značajke [10]. Razvijena programska potpora za uređaj ESP32-C3 sastoji se od nekoliko komponenti:

- dinamičko povezivanje na bežičnu mrežu,
- spajanje na platformu AWS,
- učitavanje novog softvera,
- očitavanje senzorskih mjerena,
- slanje podataka u oblak protokolom MQTT.

Neke od navedenih komponenti izvršavaju se slijedno, dok se druge izvršavaju paralelno. Spajanje na Wi-Fi i povezivanje s platformom AWS ključni su koraci koji prethode bilo kakvom pokušaju slanja podataka u oblak. Isto tako, praćenje ažuriranja softvera i očitavanje mjerena izvršavaju se paralelno u posebnim procesima budući da nisu sekvencijalni niti međusobno isključivi zadaci. U nastavku je pobliže opisan svaki navedeni segment programske potpore.

### 5.1.1. Dinamičko povezivanje mikrokontrolera na Wi-Fi

Radni okvir ESP-IDF nudi dinamičko spajanje na Wi-Fi mrežu pomoću zasebne komponente. Ovaj se postupak naziva provizioniranje (engl. *provisioning*). Ova komponenta pruža aplikacijska programska sučelja (engl. *Application Programming Interface - API*) koja kontroliraju pružanje usluge za primanje i konfiguriranje Wi-Fi vjerodajnica putem sigurnih komunikacijskih protokola. Sigurnosni protokoli definirani su u komponenti protokolne komunikacije (engl. *protocomm*) koja upravlja sigurnim sjednicama (engl. *sessions*) i pruža radni okvir za višestruki prijenos podataka. Također je moguće direktno koristiti sloj protokolne komunikacije radi implementacije specifične za aplikaciju [11].

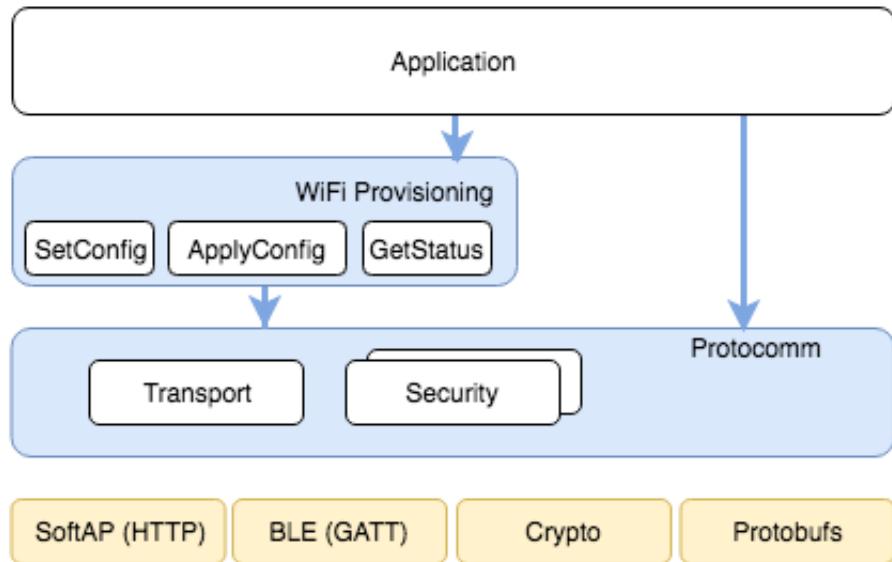
Sloj protokolne komunikacije interno koristi mehanizam protokolnih međuspremnika (engl. *protocol buffers - protobuf*) za sigurno uspostavljanje sjednice. Protokolni međuspremnici namijenjeni su za serijalizaciju strukturiranih podataka neovisno o programskom jeziku i platformi. Koristan je pri izradi programa i sustava koji međusobno komuniciraju putem mreže zbog kompaktnosti i niske latencije [16].

Sloj protokolne komunikacije pruža radni okvir za različite načine komunikacije:

1. protokol BLE,
2. Wi-Fi (SoftAP u kombinaciji s HTTP serverom).

Pružajući korisnicima okvir za ostvarivanje usluge dinamičkog povezivanja u mrežu, neovisno o načinu komunikacije, ovakva vrsta podrške naziva se unificirano provizioniranje (engl. *unified provisioning*). Ovakav način prijave uređaja na mrežu zahtjeva interakciju korisnika putem vanjskog uređaja za slanje vjerodajnica na mikrokontroler. Tvrtka *Espressif* pruža jednostavna mobilna rješenja koja se mogu koristiti gotova ili pak uklopiti u vlastitu mobilnu aplikaciju. Na slici 5.1 prikazana je arhitektura usluge.

Kao što je ranije opisano, arhitektura je bazirana na sloju protokolne komunikacije koji je odgovoran za prijenos podataka i sigurnost. Služi za jednostavne povratne



**Slika 5.1:** Arhitektura unificiranog provoziranja [11]

pozive aplikaciji (engl. *callbacks*) i dobivanje Wi-Fi statusa. Sama aplikacija ima kontrolu nad implementacijom povratnih poziva.

Aplikacija stvara instancu protokolne komunikacije koja se preslikava na određeni prijenosni protokol i sigurnosnu shemu. Svaki prijenos podataka u sloju protokolne komunikacije ima koncept krajne točke (engl. *endpoint*) koji odgovara logičkom komunikacijskom kanalu za određenu vrstu informacija. Primjerice, sigurnosno rukovanje (engl. *handshake*) odvija se na različitoj krajnjoj točki u odnosu na točku za Wi-Fi konfiguraciju. Svaka se krajnja točka identificira nizom znakova i mijenja se ovisno o internom prikazu krajne točke. U slučaju prijenosa pomoću Wi-Fi veze odnosno SoftAP funkcionalnosti, krajnja točka prikazuje se kao URI, dok u slučaju prijenosa podataka putem protokola BLE odgovara GATT karakteristici sa specifičnim identifikatorom.

Oglašavanje i otkrivanje uređaja prepušteno je aplikaciji i ovisno o odabranom protokolu, vanjske aplikacije mogu odabrati odgovarajuću metodu za oglašavanje i otkrivanje. Za Wi-Fi prijenos obično se koristi ime mreže pristupne točke. Za prijenos putem protokola BLE može se koristiti ime samog uređaja.

Kao što je opisano, podržano je korištenje protokola BLE kao i Wi-Fi usluge za prijenos vjerodajnica. Pri odabiru prijenosnog kanala za spajanje uređaja u mrežu, potrebno je razmotriti nekoliko točaka. Za početak, prijenos temeljen na protokolu BLE prednost održavanja netaknutog komunikacijskog kanala između uređaja i klijenta tijekom prijenosa podataka, što osigurava pouzdanu povratnu informaciju. S druge strane,

prijenos putem Bluetootha troši oko 110 KB memorije tijekom rada, što je na uređaju niskih resursa velika potrošnja. Korisno je što se korištena memorija može vratiti na hrpu (engl. *heap*) po završetku umrežavanja uređaja ukoliko se BLE funkcionalnosti više ne koriste. Prijenos temeljen na Wi-Fi mreži, odnosno SoftAP funkcionalnosti, vrlo je interoperabilan i ne troši dodatnu memoriju. Međutim, mikrokontroler koristi isti radio za emitiranje pristupne točke i za spajanje na željenu mrežu. Budući da se te akcije mogu odvijati na različitim kanalima, postoji mogućnost da se ažuriranja statusa veze ne dostave na mobilni uređaj. Također, mobilni se uređaj mora odspojiti s izvorne Wi-Fi mreže radi privremenog spajanja na pristupnu točku mikrokontrolera. Uređaj će se spojiti na izvornu mrežu tak kada mikrokontroler ugasi pristupnu točku [11].

Za razvoj predloženog rješenja korišteno je slanje vjerodajnica pomoću Wi-Fi mreže, odnosno privremene pristupne točke. Kao što je ranije opisano, protokol BLE troši značajnu količinu *heap* memorije, a razvojni sustav ESP32-C3 nema dovoljno radne memorije koja bi pokrila prijavu u mrežu uz ostale radne procese. Mikrokontroler najprije stvori privremenu pristupnu točku na koju se mobilni uređaj spaja pomoću mobilne aplikacije. Zatim, nakon skeniranja dostupnih Wi-Fi mreža u blizini, u mobilnoj aplikaciji odabire se željena mreža i unese lozinka. Vjerodajnice se zatim pošalju putem Wi-Fi mreže, i mobilni uređaj može se odspojiti s privremene pristupne točke. Vjerodajnice se pohrane u memoriju tipa NVS (engl. *non-volatile storage*) koja ne zahtijeva konstantno napajanje kako bi se zadržala na uređaju. Ovime je omogućeno povezivanje uređaja u sustav čak i kada dođe do prekida napajanja [26]. Memorija tipa NVS može se jedino programski obrisati, te bi u idealnom izvedbenom rješenju postojao vanjski gumb spojen na mikrokontroler koji bi pokretao brisanje te memorije i tako omogućio ponovno spajanje na željenu mrežu. Sljedeći programski isječak prikazuje inicijalizaciju memorije NVS, mrežnog sučelja te stvaranje pristupne točke.

```
/* Init NVS partition */
esp_err_t ret = nvs_flash_init();
/* Init TCP/IP */
ESP_ERROR_CHECK(esp_netif_init());
/* Init the event loop */
ESP_ERROR_CHECK(esp_event_loop_create_default());
 wifi_event_group = xEventGroupCreate();
/* Init Wi-Fi including netif with default config */
esp_netif_create_default_wifi_sta();
esp_netif_create_default_wifi_ap();
```

```

wifi_prov_mgr_config_t config = {
    .scheme = wifi_prov_scheme_softap,
    .scheme_event_handler = WIFI_PROV_EVENT_HANDLER_NONE
};

/* Init provisioning manager with above config */
ESP_ERROR_CHECK(wifi_prov_mgr_init(config));
ESP_ERROR_CHECK(wifi_prov_mgr_start_provisioning(
    security, (const void *) sec_params, service_name,
    service_key));
wifi_prov_print_qr(service_name, username, pop,
PROV_TRANSPORT_SOFTAP, disp);

```

**Isječak koda 5.1:** Stvaranje pristupne točke

## LCD zaslon

Za povezivanje mobilnog uređaja na privremenu pristupnu točku koju emitira razvojni sustav, potrebno je skenirati QR kod koji mikrokontroler generira. Budući da ESP32-C3 nema vlastito sučelje, na sustav je spojen zaslon OLED SSD1306 veličine 128×64 piksela. Uređaj sa zaslonom komunicira putem I2C sučelja, a za prikaz sadržaja na zaslonu korištena je biblioteka LVGL (engl. *Light and Versatile Graphics Library*). To je grafička biblioteka otvorenog koda namijenjena izradi aplikacija s grafičkim korisničkim sučeljem (engl. *Graphical User Interface - GUI*) za ugradbene sustave. Pruža radni okvir s mnogim značajkama, temama i paletama boja. Isto tako, biblioteka troši vrlo malo resursa, što je čini pogodnom za uređaje poput razvojnog sustava ESP32-C3 [2]. Generiranje QR koda obavlja se pomoću biblioteke *QR-Code-Generator* koja je prilagođena ESP32 uređajima. Tablica 5.1 prikazuje način spajanja pločice sa zaslonom. Kao što se vidi iz konfiguracije, osim napajanja i uzemljenja, potrebno je spojiti liniju za prijenos podataka te liniju takta.

**Tablica 5.1:** Povezivanje uređaja i LCD zaslona

Pin razvojnog sustava	Pin zaslona
GND	GND
5V	Vcc
GPIO 4	SCL
GPIO 5	SDA

```

static void wifi_prov_print_qr(const char *name, const
    char *username, const char *pop, const char *transport,
    lv_disp_t *disp) {
    char payload[150] = {0};
    snprintf(payload, sizeof(payload),
        "{\"ver\": \"%s\", \"name\": \"%s\", \"username\": \"%s\",
        \"pop\": \"%s\", \"transport\": \"%s\"}",
        PROV_QR_VERSION, name, username, pop, transport);
    esp_qrcode_config_t cfg = {
        .display_func = generate_qr_code_lcd,
        .max_qrcode_version = 10,
        .qrcode_ecc_level = ESP_QRCODE_ECC_LOW
    };
    esp_qrcode_generate(&cfg, payload);
}

```

### Isječak koda 5.2: Generiranje QR koda iz pristupne točke

Prethodna funkcija povezuje pristupnu točku s QR kodom. Podaci o samoj pristupnoj točki učitaju se u privremenu varijablu, čiji se sadržaj prosleđuje biblioteci za generiranje QR koda. Dobiveni se podaci zatim prosleđuju funkciji za prikaz koda na zaslonu. QR kod prikazuje se na zaslonu piksel po piksel, skalirajući veličinu QR koda na temelju širine i duljine samog zaslona.

```

void generate_qr_code_lcd(esp_qrcode_handle_t qrcode)
{
    ESP_LOGI(TAG, "%s", "Started generate_qr_code_lcd...");

    int size = qrcodegen_getSize(qrcode);

    // Calculate the scale factor
    int scale = (int)fmin(EXAMPLE_LCD_H_RES / size,
        EXAMPLE_LCD_V_RES / size);

    // Calculate horizontal shift
    int shift_x = (EXAMPLE_LCD_H_RES - size * scale)/2;

```

```

// Calculate vertical shift
int shift_y = (EXAMPLE_LCD_V_RES - size * scale)/2;

if (lvgl_port_lock(0)) {
    lv_obj_t *screen = lv_scr_act();
    lv_obj_clean(screen); // Clear the screen to ensure
    it's dark

    // Create a canvas object
    lv_obj_t *canvas = lv_canvas_create(screen);
    static lv_color_t cbuf[LV_CANVAS_BUF_SIZE_TRUE_COLOR(
        EXAMPLE_LCD_H_RES, EXAMPLE_LCD_V_RES)];
    lv_canvas_set_buffer(canvas, cbuf, EXAMPLE_LCD_H_RES,
        EXAMPLE_LCD_V_RES, LV_IMG_CF_TRUE_COLOR);
    lv_canvas_fill_bg(canvas, lv_color_white(),
        LV_OPA_COVER);

    // Draw the QR code on the canvas
    for (uint8_t y = 0; y < size; y++) {
        for (uint8_t x = 0; x < size; x++) {
            if (qrcodegen_getModule(qrcode, x, y)) {
                for (int dy = 0; dy < scale; dy++) {
                    for (int dx = 0; dx < scale; dx++) {
                        lv_canvas_set_px(canvas, shift_x + x *
                            scale + dx, shift_y + y * scale + dy, lv_color_black());
                    }
                }
            }
        }
    }

    // Release the mutex
    lvgl_port_unlock();
}
}

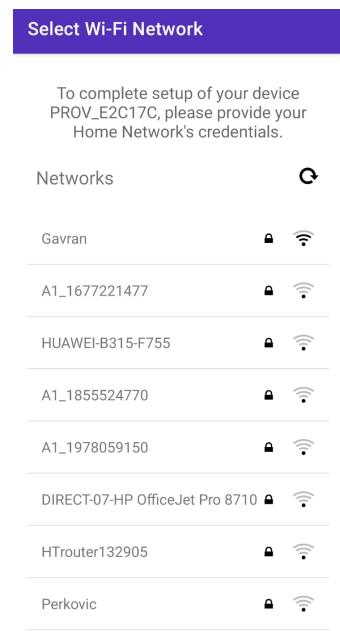
```

### Isječak koda 5.3: Funkcija za prikaz QR koda na zaslonu

Na slikama 5.2 i 5.3 prikazana je mobilna aplikacija te trenutak nakon skeniranja QR koda. Mobilni uređaj zahtijeva spajanje na privremenu pristupnu točku, a iduća slika prikazuje dostupne Wi-Fi mreže u blizini mobilnog uređaja, ujedno i mikrokontrolera, na koje se razvojni sustav može spojiti. Odabirom jedne od mreža i unosom lozinke vjerodajnice se šalju na razvojni sustav.



Slika 5.2: Obavijest nakon skeniranja QR koda



Slika 5.3: Odabir dostupne Wi-Fi mreže u blizini

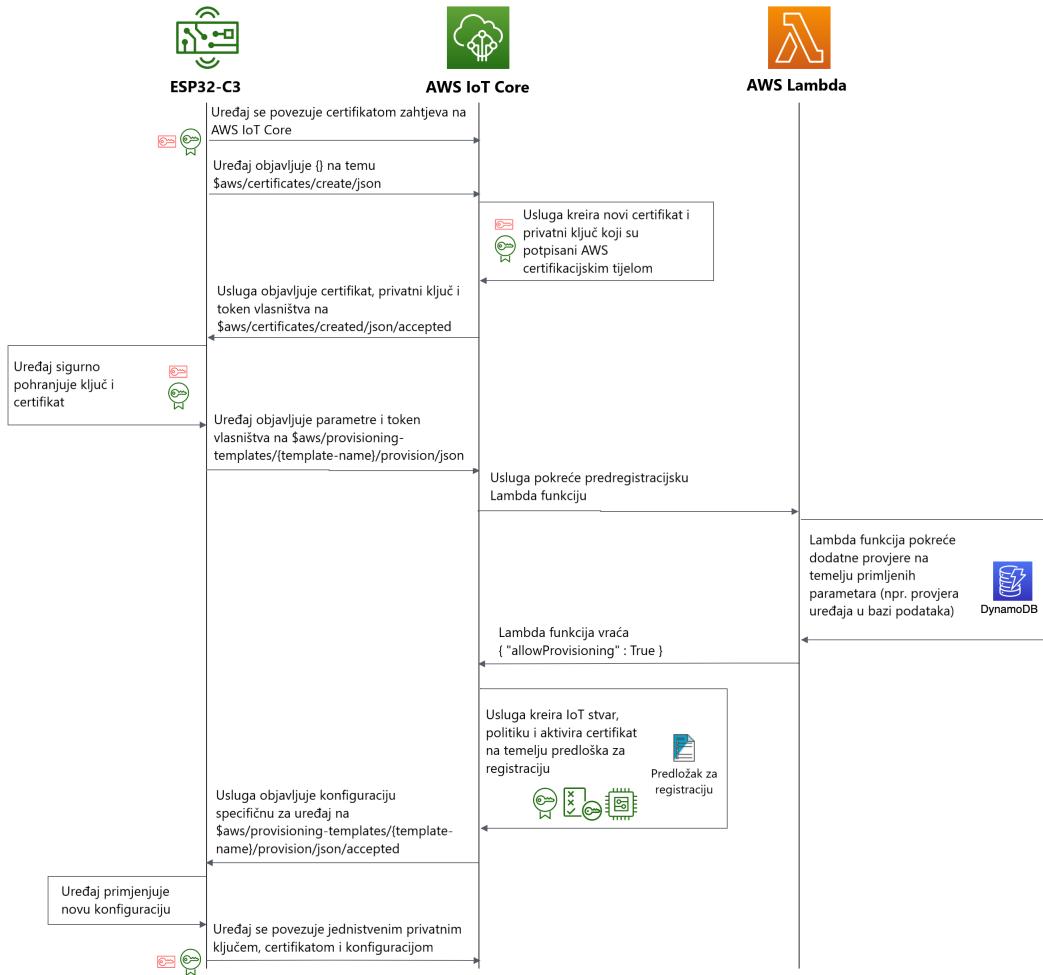
## 5.1.2. Registracija u sustav AWS

Nakon uspješnog povezivanja na Wi-Fi, sljedeći je korak registracija uređaja na platformu AWS. Korištena je biblioteka za AWS IoT Fleet Provisioning koja se održava u sklopu projekta otvorenog koda *FreeRTOS* [15]. Biblioteka omogućuje mnoštvo odn. floti IoT uređaja registraciju i instalaciju jedinstvenih certifikata na platformu. Bibliotekom je moguće uređaje registrirati i pomoću autoriziranog korisnika, ali i certifikatima zahtjeva. Ova biblioteka ne ovisi o dodatnim bibliotekama osim standardne C biblioteke i stoga se može koristiti s bilo kojom bibliotekom za MQTT protokol.

Za potrebe ovog sustava odabrana je registracija pomoću certifikata zahtjeva. Na slici 5.4 prikazan je slijed događaja pri registraciji uređaja. Uređaj se najprije spaja

na AWS certifikatom zahtjeva koji unaprijed postoji na mikrokontroleru te ostvaruje MQTT vezu. Zatim objavljuje praznu poruku na temu `\$aws/certificates/create/json`, kojom zapravo podnosi zahtjev za kreiranjem vlastitog jedinstvenog certifikata. Usluga kreira traženi certifikat, pripadni ključ te generira značku vlasništva (engl. *ownership token*), što sve skupa i objavljuje na temu `\$aws/certificates/created/json/accepted`, na koju je uređaj pretplaćen. Razvojni sustav zatim sigurno pohranjuje dobivene vjerodajnice te objavljuje vlastite parametre i dobivenu značku na temu predloška za registraciju. Ako je kreirana, u sustavu AWS zatim se pokreće Lambda funkcija koja obavlja dodatne provjere nad dobivenim parametrima i tako provjerava valjanost zahtjeva. Primjerice, uređaj pri slanju parametara šalje i hardversku tajnu, koja se može zatim provjeriti u bazi podataka sustava AWS s pohranjenim takvim tajnama i provjeriti valjanost te tajne. Pri uspješnoj provjeri, Lambda funkcija vraća `{"allowProvisioning" : True}`, čime daje konačno zeleno svjetlo za registraciju. Usluga posljedično kreira stvar i pripadnu politiku definiranu u predlošku te aktivira novostvoreni certifikat. Objavljuje novu konfiguraciju specifičnu za uređaj na temu za potvrdu uspješne registracije. Razvojni se sustav, nakon primjene nove konfiguracije, povezuje jedinstvenim privatnim ključem te certifikatom na AWS. Ova se nova MQTT veza dalje koristi za komunikaciju i razmjenu podataka. Ako je prvo povezivanje neuspješno, aplikacija pokušava spojiti uređaj još dva puta prije nego se spajanje smatra potpuno neuspjelim. U tom slučaju, daljnje akcije čitanja mjerena i slanja u oblak ne pokreću se jer je spajanje na platformu ključno za ostale radnje, stoga aplikacija izlazi iz glavnog programa. Jedino se ponovnim pokretanjem sustav može ponovno pokušati spojiti na internet i na platformu.

Opisani tok ostvaren je uz pomoć ranije spomenute biblioteke, kao i knjižice *coreMQTT* za komunikaciju putem protokola MQTT s API-jem platforme AWS. Za manipulaciju certifikatima i privatnim ključevima, korištena je biblioteka *corePKCS11*. Ona koristi standard PKCS #11 (engl. *Public-key Certificate Standards*), što je široko korišten API za manipuliranje uobičajenim kriptografskim objektima. Funkcije koje navodi omogućuju aplikacijama korištenje, stvaranje, modificiranje i brisanje kriptografskih objekata, bez izlaganja tih objekata memoriji aplikacije. Primjerice, konkretna integracija s bibliotekom za registraciju uređaja koristi mali podskup PKCS #11 API-ja za, između ostalog, pristup privatnom ključu potrebnom za stvaranje mrežne veze koja je autentificirana i zaštićena protokolom TLS bez da aplikacija ikada dođe u doticaj s ključem [19]. Certifikati zahtjeva te korijenski certifikat platforme AWS pohranjeni su u memoriju tipa SPIFFS (engl. *Serial Peripheral Interface Flash File System*). To je lagani datotečni sustav namijenjen serijskim (SPI) memorijama tipa *flash*.



Slika 5.4: Tok registracije uređaja certifikatom zahtjeva [3]

Omogućava mikrokontrolerima korištenje integrirane memorije tipa *flash* za pohranu datoteka na sličan način kao što se koriste datotečni sustavi na običnim računalima. Optimizirana je za česte operacije čitanja i pisanja. Ova vrsta memorije nalik je memoriji tipa EEPROM (engl. *Electrically Erasable Programmable Read-Only Memory*), no SPIFFS ima značajnije prednosti u odnosu na EEPROM [9]:

- nudi više prostora za pohranu,
- organizira podatke u strukturu datotečnog sustava,
- optimiziran za nasumične operacije čitanja i pisanja,
- posjeduje mehanizme za izjednačavanje trošenja koji pomažu pri ravnomjernoj rasporedi ciklusa pisanja po memoriji tipa *flash*,
- nudi mogućnost upravljanja datotekama.

Sljedeći programski kod prikazuje inicijalizaciju memorije tipa SPIFFS. Ove se funkcije pokreću neposredno prije i nakon registracije uređaja u AWS, budući da je

memorija potrebna samo za korištenje certifikata zahtjeva. Novi se certifikati direktno upisuju u objekt modula PKCS, stoga nije potrebno daljnje korištenje ovog datotečnog sustava.

```
esp_vfs_spiffs_conf_t conf = {
    .base_path = "/spiffs",
    .partition_label = "spiffs_storage",
    .max_files = 5,
    .format_if_mount_failed = true
};

void filesystem_init(void)
{
    // Use settings defined above to initialize and mount
    // SPIFFS filesystem.
    esp_err_t ret = esp_vfs_spiffs_register(&conf);

    if (ret != ESP_OK) {
        if (ret == ESP_FAIL) {
            ESP_LOGE(TAG, "Failed to mount or format filesystem");
        } else if (ret == ESP_ERR_NOT_FOUND) {
            ESP_LOGE(TAG, "Failed to find SPIFFS partition");
        } else {
            ESP_LOGE(TAG, "Failed to initialize SPIFFS (%s)",
                    esp_err_to_name(ret));
        }
        return;
    }

    size_t total = 0, used = 0;
    ret = esp_spiffs_info(conf.partition_label, &total, &used);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to get SPIFFS partition
information (%s). Formatting...", esp_err_to_name(ret))
    }
}
```

```

    );
    esp_spiffs_format(conf.partition_label);
    return;
} else {
    ESP_LOGI(TAG, "Partition size: total: %d, used: %d",
    total, used);
}
}

void filesystem_deinit(void)
{
    // All done, unmount partition and disable SPIFFS
    esp_vfs_spiffs_unregister(conf.partition_label);
    ESP_LOGI(TAG, "SPIFFS unmounted");
}

```

**Isječak koda 5.4:** Inicijalizacije memorije tipa SPIFFS

Sljedeći programski odsječak prikazuje uspostavu MQTT sjednice pomoću certifikata zahtjeva te podnošenje zahtjeva za novim certifikatom. Na samom početku najprije se certifikati s definiranom putanjom učitaju u modul PKCS za daljnje korištenje. U odsječku se također može primijetiti binarna serijalizacija podataka. Format CBOR (engl. *Concise Binary Object Representation*) Representation), za razliku od formata JSON, pretvara podatke u binarni oblik te ih tako šalje mrežom, što rezultira manjom latencijom i nosivošću (engl. *payload*) [6]. Pri prijenosu velike količine podataka pri ograničenim resursima, poput u IoT sustava, ušteda na veličini poslanih podataka znatno utječe na efikasnost sustava. Isto tako, binaran je format prilagodljiv u odnosu na JSON, gdje mora postojati unaprijed definirana shema za primitak podataka.

```

/* Insert the claim credentials into the PKCS #11 module
 */
status = loadClaimCredentials( *p11Session,
    CLAIM_CERT_PATH,
    pkcs11configLABEL_CLAIM_CERTIFICATE,
    CLAIM_PRIVATE_KEY_PATH,
    pkcs11configLABEL_CLAIM_PRIVATE_KEY );
LogInfo( ( "Establishing MQTT session with claim
certificate..." ) );

```

```

status = EstablishMqttSession(
    provisioningPublishCallback,
    *p11Session,
    pkcs11configLABEL_CLAIM_CERTIFICATE,
    pkcs11configLABEL_CLAIM_PRIVATE_KEY );
status = subscribeToCsrResponseTopics();
status = generateKeyAndCsr( *p11Session,
    pkcs11configLABEL_DEVICE_PRIVATE_KEY_FOR_TLS,
    pkcs11configLABEL_DEVICE_PUBLIC_KEY_FOR_TLS,
    csr,
    CSR_BUFFER_LENGTH,
    &csrLength );
/* Publish the CSR to CreateCertificatefromCsr API. */
PublishToTopic( FP_CBOR_CREATE_CERT_PUBLISH_TOPIC,
    FP_CBOR_CREATE_CERT_PUBLISH_LENGTH,
    ( char * ) payloadBuffer,
    payloadLength );
status = waitForResponse();

```

**Isječak koda 5.5:** Spajanje certifikatom zahtjeva i zahtjev za novim certifikatom

### 5.1.3. Očitavanje senzorskih mjerena

Razvijeni sustav, osim zaslona, sadrži dva senzora koja mjere stanja iz okoline: senzor za temperaturu i vlagu zraka te senzor za vlažnost tla. Mjerenja ovih senzora šalju se na platformu AWS i simuliraju stvarna poljoprivredna mjerenja. Mjerenje temperature i vlage zraka vrši se pomoću modula DHT11, dok se vlažnost tla mjeri senzorom VMA303. Tablice 5.2 i 5.3 prikazuju konfiguracije spajanja senzora s razvojnim sustavom.

Pin ESP32-C3	Pin DHT11
GND	GND
3.3V	Vcc
GPIO 0	S

**Tablica 5.2:** Spajanje uređaja i modula DHT11

Pin ESP32-C3	Pin VMA303
GND	GND
3.3V	Vcc
GPIO 2	S

**Tablica 5.3:** Spajanje uređaja i senzora VMA303

Glavna funkcija senzora DHT11 jest mjerjenje temperature i vlažnosti okoline. Senzor je tvornički kalibriran te može mjeriti temperaturu od 0°C do 50°C i vlažnost od 20% do 90% s točnošću od  $\pm 1^\circ\text{C}$  i  $\pm 1\%$ . Senzor uključuje komponentu za mjerjenje vlažnosti i NTC (engl. *Negative Temperature Coefficient*) komponentu za mjerjenje temperature. DHT11 može se nabaviti kao senzor ili kao modul. Senzor dolazi kao 4-pinski paket od kojeg se koriste samo tri pina, dok modul dolazi s tri pina. Razlika između modula i senzora je u tome što modul ima ugrađeni kondenzator za filtriranje i otpornik za lakše spajanje s razvojnim sustavom [23]. U razvijenom je sustavu korišten modul koji već ima integriran otpornik.

Komunikacija i sinkronizacija između mikrokontrolera i senzora odvija se jednom podatkovnom linijom. Inicijalizacija komunikacije zahtijeva interakciju mikrokontrolera povlačenjem i otpuštanjem signala, čime jedan drugom signaliziraju spremnost na prijenos podataka. Najprije mikrokontroler povlači signal na nisku razinu na najmanje 18 milisekundi, nakon čega otpušta signal i pušta ga da se vrati na visoku razinu te čeka senzor da ga spusti natrag na nisku. DHT11 zatim povlači signal na nisku razinu na otprilike 80 mikrosekundi, nakon čega otpušta signal te se on vraća ponovno na visoku razinu, i taj proces također traje oko 80 mikrosekundi. Po završetku inicijalizacije DHT11 sekvensijalno prenosi 40 podatkovnih bitova, gdje prvi i treći bajt predstavljaju vlagu zraka i temperaturu. Drugi i četvrti bajt sadrže samo nule, a peti je kontrolni zbroj (engl. *checksum*) svih ostalih bajtova na sljedeći način:

```
byte_5 == (byte_1 + byte_2 + byte_3 + byte_4) & 0xFF
```

**Isječak koda 5.6:** Kontrolni zbroj poslanih bajtova s DHT11

Sensor VMA303 mjeri vlažnost tla tako što mjeri pad napona preko dvije elektrode. Senzor u sebi već ima integriran komparator, što nije potrebno spajati dodatnu vanjsku komponentu. Osim elektroda, senzor se sastoji od kontrolnog elektronskog modula koji interpretira signale sa sonde i daje izlaz u digitalnom ili analognom obliku. Kada je sonda ubodena u tlo, metalne elektrode formiraju strujni krug. Vlažno tlo ima veću vodljivost što označava manji otpor, dok suho tlo ima manju vodljivost, time i veći otpor. Kontrolni modul šalje mali električni signal kroz jedan od pinova koji putuje kroz tlo i prima ga drugi pin. Na osnovu količine elektriciteta koji prođe kroz tlo, ovisno o vlažnosti zemlje, modul generira odgovarajući izlazni signal. U analognom načinu rada, modul daje izlazni napon proporcionalan nivoj vlažnosti tla. Viša vlažnost rezultira višim naponom na izlaznom pinu, dok niža vlažnost daje niži napon. U slučaju digitalnog izlaza, modul koristi komparator za uspoređivanje izmjerene vrijednosti s unaprijed postavljenim pragom. Ako je vlažnost ispod praga, izlazni pin daje nisku

razinu, dok u suprotnom vraća visoku razinu signala.

U razvijenom se sustavu koristi analogni način rada, te je potrebna analogno-digitalna pretvorba za analizu ulaznog signala. Pretvorba se odvija u jednokanalnom načinu rada te se koristi sukcesivna aproksimacija razlučivosti 12 bita. Sljedeći programski isječak prikazuje inicijalizaciju parametara ključnih za čitanje signala i pretvorbu. Vrijednosti *DRY\_SOIL* i *WET\_SOIL* označavaju najmanju i najveću moguću vrijednost pri mjerenu vlažnosti tla, odnosno 0% i 100%, te se stvarni postotak dobiva skaliranjem na te dvije vrijednosti i pretvaranjem u postotni oblik. Varijable su dobivene eksperimentalnim mjerjenjem sirovih vrijednosti pri vlažnoj i suhoj zemlji.

```
void adc_init(void) {
    adc1_config_width(ADC_WIDTH);
}

void moisture_init() {
    adc1_config_channel_atten(MOISTURE_CHANNEL, ADC_ATTEN);
}

float read_moisture(void) {

    int adc_value = adc1_get_raw(MOISTURE_CHANNEL);
    float moisture = 0;

    if (adc_value >= WET_SOIL) {
        moisture = 100.0;
    } else if (adc_value <= DRY_SOIL) {
        moisture = 0.0;
    } else {
        moisture = ((100.0) / ((double) DRY_SOIL - (double)
        WET_SOIL)) * ((double) (adc_value) - (double) WET_SOIL);
    }

    return moisture;
}
```

Senzorska mjerena odvijaju se u posebnom FreeRTOS zadatku koji se pokreće nakon uspješnog povezivanja na Wi-Fi i registracije na platformu. Mjerena se vrše

svakih pet sekundi i prikazuju se na zaslonu. Isti se zadatak koristi i za slanje očitanih podataka nakon čitanja i prikaza. U nastavku je prikazan opisani zadatak. Također, može se primijetiti da isti zadatak pokreće funkciju za ažuriranje sjene uređaja.

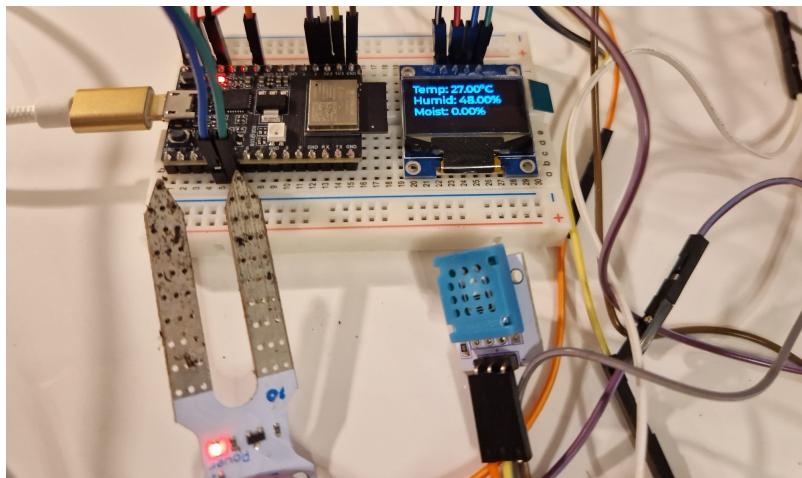
```
void publish_message_task(void *pvParameters) {
    float temperature, humidity, moisture;
    esp_err_t dht_temp;
    char display_text[512];

    while (1) {
        dht_temp = dht_read_float_data(SENSOR_TYPE, DHT_PIN,
                                         &humidity, &temperature) != ESP_OK;
        moisture = read_moisture();

        if (dht_temp == ESP_OK) {
            if (lvgl_port_lock(0)) {
                lv_obj_t *screen = lv_scr_act();
                lv_obj_clean(screen); // Clear the screen
                lv_obj_t *label = lv_label_create(screen);
                lv_label_set_long_mode(label, LV_LABEL_LONG_WRAP);
                sprintf(display_text, "Temp: %.2f C\nHumid: %.2f
%%\nMoist: %.2f%%", temperature, humidity, moisture);
                lv_label_set_text(label, display_text);
                // Release the mutex
                lvgl_port_unlock();
            }
            sendMessage(temperature, humidity, moisture);
            color_name = device_shadow_main();
        }
        vTaskDelay(pdMS_TO_TICKS(5000));
    }
}
```

**Isječak koda 5.7:** FreeRTOS zadatak za očitanje senzorskih mjerjenja i slanje podataka

Slika 5.5 prikazuje razvojni sustav ESP32-C3 spojen na periferne senzore te LCD zaslon. Isto tako, na zaslonu su vidljiva očitanja dobivena sa senzora.



**Slika 5.5:** Razvojni sustav sa spojenom periferijom

#### 5.1.4. Slanje očitanih podataka protokolom MQTT

Kao što je ranije spomenuto, slanje očitanih podataka odvija se u istom zadatku odnosno dretvi kao i očitanje mjerena, neposredno nakon dohvata vrijednosti. Slanje se odvija na temelju ranije ostvarene MQTT veze i razmijenjenim certifikatima. Podaci se šalju u formatu JSON radi jednostavnije kasnije obrade i čitljivosti pri primjeku poruka. Za kreiranje podatkovne strukture za slanje korištena je biblioteka *cJSON*. To je lagana i brza biblioteka za rukovanje podacima u obliku JSON namijenjena programskom jeziku C. Omogućava lako parsiranje, generiranje i manipulaciju JSON objekata te podržava sve tipove vrijednosti. Zbog jednostavne i optimizirane implementacije, nudi visoke performanse. Isto tako, ne oslanja se ni na kakve vanjske biblioteke, što je čini portabilnom u svim aplikacijama. Sljedeći programski isječak prikazuje funkcije za pripremanje objekta i njegovo slanje protokolom MQTT. Tema na koju šalje uređaj jest `device/{clientIdentifier}/data`, gdje je identifikator zapravo serijski naziv uređaja. Time je osigurano da svaki uređaj šalje na točno odgovarajuću temu. Koristi se kvaliteta usluge 0 (engl. *Quality of Service - QoS*) bez potvrde prijema jer se podaci šalju svakih pet sekundi, a senzorska mjerena se ne mijenjaju drastično u tako kratkom periodu, stoga si razvijeni sustav može priuštiti povremeni gubitak poruka. Ovim se načinom također štedi vrijeme čekanja i resursi koji su korištenom razvojnom sustavu potrebni.

```
void prepareJSONMessage(float temperature, float humidity
    , float moisture, uint8_t *buffer, size_t *length) {
    // Create the main JSON object
    cJSON *jsonObject = cJSON_CreateObject();
```

```

    struct timeval now;
    gettimeofday(&now, NULL);
    int64_t time_in_ms = (int64_t)now.tv_sec * 1000 + now.
        tv_usec / 1000;
    cJSON_AddNumberToObject(jsonObject, "timestamp",
        time_in_ms);

    cJSON_AddStringToObject(jsonObject, "device_id",
        CLIENT_IDENTIFIER);

    // Create a nested JSON object for data
    cJSON *dataObject = cJSON_CreateObject();
    cJSON_AddNumberToObject(dataObject, "temperature",
        temperature);
    cJSON_AddNumberToObject(dataObject, "humidity",
        humidity);
    cJSON_AddNumberToObject(dataObject, "moisture",
        moisture);

    // Add the data object to the main JSON object
    cJSON.AddItemToObject(jsonObject, "data", dataObject);

    // Print the JSON object to a string
    char *jsonString = cJSON_PrintUnformatted(jsonObject);

    // Copy the JSON string to the buffer and set the
    // length
    *length = strlen(jsonString);
    memcpy(buffer, jsonString, *length);

    // Clean up
    cJSON_Delete(jsonObject);
    cJSON_free(jsonString); // free the allocated string
}

```

```

void sendMessage( float temperature, float humidity, float
moisture) {
    prepareJSONMessage(temperature, humidity, moisture,
    payloadBuffer, &payloadLength);
    bool status = false;

    status = PublishToTopic( MQTT_DATA_TOPIC,
MQTT_DATA_TOPIC_LENGTH,
( char * ) payloadBuffer,
payloadLength );

    if( status == false )
    {
        LogError( ( "Failed to publish to topic: %.*s.",
MQTT_DATA_TOPIC_LENGTH,
MQTT_DATA_TOPIC ) );
    }
}

```

**Isječak koda 5.8:** Funkcije za pripremanje JSON objekta i njegovo slanje protokolom MQTT

Iz prve funkcije vidljivo je da uređaj u JSON objekt pohranjuje očitane vrijednosti temperature, vlage zraka te vlažnosti tla. Također, šalje identifikator uređaja kako bi se naznačilo koji uređaj šalje podatke. Isto tako, šalje se i vremenska oznaka, što je pogodno za buduću obradu i analizu podataka. Budući da mikrokontroler interno mjeri vrijeme od početka prvog pokretanja nakon instalacije *firmwarea*, potrebno je sinkronizirati njegov interni sat s lokalnim vremenom. Kako bi se osigurano točno vrijeme, mikrokontroler se povezuje na SNTP poslužitelj. Protokol SNTP (engl. *Simple Network Time Protocol*) jest protokol koji omogućava uređajima da se sinkroniziraju s mrežnim vremenskim poslužiteljima kako bi dobili precizno vrijeme [17]. Budući da je uređaj ranijim procesima već spojen na Wi-Fi, potrebno je korištenjem gotove biblioteke tvrtke *Espressif* spojiti se na poslužitelj te dohvatiti trenutno vrijeme i sinkronizirati vlastito u skladu s dohvaćenim. Uređaj se spaja na pool.ntp.org što je domensko ime za globalni mrežni projekt koji pruža vremenske poslužitelje za sinkronizaciju vremena putem protokola NTP (engl. *Network Time Protocol*). Ovu je sinkronizaciju potrebno napraviti samo jednom, i to nakon spajanja na internet. Sljedeći

programske je prikazuje funkciju koja se spaja na poslužitelj i obavlja sinkronizaciju. Ugrađen je i mehanizam pokušaja ponovnog spajanja svake dvije sekunde u najviše deset iteracija ukoliko prvotno spajanje i dohvati nisu uspješni.

```
void initialize_ntp(void)
{
    ESP_LOGI(TAG, "Initializing NTP");
    esp_ntp_setoperatingmode(NTP_OPMODE_POLL);
    esp_ntp_setservername(0, "pool.ntp.org");
    ntp_set_time_sync_notification_cb(
        time_sync_notification_cb);
    esp_ntp_init();
}

void obtain_time(void)
{
    initialize_ntp();

    // Wait for time to be set
    time_t now = 0;
    struct tm timeinfo = { 0 };
    int retry = 0;
    const int retry_count = 10;

    while (timeinfo.tm_year < (2024 - 1900) && ++retry <
           retry_count) {
        ESP_LOGI(TAG, "Waiting for system time to be set...
(%d/%d)", retry, retry_count);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
        time(&now);
        localtime_r(&now, &timeinfo);
    }

    if (retry < retry_count) {
        ESP_LOGI(TAG, "Time synchronized successfully");
    } else {

```

```

    ESP_LOGE(TAG, "Failed to synchronize time");
}
}

```

**Isječak koda 5.9:** Funkcije za sinkronizaciju vremena s SNTP poslužiteljem

Nakon generiranja JSON objekta, ovako izgleda konačan objekt koji uređaj šalje AWS-u:

```
{
  "timestamp": 123456789,
  "device_id": "serialNumber123",
  "data": {
    "temperature": 23.00,
    "humidity": 55.00,
    "moisture": 62.00
  }
}
```

**Isječak koda 5.10:** JSON objekt za slanje na platformu

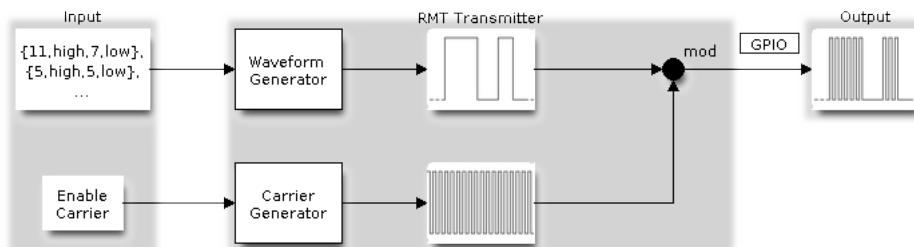
### 5.1.5. Sjena uređaja

Budući da nije kreirana nikakva dodatna aplikacija koja bi koristila sjenu uređaja i tražila zahtjev za promjenom stanja odnosno sjene, uređaj sam sebi šalje zahtjev za promjenom i isto tako šalje i ažurirano stanje. Promjenjivo stanje uređaja simulirano je bojom LED diode koja konstantno treperi periodom od dvije sekunde.

LED dioda integrirana je u razvojni sustav i vrsta je adresirajuće LED diode. Takve diode mogu se pojedinačno kontrolirati unutar trake ili matrice, čime su omogućeni složeni svjetlosni efekti i uzorci. Za razliku od običnih LED dioda koje se mogu samo uključiti ili isključiti, adresirajuće LED diode imaju integrirane upravljačke čipove koji omogućuju svakom pikselu u traci da primi jedinstvenu naredbu [8]. Pri upravljanju LED diodom, koristi se način rada RMT (engl. *Remote Control Transceiver*). To je periferni sklop koji služi kao infracrveni primopredajnik. Međutim, zbog fleksibilnosti formata podataka, RMT se može proširiti na svestrani primopredajnik opće namjene, koji odašilje ili prima i druge vrste signala. Iz perspektive slojevitosti mreže, hardver sadrži i fizički i sloj podatkovne veze. Fizički sloj definira komunikacijski medij i reprezentaciju signala bita, dok sloj podatkovne veze definira format RMT okvira.

Ovaj periferni sklop koristi se za generiranje i dekodiranje nizova impulsa, što ga čini prikladnim za zadatke koji uključuju precizno mjerjenje vremena, kao što je pokretanje adresirajućih LED dioda. One zahtijevaju striktno vremensko određivanje kako bi ispravno protumačile podatkovne signale [14].

RMT hardver ima vlastiti format uzorak podataka zvan RMT simbol. Svaki simbol sastoji se od dva para po dvije vrijednosti. Prva vrijednost u paru je 15-bitna vrijednost koja predstavlja trajanje signala u jedinicama RMT taktova. Druga u paru je 1-bitna vrijednost koja predstavlja logičku razinu signala, tj. visoku ili nisku. Slika 5.6 prikazuje način na koji funkcioniра RMT odašiljač. Upravljački program kodira korisničke podatke u format podataka tipa RMT, i zatim odašiljač može generirati valne oblike prema artefaktima kodiranja. Također je moguće modulirati visokofrekventni nosivi signal prije usmjeravanja na GPIO izlaze.



**Slika 5.6:** Pregled RMT odašiljača [14]

Sljedeći programski isječak prikazuje konfiguraciju LED diode kao i zadatak koji vrši treperenje. Koristi se biblioteka *led\_strip*, što je službeni upravljački program (engl. *driver*) za adresirajuće LED diode u modulima ESP32. Konfiguraciji se pridje luje GPIO pin na kojem se nalazi dioda te se postavlja razlučivost RMT signala na 10 MHz. Ovo određuje vremensku preciznost signala poslanih LED diodi. Isto tako, moguće je koristiti izravan pristup memoriji (engl. *Direct Memory Access - DMA*), no u ovom slučaju nije potreban. U glavnom je programu kreiran poseban FreeRTOS zadatak za treperenje LED diode. Kreirana je posebna struktura *color\_t* koja predstavlja boju u RGB formatu i odabrana se boja proslijedi funkciji *blink\_led()*. Također, kreirana je posebna funkcija koja na temelju tekstualne riječi boje generira boju u RGB formatu. U razvijenom sustavu podržane su sljedeće boje: crvena, zelena, plava, žuta, ljubičasta i bijela. Parametar *color\_name* globalna je varijabla koja se postavlja u zadatku koji izvršava slanje poruka i ažuriranje sjene uređaja.

```

typedef struct {
    uint8_t r;

```

```

        uint8_t g;
        uint8_t b;
    } color_t;

static void configure_led()
{
    led_strip_config_t strip_config = {
        .strip_gpio_num = BLINK_GPIO,
        .max_leds = 1, // at least one LED on board
    };
    led_strip_rmt_config_t rmt_config = {
        .resolution_hz = 10 * 1000 * 1000, // 10MHz
        .flags.with_dma = false,
    };
    ESP_ERROR_CHECK(led_strip_new_rmt_device(&strip_config,
                                             &rmt_config, &led_strip));
    led_strip_clear(led_strip);
}

static void blink_led(uint8_t s_led_state, color_t color)
{
    if (s_led_state) {
        led_strip_set_pixel(led_strip, 0, color.r, color.g,
                            color.b);
        led_strip_refresh(led_strip);
    } else {
        led_strip_clear(led_strip);
    }
}

void blinky_task(void *pvParameters) {
    uint8_t s_led_state = 0;
    color_t color;

    configure_led();
}

```

```

while (1) {
    if (s_led_state == true) {
        color = get_color_from_string(color_name);
    }

    blink_led(s_led_state, color);
    s_led_state = !s_led_state;
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
}

```

#### Isječak koda 5.11: Upravljanje LED diodom

Kao što je ranije spomenuto, uređaj najprije generira zahtjev za promjenom stanja, odnosno pošalje željeno stanje AWS-u i na temelju tog zahtjeva mijenja stvarno stanje i prijavljuje ga. Željeno je stanje nasumičan odabir jedne od ranije navedenih boja u tekstualnom obliku. Razvijeni sustav koristi samo jednu sjenu uređaja, i to klasičnu. Svaka tema vezana za sjenu uređaja povezana je sa stvari odnosno njezinim imenom, te se slanjem na pojedinačne teme podrazumijeva da je u temu integriran naziv stvari o čijoj se sjeni uređaja radi. Funkcionalnost sjene uređaja funkcioniра na sljedeći način: najprije, uređaj pokušava obrisati sjenu ako postoji. Za to se mora pretplatiti na teme */delete/accepted* i */delete/rejected* te objaviti na temu *delete* i primati poruke na pretplaćenim temama. Brisanje se smatra uspješnim u ova dva slučaja:

- poruka je stigla na temu */delete/accepted*,
- poruka je stigla na temu */delete/rejected* s greškom 404, što znači da pri pokušaju brisanja sjena uređaja ne postoji.

Uređaj se nakon uspješnog brisanja odjavljuje s pretplaćenih tema i prijavljuje se na teme za ažuriranja sjene uređaja. Zatim uređaj nasumično generira boju koja će predstavljati željeno stanje uređaja i šalje je na temu */update*, na koju je ujedno i pretplaćen. Ta ista poruka stigne na pretplaćenu temu i uređaj ažurira trenutnu boju LED diode u diodu želenog stanja i zatim šalje prijavljeno stanje na platformu. Verzije sjene uređaja automatski se povećavaju za jedan. Isto tako, svaki zahtjev ima vlastiti klijentski token koji je postavljen na serijski broj uređaja. Time je osigurano da se sjene različitih uređaja ne mijesaju, nego da uređaj samom sebi šalje zahtjev. Po primitku željene poruke, program provjerava odgovara li klijentski token vlastitom tokenu, te u tom slučaju odobrava promjenu stanja, mijenja boju LED diode i šalje platformi novo

prijavljeno stanje. Sljedeći programski isječak prikazuje skraćenu verziju opisanog procesa preplate, brisanja sjene, odjave, ponovne preplate te objave ažuriranja sjene uređaja. Iz isječka se može vidjeti da se, za razliku od slanja podataka, proces komunikacije s platformom vezano za sjenu uređaja odvija kvalitetom usluge QoS1, što znači da sustav čeka odgovor na poruku. Time se osigurava da će se stanje stvarno ažurirati na novu vrijednost.

```

LogInfo( ( "Sub to '/delete/accepted' and '/delete/
           rejected' topics." ) );
status = SubscribeToTopic( SHADOW_TOPIC_STR_DELETE_ACC(
    THING_NAME, SHADOW_NAME ),
    SHADOW_TOPIC_LEN_DELETE_ACC( THING_NAME_LENGTH,
    SHADOW_NAME_LENGTH ) );
status = SubscribeToTopic( SHADOW_TOPIC_STR_DELETE_REJ(
    THING_NAME, SHADOW_NAME ),
    SHADOW_TOPIC_LEN_DELETE_REJ( THING_NAME_LENGTH,
    SHADOW_NAME_LENGTH ) );
LogInfo( ( "Publish to 'delete' topic to try to delete
           the shadow doc." ) );
status = PublishToTopicQoS1( SHADOW_TOPIC_STR_DELETE(
    THING_NAME, SHADOW_NAME ),
    SHADOW_TOPIC_LEN_DELETE( THING_NAME_LENGTH,
    SHADOW_NAME_LENGTH ),
    updateDocument,
    0U );

/* Unsubscribe from topics... */
/* Subscribe to /update/delta topics... */

LogInfo( ( "Send desired color." ) );
(void) memset( updateDocument,
0x00,
sizeof( updateDocument ) );

const char* randomColor = get_random_color_string();

```

```

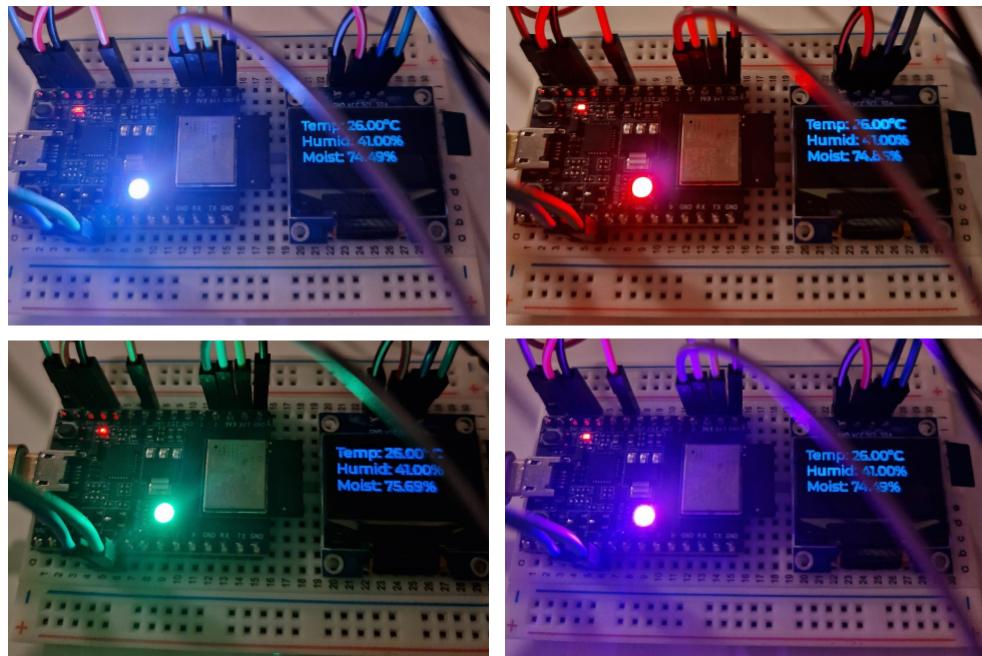
snprintf( updateDocument,
    SHADOW_DESIRED_JSON_LENGTH + 1,
    SHADOW_DESIRED_JSON,
    randomColor,
    clientToken );

status = PublishToTopicQoS1( SHADOW_TOPIC_STR_UPDATE(
    THING_NAME, SHADOW_NAME ),
    SHADOW_TOPIC_LEN_UPDATE( THING_NAME_LENGTH,
    SHADOW_NAME_LENGTH ),
    updateDocument,
    ( SHADOW_DESIRED_JSON_LENGTH + 1 ) );

```

**Isječak koda 5.12:** Proces ažuriranja sjene uređaja

Slika 5.7 prikazuje nekoliko različitih boja LED diode koje se kontroliraju sjenom uređaja.



**Slika 5.7:** Različite boje LED diode

### 5.1.6. Ažuriranje softvera

Programska potpora za ažuriranje softvera koristi OTA biblioteku u sklopu ranije spomenutog razvojnog paketa za integraciju platforme AWS i ESP32 uređaja. Koristi se

protokol HTTP za primanje datoteke ažuriranja radi manjeg konflikta s postojećom programskom potporom koja koristi MQTT za slanje poruka i sjenu uređaja. Za dodatno korištenje MQTT veze, bilo bi potrebno uvesti mehanizam dretvi i semafora radi sinkronizacije događaja i MQTT veze.

Za korištenje OTA funkcionalnosti potrebno je kreirati dodatne particije na uređaju. Koriste se dvije aplikacijske particije, što omogućava uređaju da zadrži radnu verziju *firmwarea* dok se novi preuzima i instalira na drugu particiju. Na taj način, ako dođe do problema tijekom preuzimanja ili instalacije novog softvera, uređaj se može sigurno vratiti na prethodnu verziju (engl. *rollback*) i nastaviti rad bez prekida. Također, dodatna particija omogućava neometan rad trenutnog programa dok se novi preuzima u pozadini, pogotovo u slučajevima loše mreže ili velike količine podataka. To smanjuje i vrijeme neaktivnosti. Nadalje, potrebna je i treća particija koja je podatkovna. Ona se koristi za čuvanje metapodataka vezanih za OTA ažuriranja. Ti podaci uključuju informacije kao što su trenutno aktivna aplikacija odnosno koja se particija koristi, status posljednjeg ažuriranja, te informacije potrebne za odlučivanje o prebacivanju između particija nakon ponovnog pokretanja uređaja. Kad se novo ažuriranje preuzme i instalira na drugu particiju, podatkovna se particija ažurira s informacijama koju particiju treba učiti pri sljedećem pokretanju. Tako uređaj može zaključiti koju particiju treba koristiti pri ponovnom pokretanju. Isto tako, ova particija može sadržavati informacije o verifikaciji integriteta i valjanosti novog *firmwarea*. Primjerice, može sadržavati digitalne potpise koji se koriste je li novi softver ispravno preuzet i da nije modificiran.

**Tablica 5.4:** Particije na uređaju

Ime	Vrsta	Podvrsta	Pomak	Veličina
<i>nvs</i>	data	nvs	0x9000	24K
<i>storage</i>	data	nvs		16K
<i>phy_init</i>	data	phy		4K
<i>factory</i>	app	factory		1800K
<i>spiffs_storage</i>	data	spiffs		960K
<i>otadata</i>	data	ota		8K
<i>ota_0</i>	app	ota_0		500K
<i>ota_1</i>	app	ota_1		500K

Tablica 5.4 prikazuje sve particije na uređaju, njihove vrste i veličine. Osim ranije opisanih particija potrebnih za OTA ažuriranja, tu se nalazi još nekoliko particija neophodnih za rad sustava. Navedene su i ranije opisane particije memorija tipa NVS i

SPIFFS. Particija *factory* služi za pohranu osnovnog *firmwarea* koji je inicijalno programiran na uređaj. Particija *phy\_init* koristi se za pohranu konfiguracijskih podataka fizičkog sloja (engl. *Physical Layer - PHY*). Odnosi se na fizičke karakteristike bežične komunikacije poput kalibracijskih parametara i postavki za Wi-Fi. Particija NVS koristi se za pohranu Wi-Fi vjerodajnica, a posljednja particija *storage* također je vrsta memorije tipa NVS, no koristi se za pohranu novodobivenih certifikata od AWS-a. Kada se zbroje veličine svih particija, iznos je jednak otprilike 4 MB, što je jednako veličini memorije tipa *flash* na uređaju ESP32-C3. Budući da su particije za OTA softver znatno manje nego izvorna memorija, primljeno ažuriranje mora biti manjeg opsega u odnosu na izvorni *firmware*.

Iako se u razvijenom sustavu koristi HTTP za OTA ažuriranje, i dalje je potrebna MQTT veza. Razvojni sustav mora se pretplatiti na sve teme iz isječaka 4.6. i 4.7. radi upravljanje OTA procesom, uključujući primanje obavijesti o novim ažuriranjima, praćenje statusa preuzimanja i instalacije, te prijavljivanje grešaka. Protokol HTTP koristi se isključivo za preuzimanje softvera. Budući da se koriste oba protokola, potrošnja memorije je nešto veća.

Kao što je spomenuto u ranijem poglavlju, OTA agent služi za pojednostavljenje OTA ažuriranja. Agent funkcioniра као kompleksан процес који омогућује сигурно и pouzdano ažuriranje softvera преко мреже. Pretplaćuje на теме послова како би пратио надолазећа аžuriranja и обавijести о новим OTA задацима. Када прими поруку, agent проверава садржај задатка, укључујући проверу интегритета и аутентичности добивене датотеке помоћу дигиталних потписа која се врши локално пohранjenim certifikatom. Тaj certifikat mora бити jednak certifikatu који је кориштен за потписивање добивене датотеке. Генериран је локално на рачуналу помоћу библиотеке *OpenSSL* и исто тако касније кориштен за потпис датотеке у облаку. За генерирањеcertifikата и njemu припадног privatnog ključa кориштен је алгоритам ECDSA (engl. *Elliptic Curve Digital Signature Algorithm*). Када је задатак потврђен, agent користи протокол HTTP за preuzimanje softvera са specificiranog URL-а. Тijekom preuzimanja, agent проверава интегритет користећи *hash* алгоритме. *Firmware* се preuzima у једну од OTA particija te nakon kompletног preuzimanja, agent још једном проверава интегритет цијеле датотеке прије него nastavi с instalацијом. Ако је нови код valjan, конфигурира програм за покretanje uređaja (engl. *bootloader*) да користи нову particiju при idućem pokretanju. Напослjetku ponovno покреће uređaj да се покrene nova verzija softvera. Tokom cijelog opisanog процеса, agent паралелно шalje statuse natrag platformi putem MQTT везе. Порuke укључују информације о почетку preuzimanja, napretku, uspjehu ili neuspjehu preuzimanja te instalaciji.

Sljedeći programski isječak prikazuje pokretanje OTA agenta. Veći dio funkcionalnosti OTA agenta integriran je u korištenu biblioteku koja interno obavlja rad agenta. Funkcija radi na sljedeći način:

1. inicijalizira se OTA agent,
2. pokrene se petlja koja omogućuje OTA ažuriranje i vraća statistiku o primljenim paketima svake sekunde,
3. u slučaju prekida MQTT veze, obustavljuju se sve operacije OTA agenta.

```
/* Set OTA Library interfaces.*/
setOtaInterfaces( &otaInterfaces );

/* Set OTA Code Signing Certificate */
otaPal_SetCodeSigningCertificate( pcAwsCodeSigningCertPem
);

/* Init OTA Library. */
otaRet = OTA_Init( &otaBuffer,
&otaInterfaces,
( const uint8_t * ) ( CLIENT_IDENTIFIER ),
otaAppCallback );

/* Create OTA Task. */
pthread_create( &threadHandle, NULL, otaThread, NULL )

/* OTA Demo loop. */

/* Wait till OTA library is stopped, output statistics
   for currently running
   * OTA job */
while( ( ( state = OTA_GetState() ) !=

OtaAgentStateStopped ) )

{
    /* Loop to receive packet from transport interface. */
    mqttStatus = MQTT_ProcessLoop( &mqttContext );
```

```

if( ( mqttStatus == MQTTSuccess ) || ( mqttStatus == MQTTNeedMoreBytes ) )
{
    /* Get OTA statistics for currently executing job. */
    OTA_GetStatistics( &otaStatistics );

    LogInfo( ( " Received: %"PRIu32" Queued: %"PRIu32"
               Processed: %"PRIu32" Dropped: %"PRIu32"",
               otaStatistics.otaPacketsReceived,
               otaStatistics.otaPacketsQueued,
               otaStatistics.otaPacketsProcessed,
               otaStatistics.otaPacketsDropped ) );

    Clock_SleepMs( OTA_EXAMPLE_LOOP_SLEEP_PERIOD_MS );
}

if (mqttSessionEstablished != true)
{
    /* Suspend OTA operations. */
    otaRet = OTA_Suspend();
}
}

/* Wait for OTA Thread. */
returnJoin = pthread_join( threadHandle, NULL );

```

#### **Isječak koda 5.13:** Rad OTA agenta

Kao što je opisano, po primitku MQTT poruke o dostupnom ažuriranju, kreira se HTTP zahtjev za dohvatom novog softvera. Uredaj koristi URL adržan u MQTT poruci kako bi započeo preuzimanje novog koda. Koristi se GET zahtjev za preuzimanje datoteke, a URL može biti poveznica na AWS S3 pohranu ili pak neki drugi poslužitelj koji je postavljen za distribuciju *firmwarea*. U ovom slučaju, koristi se AWS S3 za dohvat datoteke. Odgovor na zahtjev jest datoteka koja se preuzima u segmentima. Sljedeći isječak prikazuje kreiranje zahtjeva te spajanje na S3 odakle će se preuzeti datoteka. Funkcija za obradu primljenog odgovora handleHttpResponse() stavlja događaj u međuspremnik odakle će agent preuzeti zadatka.

## 5.2. Programska potpora za oblak

Programska potpora za platformu AWS većim dijelom nije kod u standardnom smislu, no kako bi se uređaj i platforma uopće mogli komunicirati međusobno, potrebno je omogućiti povezivanje i komunikaciju na samoj platformi. Sva se komunikacija odvija u istoj AWS regiji. Zbog dostupnosti većine IoT usluga i relativne geografske bliskosti, korištena je regija *eu-north-1* odnosno Stockholm, Švedska.

Isto tako, važno je istaknuti kako se sve opisane radnje u sustavu AWS mogu izvršavati pomoću naredbenog retka platforme AWS (engl. *Command-line interface - CLI*), no radi jednostavnosti i preglednosti, korišteno je korisničko sučelje platforme.

Programska potpora sastoji se od sljedećih segmenata:

- omogućavanje dinamičke registracije uređaja,
- slanje novog softvera na uređaj,
- pristup zadnjem stanju uređaja nakon gubitka veze,
- obrada podataka dobivenih protokolom MQTT,
- pohrana podataka.

### 5.2.1. Dinamička registracija uređaja

Kao što je ranije opisano, u razvijenom sustavu koriste se certifikati zahtjeva za prvotno spajanje na platformu AWS, stoga je potrebno kreirati odgovarajući predložak za registraciju. Predložak također mora imati dodijeljenu politiku koja autorizira certifikat zahtjeva i ta se politika dodjeljuje generiranim certifikatima zahtjeva. Odabiru se politike koje omogućavaju točno onoliko koliko je potrebno za spajanje u sustav, a to su dopuštenja za MQTT komunikaciju i spajanje na AWS. Također, potrebno je odabrati koji su certifikati valjni kao zahtjev. Poželjno je i dodijeliti Lambda funkciju kao predregistracijsku provjeru dodatne valjanosti zahtjeva i poslanih parametara. Za razvijeni sustav nije kreirana Lambda funkcija, što znači da je svaki zahtjev automatski odobren. Isto tako, moguće je automatski pri registraciji uređaja kreirati i pripadnu stvar (engl. *thing*), što je korisno za kasniju organizaciju i pregled certifikata. Svaka stvar može imati modularan naziv, ovisno o parametrima koji se pošalju. Za ovaj sustav postavljen je prefiks *ESP32Thing\_* koji označava da su uređaji vrste ESP32, a ostatak naziva stvari ovisi o serijskom broju samog uređaja. To osigurava da svaki uređaj kreira jedinstvenu stvar u AWS-u. Moguće je odabrati i vrstu stvari (engl. *thing type*) koja će se automatski pridijeliti stvari, te u ovom slučaju kreirana je nova vrsta stvari

naziva ESP32-C3. Stvarima se može dodijeliti i grupa, no u ovom sustavu nije bilo potrebe za kreiranjem dodatne grupe stvari budući da se povezuje samo jedna vrsta uređaja u sustav. Naposlijetku, potrebno je odabrati koje će se politike dodijeliti novogeneriranom jedinstvenom certifikatu, čime će uređaj dobiti pristup uslugama sustava AWS. Odabранe politike u ovom sustavu imaju sva dopuštenja radi lakše demonstracije funkcionalnosti. U nastavku se može vidjeti isječak kreiranog predloška, odnosno parametri stvari koja se kreira korištenjem predloška. U odsječku se isto tako može vidjeti funkcija za kreiranje imena stvari koja koristi predefinirani prefiks i serijski broj uređaja.

```

"thing": {
    "Type": "AWS::IoT::Thing",
    "OverrideSettings": {
        "AttributePayload": "MERGE",
        "ThingGroups": "DO_NOTHING",
        "ThingTypeName": "REPLACE"
    },
    "Properties": {
        "AttributePayload": {},
        "ThingGroups": [],
        "ThingName": {
            "Fn::Join": [
                "",
                [
                    "ESP32Thing_",
                    { "Ref": "SerialNumber" }
                ]
            ]
        },
        "ThingTypeName": "ESP32-C3"
    }
}

```

**Isječak koda 5.14:** Odjeljak *stvar* u predlošku za registraciju

Na slici 5.8 nalazi se popis politika koje su dodijeljene uređaju registriranom u sustav. Omogućene su sve politike, odnosno dodijeljene su sve dozvole koje uređaj može imati. Politika *DevicePolicy* odnosi se na komunikaciju MQTT protokolom, *DeviceShadowPolicy* na akcije vezane uz sjenu uređaja, *JobPolicy* na izvršavanje i dohvat poslova te *CertificatePolicy* dozvoljava povezivanje certifikatom.

Policies (4) <a href="#">Info</a>	
AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.	
	Name
<input type="checkbox"/>	<a href="#">JobPolicy</a>
<input type="checkbox"/>	<a href="#">DeviceShadowPolicy</a>
<input type="checkbox"/>	<a href="#">DevicePolicy</a>
<input type="checkbox"/>	<a href="#">CertificatePolicy</a>

**Slika 5.8:** Popis politika dodijeljenih registriranom uređaju

### 5.2.2. Obrada i pohrana dobivenih podataka

Podaci poslani protokolom MQTT u ranije prikazanom formatu JSON šalju se platformi na određenu temu definiranu serijskim nazivom uređaja. Svi se podaci šalju sigurno uspostavljenom vezom na vrata 8883 i krajnju točku brokera koju AWS generira ovisno o regiji kojoj se šalju podaci. Korištenjem testnog klijenta koji nudi AWS i pretplatom na željene teme moguće je u stvarnom vremenu pratiti podatke koji uređaj šalje.

Primljeni se podaci moraju pohraniti u bazu podataka kako bi se kasnije mogli koristiti za pregled i analizu. AWS nudi uslugu preusmjeravanja poruka (engl. *message routing*) protokom podataka između povezanih uređaja i aplikacija. Koristi pravila za usmjeravanje (engl. *rules engine*) kako bi obradila i preusmjerila poruke koje dolaze s uređaja prema drugim uslugama ili vanjskim krajnjim točkama. Moguće je usmjeriti poruke izravno prema sustavima za pohranu koji se nude u sklopu AWS usluga, no isto tako i prema Lambda funkcijama koje mogu vršiti detaljniju obradu podataka. Pravila usmjeravanja definiraju se SQL upitima nad MQTT temama, čime se omogućava filtriranje, transformacija i usmjeravanje podataka na osnovu specifičnih kriterija. Upitima je moguće izdvojiti specifična polja iz JSON poruka ili pak računati nove vrijednosti na temelju dobivenih.

Svaka ruta mora imati jedinstveno ime te upit kojim se dohvaćaju podaci. Upit je standardan SQL upit, no izvor podataka je tema na koju dolaze poruke u AWS. Unutar tema također je moguće koristiti i zamjenske znakove (engl. *wildcards*) kako bi upit odgovarao više tema. U razvijenom sustavu korišten je upit koji obuhvaća sve teme na koje uređaji šalju svoje podatke, neovisno o serijskom broju uređaja. Upit se nalazi u sljedećem odlomku koda.

```
SELECT * FROM 'device/+data'
```

#### Isječak koda 5.15: SQL upit rute za podatke s uređaja

Nadalje, potrebno je odabrati kamo će se odabrani podaci slati. Dobivene je podatke moguće slati u zapise (engl. *logs*), metrike, u baze podataka unutar AWS-a, na novu MQTT temu, ili pak preusmjeriti podatke Lambda funkciji. Iako AWS nudi direktno slanje u baze podataka, poput baze DynamoDB, ovo rješenje nije odabранo zbog komplikiranog dohvata podataka i njihove vizualizacije u korištenoj web aplikaciji. Stoga se podaci preusmjeravaju na Lambda funkciju koja vrši daljnju obradu i slanje podataka u bazu.

Još jedna od usluga platforme AWS jest Amazon Timestream koja nudi potpuno upravljane baze podataka vremenskih serija za radna opterećenja od upita niske latencije pa sve do unosa velikih podataka. Usluga je namijenjena podacima koji sadrže vremenuku oznaku za analizu vremenskih serija. Automatski skalira kapacitet prema potrebi, čime omogućava rukovanje mnoštvom podataka bez ručnog dodavanja resursa. Arhitektura je dizajnirana tako da omogućava brzi unos i pokretanje upita, pružajući visoku propusnost i nisku latenciju. Također, automatizira procese arhiviranja i upravljanja podacima. Korisnici mogu definirati politike zadržavanja podataka kako bi se stari podaci automatski premjestili u jeftinijem i sporijem spremniku za pohranu, dok noviji podaci ostaju u pohrani brzog pristupa. Time se optimiziraju troškovi i performanse bez ručne intervencije. U sklopu usluge Amazon Timestream nude se dvije baze podataka vremenskih serija: LiveAnalytics i InfluxDB. Baza LiveAnalytics primarno je namijenjena podacima koji zahtijevaju detaljnu analizu i praćenje velike količine podataka u stvarnom vremenu. Ima ugrađene analitičke funkcije za praćenje trendova i uzoraka. Isto tako, pogodna je za pohranu zapisa i događaja. InfluxDB je baza podataka otvorenog koda, što je čini fleksibilnjom pri konfiguraciji u odnosu na LiveAnalytics. Baza je pogodna za pohranu metrika i očitanja s IoT uređajima [3]. Kao što je opisano, obje baze podataka pripadaju skupini baza vremenskih serija, koje spadaju u skupinu NoSQL bazi podataka. Takve baze pohranjuju podatke u strukturi drukčijoj od klasičnih relacijskih modela, primjerice u formatu JSON s parovima ključ-vrijednost. NoSQL sustavi nastali su iz novih zahtjeva za većom fleksibilnošću i boljim performansama u pohrani i obradi velike količine podataka, uglavnom zbog popularnosti interneta i internetskih tehnologija te sve veće količine podataka [27].

U razvijenom je sustavu odabrana baza InfluxDB za pohranu podataka zbog jednostavne konfiguracije i integracije s web aplikacijom. U InfluxDB, kanta (engl. *buc-*

*ket*) je osnovna jedinica za pohranu podataka. Kante služe kao logički spremnici za vremenske serije podataka te određuju gdje i kako se podaci čuvaju. Svi podaci unutar jedne kante grupirani su zajedno te se mogu jednostavno pretraživati i analizirati. Isto tako, nad kantom je moguće definirati politiku zadržavanja podataka (engl. *retention policy*) koja određuje koliko dugo će podaci biti zadržani prije nego se automatski obrišu. Također, pomoću njih se omogućava kontrola pristupa određenoj skupini podataka. Svaka kanta pripada organizaciji, što je radna okolina za skupinu korisnika. Sve nadzorne ploče, kanta i korisnici pripadaju jednoj organizaciji. Podaci su u bazi organizirani po stupcima gdje su postavljeni vremenski blokovi za mjerene vrijednosti, skup oznaka (engl. *tags*), te polje kojem podaci pripadaju. Svi podaci pohranjeni u InfluxDB imaju stupac `_time` koji pohranjuje vremenske oznake. Oznake su na disku pohranjene u formatu nanosekunde, no preciznost vremenske oznake podataka koji se šalju može se definirati pri samom slanju tih podataka u samom API pozivu. Granulacija može biti na svim razinama od sekunde do nanosekunde. Sljedeći važan stupac koji ima svaki zapis u bazi jest `_measurement` koji označava naziv mjerjenja. Podaci se grupiraju na temelju te vrijednosti. Posljednji stupac, `_field`, može imati beskočno mnogo parova ključ-vrijednost koji simboliziraju stvarna polja i mjerene vrijednosti. Isto tako, mjeranjima se mogu dodijeliti i oznake na temelju kojih se također može pretraživati i filtrirati. Važno je naglasiti kako polje `_field` nije indeksirano, što znači da pretraživanje po njima nije efikasno jer baza mora proći po svim unosima. Stoga se preporuča postavljati vrijednost običnih oznaka budući da je pretraživanje po oznakama indeksirano [18].

Nadalje, baza InfluxDB koristi specifičan format podataka za pohranu. Linijski protokol (engl. *line protocol*) koji koristi baza tekstualni je format za upisivanje točaka u vremenu. Jedan red teksta u formatu linijskog protokola predstavlja jednu podatkovnu točku. Sadrži informacije o vrijednosti mjerjenja, oznakama, vremenskoj oznaci i skupu polja. Sljedeći programski isječak prikazuje primjer podatka u formatu linijskog protokola koji se šalje u bazu podataka. Kao što je vidljivo iz isječka, obavezno je navesti naziv mjerjenja kojem podatak pripada. Isto tako, mogu se dodati i oznake za kasnije pretraživanje i grupiranje. Zatim se nalazi popis polja odvojen zarezom koji predstavljaju vrijednosti samih mjerena. Na kraju formata nalazi se obavezna vremenska oznaka. Podaci moraju striktno pratiti ovakav format kako bi baza uspješno parsirala i pohranila dobivene podatke. Format linijskog protokola jednostavan je za upotrebu te je kompaktan, čime smanjuje veličinu podataka i poboljšava efikasnost upisa i preuzimanja. Pogodan je jer podržava dinamičko dodavanje novih polja bez potrebe za izmjenom postojeće strukture podataka. S druge strane, ovakav format nije

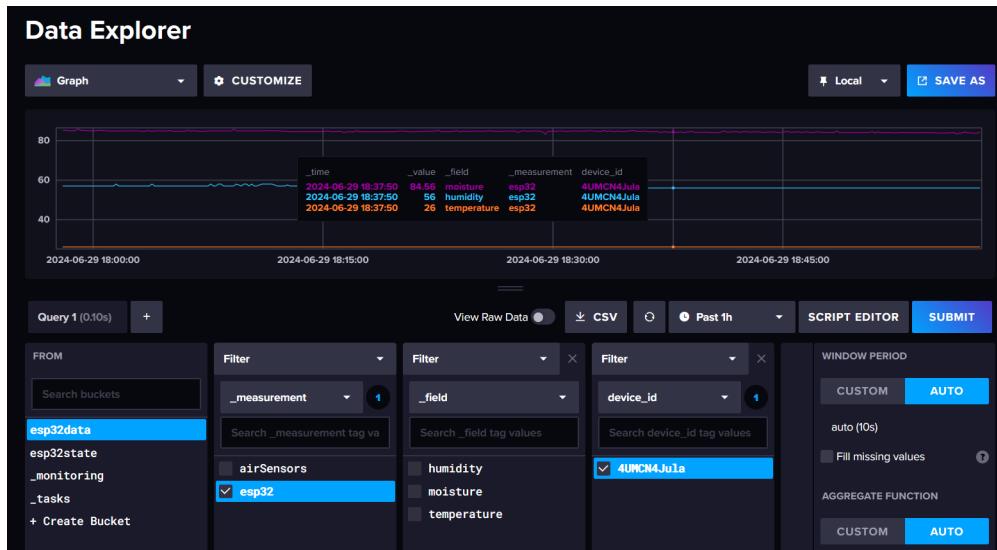
prikidan za kompleksne strukture podataka ili duboke hijerarhije. Isto tako, zbog jednostavnosti je osjetljiv na greške i formatiranje, što zahtijeva pažljivo rukovanje s formatom podataka.

```
esp32,device_id=MyId123 temp=28,co=11.1 123456789
| -----+-----+-----+-----+
|       |       |       |
|       |       |       |
+-----+-----+-----+-----+
|measurement|,tag_set| |field_set| |timestamp|
+-----+-----+-----+-----+
```

**Isječak koda 5.16:** Podatak u bazi InfluxDB u formatu linijskog protokola

Budući da se baza podataka nudi u sklopu usluge Amazon Timestream, nije potrebno kreirati vlastite resurse kako bi se baza pokrenula i održavala. Pri kreiranju baze podataka, potrebno je definirati ime baze i glavnog administratora s lozinkom koji se koristi pri prvom spajanju na bazu. Isto tako, potrebno je navesti ime organizacije i početne kante u koju će se spremati podaci. Zatim se konfigurira sama instanca na kojoj će baza biti pokrenuta, odnosno resursi instance. Bira se na temelju korištenog RAM-a, mrežne propusnosti i jačine CPU-a. Potrebno je upisati veličinu diska za pohranu. Baza može biti javno dostupna svima, ili se pak mogu definirati posebne virtualne privatne mreže u oblaku (engl. *virtual private cloud* - VPC) sa podmrežama. Radi jednostavnosti, baza je javno dostupna iz svih mreža. Važno je napomenuti kako, usprkos javnoj dostupnosti baze, potrebne su vjerodajnice odnosno korisnički podaci kako bi korisnik pristupio bazi. Za bazu razvijenog sustava korišteno je ime *timestream-esp32*, organizacija *fer* te kanta u koju se spremaju podaci naziva se *esp32data*. Slika 5.9 prikazuje sučelje baze InfluxDB te podaci koji su spremljeni u kantu *esp32data*. Mogu se vidjeti parametri po kojima je moguće filtrirati pretraživanje.

Kako je ranije spomenuto, podaci se moraju obraditi Lambda funkcijom kako bi bili pohranjeni u bazu podataka. Stoga je kreirana Lambda funkcija *ESP32Lambda* koja podatke pretvara u format linijskog protokola te ih šalje u InfluxDB. Ova funkcija pokrenuta je pri događaju (engl. *event-driven*), što znači da se svakim dolaskom poruka na preplaćene teme ova funkcija pokrene. AWS podržava brojne programske jezike putem izvršnih okolina u kojima su integrirane značajke potrebne za izvršavanje funkcije u željenom programskom jeziku. Odabir izvršne okoline za sobom povlači i operacijski sustav na kojem se funkcija izvršava. Ova je funkcija napisana u programskom jeziku Python zbog jednostavnosti i razumljivosti programskog koda. Funkcija



Slika 5.9: Sučelje za pregled podataka u bazi InfluxDB

je naziva `lambda_handler` koja predstavlja ulaznu točku (engl. *entrypoint*) za izvršavanje same funkcije. Iako se mogu napisati i pomoćne funkcije, ova je funkcija ključna za izvršavanje skripte. Sastoјi se od dva parametra koje joj proslijedi proces zaslužan za pokretanje same funkcije, a to su događaj i kontekst. Događaj je zapravo ulazni podatak primljen putem MQTT veze, a objekt konteksta pruža metode i svojstva koja daju informacije o pozivu, samoj funkciji i okolini izvođenja. Funkcija također može koristiti varijable okoline (engl. *environment variables*) koje se postavljaju odvojeno od samog koda. Postavljanjem varijabli jednostavno se odvajaju tajni parametri od same funkcije. Budući da funkcija zahtijeva stvaranje API poziva prema bazi kako bi zapisala podatke, potrebno je koristiti vanjske biblioteke za kreiranje HTTP zahjeva. Izvršna okolina Python koda sama po sebi ne pruža tu biblioteku, no moguće je stvoriti dodatan sloj nad Lambda funkcijom koja će sadržavati potreban paket. Slojevi su mehanizam koji omogućavaju dijeljenje i ponovno korištenje zajedničkog koda i resursa između više Lambda funkcija. Omogućava izolaciju dijelova koda, kreiranje i dijeljene biblioteka ili drugih resursa bez potrebe za mijenjanjem postojeće implementacije. Moguće je koristiti slojeve koje nudi AWS, odabrati vanjski sloj koji su kreirali drugi korisnici platforme, ili pak kreirati vlastiti. U razvijenom je sustavu korištena druga opcija, odnosno sloj koji integrira biblioteku `Requests` za kreiranje API poziva.

Kreirana Lambda funkcija nalazi se u sljedećem programskom isječku. URL baze podataka, organizacija te kanta za pohranu podataka dohvataju se iz varijabli okoline. Isto tako, kako bi funkcija uopće mogla pristupiti bazi, potrebno je generirati token u bazi koji omogućava pisanje u željenu kantu. Generirani je token također zapisan u

variabile okoline. Preciznost vremenske oznake postavljena je na milisekunde budući da podaci s uređaja ESP32-C3 dolaze u takvom formatu. Iz funkcije je vidljivo čitanje vrijednosti, stvaranje podatka u formatu linijskog protokola, slanje POST zahtjeva i čekanje na odgovor.

```
import json
import requests
import time
import os

BUCKET = os.getenv('BUCKET')
ORG = os.getenv('ORG')
INFLUXDB_URL = os.getenv('INFLUXDB_URL')
TOKEN = os.getenv('TOKEN')

def lambda_handler(event, context):
    try:
        measurement = 'esp32'
        timestamp = event["timestamp"]
        device_id = event["device_id"]
        event_data = event["data"]
        fields = ",".join([f"{key}={value}" for key, value in
event_data.items() if key != "timestamp" and key != "device_id"])
        data = f'{measurement},device_id={device_id} {fields}'
        data += f'timestamp{timestamp}'
        print(data)
        headers = {
            'Authorization': f'Token {TOKEN}',
            'Content-Type': 'text/plain'
        }
        params = {
            'org': ORG,
            'bucket': BUCKET,
            'precision': 'ms'
        }
    
```

```

response = requests.post(
    INFLUXDB_URL,
    headers=headers,
    data=data,
    params=params
)
response.raise_for_status()
print("Data sent to InfluxDB successfully")
except Exception as e:
    print("Error:", e)

return {
    'statusCode': 200,
    'body': json.dumps('Data processed and sent to
InfluxDB')
}

```

**Isječak koda 5.17:** Lambda funkcija za slanje podataka u InfluxDB

### 5.2.3. Sjena uređaja

Za ažuriranje sjene uređaja u AWS-u nije potrebna nikakva dodatna konfiguracija. Jedini je uvjet da stvar za koju je uređaj vezan postoji, no to je svakako osigurano registracijom na platformu. Uređaj kreira prvotnu klasičnu sjenu uređaja i ažurira je svakom iteracijom programa. Druge aplikacije mogu dohvatiti podatke o sjeni uređaja pretplatom na njegove teme. Prefiks za sve teme jest `$aws/things/{thingName}/shadow`. Budući da pregled sjene uređaja u AWS-u sam po sebi nije osobito koristan, po uzoru na proces pohrane senzorskih očitanja, kreirana je ruta koja poruke dospjele na temu ažuriranja sjene prosljeđuje Lambda funkciji, koja pak obrađuje podatke i sprema ih u postojeću bazu InfluxDB. Kreiran je SQL upit koja sve poruke dospjele na temu uspješnog ažuriranja, odnosno poruke prijavljenog stanja, prosljeđuje posebnoj funkciji.

```
SELECT * FROM '$aws/things/+shadow/update/accepted'
```

**Isječak koda 5.18:** SQL upit rute za sjene uređaja

Iako sami uređaji u prijavljenom stanju ne postavljaju serijski broj na temelju kojeg bi se izvorišni uređaji mogli razlučiti, svaka poruka sadrži ranije opisani klijentski identifikator na temelju kojeg se kasnije grupiraju podaci. Poruka u formatu JSON koja se rutom proslijedi Lambda funkciji nalazi se u sljedećem programskom isječku.

```
{  
    "state": {  
        "reported": {  
            "color": "WHITE"  
        }  
    },  
    "metadata": {  
        "reported": {  
            "color": {  
                "timestamp": 1719764472  
            }  
        }  
    },  
    "version": 15299,  
    "timestamp": 1719764472,  
    "clientToken": "deviceNumber123"  
}
```

#### Isječak koda 5.19:

Poruka ažurirane sjene

Lambda funkcija sjene uređaja *ESP32ShadowLambda* gotovo je identična funkciji za pohranu podataka, s manjim izmjenama dohvatanja identifikatora uređaja i promjene vremenske oznake. Biblioteka razvojnog sustava za sjenu uređaja automatski pridjeljuje vremensku oznaku u obliku sekunde, dok API prihvata podatke na razini milisekunde. Kreirana je nova kanta *esp32state* u bazi InfluxDB koja služi za pohranu stanja sjene. Generiran je i odgovarajući token pisanja u novu kantu, i shodno tome stvorene su nove varijable okoline. Ova se funkcija također bazira na dodatnom sloju koji sadrži paket *Requests* programskog jezika Python za komunikaciju putem API-ja. Isto tako, napravljena je provjera postoji li polje *desired* u JSON objektu, za slučaj da se na temi nađe poruka željenog stanja koja se ne sprema u bazu. Pohrana je namijenjena samo stvarnim ažuriranjima koji dolaze iz poruka prijavljenog stanja.

```
import json  
import requests
```

```

import time
import os

BUCKET = os.getenv('BUCKET')
ORG = os.getenv('ORG')
INFLUXDB_URL = os.getenv('INFLUXDB_URL')
TOKEN = os.getenv('TOKEN')

def lambda_handler(event, context):
    if "reported" not in event["state"].keys():
        print("Desired event, skip storing")
        return

    try:
        measurement = 'esp32'
        timestamp = event["timestamp"]
        current_state = event["state"]["reported"]
        device_id = event["clientToken"]
        fields = ",".join([f'{key}={value}' for key,
                           value in current_state.items() if key != "timestamp"
                           and key != "clientToken"])
        data = f'{measurement},device_id={device_id} {fields}'
        data += f'timestamp * 1000'

        headers = {
            'Authorization': f'Token {TOKEN}',
            'Content-Type': 'text/plain'
        }

        params = {
            'org': ORG,
            'bucket': BUCKET,
            'precision': 'ms'
        }

        response = requests.post(
            INFLUXDB_URL,
            headers=headers,
            data=data,

```

```

    params=params

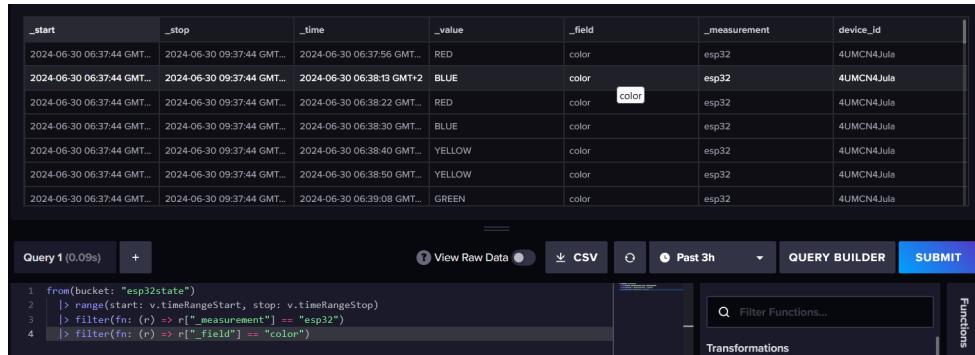
)
response.raise_for_status()
print("Data sent to InfluxDB successfully")
except Exception as e:
    print("Error:", e)

return {
'statusCode': 200,
'body': json.dumps('Data processed and sent to
InfluxDB')
}

```

**Isječak koda 5.20:** Lambda funkcija za preusmjeravanje poruka sjene uređaja

Slika 5.10 prikazuje podatke sjene uređaja u sučelju baze InfluxDB. Vidljive su promjene stanja kroz vrijeme, odnosno različite vrijednosti boje LED diode koje je uređaj prijavio.



**Slika 5.10:** Sučelje za pregled stanja uređaja u bazi InfluxDB

## 5.2.4. Ažuriranje softvera

Kako bi se mogao kreirati posao OTA ažuriranja i poslati uređaju, potrebno je neko-liko preduvjeta. Najprije, certifikat koji uređaj dobije mora imati omogućenu politiku poslova odnosno *JobPolicy*. AWS nakon kreiranja certifikata na zahtjev automatski dodjeljuje ovu politiku certifikatu odnosno uređaju. Nadalje, potrebno je kreirati S3 kantu koja će služiti kao pohrana binarnih datoteka ažuriranja. Isto tako, potrebno je kreirati novu ulogu u sigurnosnom centru AWS-a koja će omogućava pristup usluzi

AWS IoT Jobs, S3 pohrani te resursima za potpisivanje koda. Naposljeku, certifikat potpisivanja koda mora se učitati u oblak kako bi se njime potpisivale datoteke za slanje putem HTTP zahtjeva.

Što se tiče kreiranja spremnika u S3 pohrani, moguće je odabratizmeđu dvije vrste spremnika - spremnik opće namjene (engl. *general purpose*) ili spremnik imenika (engl. *directory bucket*). Spremnik opće namjene zadana je vrsta spremnika koji se koristi za većinu slučajeva upotrebe u S3. Podržavaju sve S3 značajke i većinu klase pohrane. Spremnici imenika koriste se za poslove koji zahtijevaju nisku latenciju. Mogu podržati stotine tisuća transakcija u sekundi, neovisno o broju direktorija unutar spremnika. Za razliku od spremnika opće namjene, spremnici imenika organiziraju ključeve hijerarhijski u direktorije umjesto korištenja prefiksa za organizaciju objekata. Spremnici se kreiraju na globalnoj razini te se regionalno repliciraju za visoku dostupnost i otpornost od ispadanja. Za potrebe razvijenog sustava kreiran je spremnik opće namjene *esp32ota-jgavran*. U njega je pohranjena binarna datoteka koja služi za demonstraciju OTA ažuriranja, a to je jednostavan program *blink.bin* koji periodički pali i gasi LED diodu. Svi korisnici imaju pravo čitanja iz kreiranog spremnika za lakši pristup objektu.

Lokalno kreirani certifikat s pripadnim privatnim ključem učitan je u sustav kako bi se njime digitalno potpisala binarna datoteka prije slanja na uređaj i tako potvrdila valjanost i cjevitost datoteke. Isto tako, kreirana je uloga *OTARole* koja će stvorenom OTA poslu omogućiti pristup uslugama AWS IoT Jobs, pohrani te potpisivanju koda. Popis dozvola koje uloga ima nalazi se na slici 5.11.

Permissions policies (6) <a href="#">Info</a>			
<a href="#">C</a> <a href="#">Simulate</a> <a href="#">Remove</a> <a href="#">Add permissions</a> ▾			
Filter by Type <input type="text" value="Search"/> <a href="#">All types</a>			
Policy name	Type	Attached entities	
<input type="checkbox"/> <a href="#">AmazonFreeRTOSOTAUpdate</a>	AWS managed	1	
<input type="checkbox"/> <a href="#">AWSIoTLogging</a>	AWS managed	1	
<input type="checkbox"/> <a href="#">AWSIoTRuleActions</a>	AWS managed	1	
<input type="checkbox"/> <a href="#">AWSIoTThingsRegistration</a>	AWS managed	3	
<input type="checkbox"/> <a href="#">OTAPolicy</a>	Customer inline	0	
<input type="checkbox"/> <a href="#">S3Policy</a>	Customer inline	0	

Slika 5.11: Popis dozvola uloge *OTARole*

Posao se stvori tako što se u sučelju odabere opcija za stvaranje FreeRTOS OTA posla te unese ime posla. Nadalje, odabiru se uređaji za ažuriranje kao i protokol kojim će se podaci slati. AWS istovremeno može slati ažuriranje putem MQTT i HTTP veze, a uređaj će poslati zahtjev u obliku u kojem podržava preuzimanje. Nadalje je

potrebno odabratи datoteku iz S3 pohrane, kao i certifikat kojim ће se datoteka potpisati. Potpisana datoteka, osim што se šalje odabranim protokolom na ciljne uređaje, automatski se pohranjuje u S3 spremnik. Ovisno o konfiguraciji ciljnog uređaja, potrebno je navesti i putanju gdje ће se datoteka pohraniti na uređaj. Zadnji korak prije kreiranja posla potrebno je odabratи hoće li posao biti kontinuirani ili jednokratni. Budуći da ne postoji grupa uređaja kojoj je moguћe dodijeliti kontinuirani posao, odabire se jednokratni. Također je moguћe postaviti i konfiguraciju postupnog uvođenja, bilo statičkom stopom definirajuћi maksimalan broj uređaja u minuti, ili eksponencijalnom stopom s faktorom povećanja. Dostupne su i postavke otkazivanja posla ukoliko su zadovoljeni kriteriji, poput broja uređaja koji su posao izvršili s porukom *rejected* ili *failed*. Slika 5.12 prikazuje kako se mijenjaju stanja OTA posla nakon kreiranja.

<a href="#">AFR_OTA-OTAJob_v1</a>	Snapshot	In progress - Rollout in progress
<a href="#">AFR_OTA-OTAJob_v1</a>	Snapshot	In progress - Rollout completed
<a href="#">AFR_OTA-OTAJob_v1</a>	Snapshot	<b>Completed</b>

**Slika 5.12:** Stanja pokrenutog jednokratnog OTA posla

# 6. Web aplikacija

Svi podaci koji su pohranjeni u bazi podataka trebaju biti korisnicima na raspolaganju jednostavno i intuitivno. U tu je svrhu kreirana aplikacija radi prikaza i vizualizacije podataka korisnicima u stvarnom vremenu. Web aplikacija treba biti javno dostupna na internetu kako bi korisnici razvijenog sustava mogli pregledavati podatke poslane s uređaja. Za objavu web aplikacije na internet (engl. *deployment*) kreirani su vlastiti resursi na kojima se izvršava aplikacija. Za infrastrukturu aplikacije korišteni su resursi platforme AWS koji pružaju iznajmljivanje virtualnih računala te se na njima mogu pokrenuti vlastite računalne aplikacije. Za web aplikaciju koja će korisnicima pružati uvid u podatke odabrana je Grafana koja nudi brojne funkcionalnosti za vizualizaciju podataka. Nadalje, kreiran je i alarmni sustav koji obavještava o promjenama stanje koja dolaze od uređaja.

## 6.1. Infrastruktura aplikacije

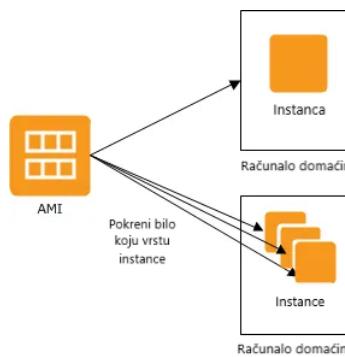
Temelj cijele infrastrukture aplikacije jest usluga Amazon EC2 (engl. *Elastic Compute Cloud*). Ova usluga pruža skalabilni računalni kapacitet na zahtjev u oblaku platforme AWS. Ova usluga omogućava pokretanje neograničenog broja virtualnih poslužitelja, ovisno o zahtjevima aplikacije i razvijanog sustava, konfiguriranje sigurnosti i umrežavanja te upravljanje pohranom. Usluga EC2 nudi razne prednosti [3]:

- fleksibilno skaliranje: omogućava brzo povećanje ili smanje kapaciteta u kombinaciji s uslugom EC2 Auto Scaling za automatsko skaliranje radi prilagodbe troškova,
- potpuna kontrola nad instancom: pruža pristup instanci kao i bilo kojem stroju, moguće pokretanje i zaustavljanje spajanjem na udaljeno računalo, ali i API pozivom,
- fleksibilne usluge objave u oblaku: nudi izbor između više vrsta instanci, operacijskih sustava i softverskih paketa te nudi konfiguraciju memorije, procesora

i pohrane,

- integracija s drugim uslugama: jednostavno se integrira s ostalim uslugama u sklopu platforme AWS,
- pouzdanost: nudi vrlo pouzdano okruženje u kojem se zamjenske instance mogu brzo pokrenuti, te je obaveza ugovora o razini usluge Amazon EC2 (engl. *Service Level Agreement - SLA*) dostupnost od 99,99% za svaku regiju,
- sigurnost: arhitektura nudi različite sigurnosne značajke i postavljanje virtualnih privatnih mreža za ograničen pristup resursima.

EC2 instance virtualni su poslužitelji koji rade na infrastrukturi računalstva u oblaku tvrtke Amazon. Jedan fizički AWS poslužitelj uslužuje više EC2 virtualnih poslužitelja koje pokreće hipervizor na računalu domaćinu. Ova usluga pružanja virtualne infrastrukture dio je skupa IaaS usluga koje AWS nudi. EC2 virtualni poslužitelji kopije su izvornog predloška na kojem se temelje. AMI (engl. *Amazon Machine Image*) temeljna je komponenta koja omogućava pokretanje virtualnih poslužitelja u AWS infrastruktu. Predstavlja virtualni stroj koji sadrži sve potrebne informacije za pokretanje instance, uključujući operacijski sustav, aplikacije te konfiguracijske podatke. To je priлагodljivi predložak za EC2 instance. Iz jednog AMI predloška, odnosno slike stroja, moguće je kreirati više istih EC2 instanci, što olakšava njihovo stvaranje i objavu na internet. Ovo je korisno za horizontalno skaliranje aplikacija. Slika 6.1 prikazuje način na koji funkcioniра AMI predložak - kreira instance bilo koje vrste te ih pokrene na domaćinskim računalima [4].



Slika 6.1: AMI predložak i distribucija slike stroja na instance [4]

AMI predlošci podržavaju dva tipa virtualizacije: hardversku virtualizaciju i paravirtualizaciju. Predložak s hardverskom virtualizacijom pruža mogućnost pokretanja operacijskog sustava izravno na vrhu virtualnog stroja bez ikakvih izmjena, kao da se pokreće na samom hardveru (engl. *bare-metal hardware*). Domaćinski sustav usluge

EC2 emulira dio ili sav temeljni hardver koji je predstavljen gostu odnosno AMI slici. Moguće je isto tako koristiti hardverska proširenja koji pružaju brzi pristup domaćinskom hardveru. Paravirtualizacija koristi poseban pokretački program koji lančano učitava jezgru u AMI sliku. Paravirtualni gosti mogu se pokretati na domaćinskom hardveru koji nema eksplisitnu podršku za virtualizaciju. Ovaj tip ne simulira hardver, nego pravi hiperpozive (engl. *hypercalls*) za izvršavanje osjetljivih CPU naredbi. Preporuča se odabir hardverske virtualizacije, iako generalno paravirtualizacija pruža bolje performanse, zbog poboljšanja hardverske virtualizacije i novih upravljačkih programa koji izjednačava rad obje vrste virtualizacije.

Pri kreiranju instance potrebno je definirati i njezinu vrstu. Vrsta koja se navede određuje hardver dostupan instanci. Svaka vrsta nudi različitu ravnotežu računalnih, memorijskih i mrežnih resursa. Vrste su grupirane u skupine na temelju potreba ciljnih aplikacija. AWS pruža sljedeće vrste instanci:

- opće namjene (engl. *general purpose*),
- optimiziranog računanja (engl. *compute optimized*),
- optimizirane memorije (engl. *memory optimized*),
- optimizirane pohrane (engl. *storage optimized*),
- ubrzanih računanja (engl. *accelerated computing*),
- računanja visokih performansi (engl. *high-performance computing*).

Vrstu instance potrebno je odabrati na temelju zahtjeva same aplikacije. Isto tako, pri kreiranju instance, uvijek je potrebno voditi računa o odabiru regije - optimalno je odabrati regiju kojoj je klijentski promet najbliži.

Za potrebe razvijenog sustava, kreirana je EC2 instanca opće namjene s najmanjim resursima budući da je klijentski promet jako malen, odnosno vrlo malo korisnika pristupa aplikaciji. Također, pridružena joj je instanca spremnika za pohranu veličine 8 GB. Operacijski sustav na kreiranoj instanci aplikacije jest Linux radi lakšeg spajanja na instancu i pokretanja web aplikacije putem komandne linije. Pri stvaranju instance kreirana je nova sigurnosna grupa koja omogućava spajanje na virtualno računalo putem SSH klijenta.

Instanci je automatski dodijeljena virtualna privatna mreža (engl. *Virtual Private Cloud - VPC*) koja predstavlja izolirani virtualni mrežni prostor unutar AWS infrastrukture. Omogućava potpunu kontrolu nad mrežnim okruženjem, uključujući izbor vlastitog IP adresnog prostora, konfiguraciju podmreža, postavljanje usmjerenja i pristupnih kontrola mreže. Svaka je virtualna privatna mreža podijeljena na manje di-

jelove mreže zvane podmreže. One mogu biti privatne ili javne te se automatski za svaku mrežu kreiraju tri podmreže. Svaka je kreirana podmreža prema zadanim postavkama privatna, te je za pristup instanci odnosno aplikaciji koja se nalazi za njom potreban javni pristup. Iako omogućavanje javnog pristupa znači da svatko može prisupiti IP adresi instance s interneta, unošenje IP adrese pri svakom pristupu aplikaciji nije prilagođeno korisniku, što narušava korisničko iskustvo korištenja aplikacije. Stoga je potrebno uvesti komponentu za balansiranje opterećenja (engl. *load balancer*). Balanseri preusmjeravaju korisničke zahtjeve na manje opterećene instance i tako ravnomjerno raspoređuju ulazni promet. Budući da je u razvijenom sustavu korištena samo jedna instance na kojoj se pokreće aplikacija, ovdje je primarna svrha balansera pružiti DNS naziv koji će biti javno dostupan korisnicima, i tako poboljšati iskustvo korištenja aplikacije.

Aplikacijskoj instanci potrebno je dodijeliti pravila za ulazne zahtjeve, odnosno omogućiti TCP komunikaciju s podmrežem u kojem se nalazi balanser. Tim se pravilom omogućuje prolaz zahtjeva s balansera prema aplikacijskoj instances. Isto tako, iz sigurnosnih razloga poželjno je koristiti protokol HTTPS za pristup balanseru. Za korištenje protokola HTTPS potrebno je učitati certifikat, no budući da domena na kojoj se nalazi balanser nije službeno registrirana, ne posjeduje niti službeno potpisani certifikat. Stoga je lokalno kreiran samopotpisani certifikat pomoću biblioteke *OpenSSL* i zatim učitan u oblak. Slika 6.2 prikazuje mrežnu povezanost balansera i instance na kojoj se nalazi aplikacija. Balanser sluša promet na vratima 443, što su vrata za protokol HTTPS. Dolazni zahtjevi prosljeđuju se skupini ciljnih instances, ovdje pod nazivom *GrafanaTargetGroup*, u kojoj se mogu nalaziti sve buduće stvorene instance na kojima se pokreće aplikacija. Nadalje, promet se prosljeđuje ciljnoj instance koja je dostupna na vratima 3000. Na tim je istim vratima pokrenuta aplikacija.



**Slika 6.2:** Mrežna povezanost balansera i aplikacijske instance

Kako bi se maksimalno izoliralo izvođenje aplikacije i tako ostavilo prostora za izvršavanje drugih procesa na virtualnom računalu, na instanci EC2 aplikacija je pokrenuta pomoću alata Docker radi kontejnerizacije aplikacije. Sljedeći programski isječak prikazuje naredbu za preuzimanje i pokretanje Docker slike za Grafanu. Isto tako, potrebno je izložiti vrata na kojih se pokreće aplikacija kako bi joj se moglo

pristupiti. Korištena je slika verzije otvorenog koda.

```
docker run -d -p 3000:3000 --name=grafana grafana/  
grafana-oss
```

**Isječak koda 6.1:** Pokretanje Docker slike za Grafanu

## 6.2. Grafana

Gotovo sigurno ću koristiti Grafanu budući da ima gotove vizualizacije koje su meni potrebne. Isto tako, malo opisati funkcionalnosti koje se nude u samoj Grafani, kako funkcionira alerting sustav. Napraviti nekoliko dashboarda, povezati s datasourceom, kreirati par alerta i pokazati kak su došli u obliku maila. Ponuditi proširenja za to - PagerDuty aplikacija recimo.

## **7. Zaključak**

Moj zaključak.

# LITERATURA

- [1] Aws regions and availability zones. 2024. URL <https://medium.com/my-experiments-with-aws/aws-regions-and-availability-zones-973b89f6f24c>.
- [2] LVGL, 2024. URL <https://lvgl.io/>.
- [3] AWS Documentation. Amazon.com, Inc., 2024. URL <https://docs.aws.amazon.com/index.html>.
- [4] Katharine Angelopoulos. Understanding cloud computing: The aws ec2 instance. 2022. URL <https://medium.com/@kangelopoulos/understanding-cloud-computing-the-aws-ec2-instance-e9e0b7d39040>.
- [5] Stephen J. Bigelow. What is edge computing? everything you need to know. 2021. URL <https://www.techtarget.com/searchdatacenter/definition/edge-computing>.
- [6] C. Bormann. CBOR, 2020. URL <https://cbor.io/>.
- [7] Marshall Brain i Talon Homer. How wifi works. 2021. URL <https://computer.howstuffworks.com/wireless-network.htm>.
- [8] American Bright. Major benefits of addressable rgb leds with integrated circuits. URL <https://www.americanbrightled.com/flexibility-control-and-customization-major-benefits-of-addressable-rgb-leds-with-integrated-circuits/>.
- [9] Programming Electronics. How to use spiffs for an esp32 file system. 2024. URL <https://www.programmingelectronics.com/spiffs-esp32/#what-is-spiffs>.

- [10] Espressif. *esp-aws-iot*, 2023. URL <https://github.com/espressif/esp-aws-iot/tree/release/202012.04-LTS>.
- [11] Espressif. *Provisioning API*, 2024. URL <https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32/api-reference/provisioning/provisioning.html>.
- [12] *ESP32-C3 Series Datasheet*. Espressif Systems, 2023. URL [https://www.espressif.com/sites/default/files/documentation/esp32-c3\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf).
- [13] *ESP32-C3-Mini 1 Datasheet*. Espressif Systems, 2023. URL [https://www.espressif.com/sites/default/files/documentation/esp32-c3-mini-1\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-c3-mini-1_datasheet_en.pdf).
- [14] *ESP-IDF Programming Guide*. Espressif Systems, 2023. URL <https://docs.espressif.com/projects/esp-idf/en/v5.0.2/esp32c3/index.html>.
- [15] FreeRTOS. *AWS IoT Fleet Provisioning*, 2023. URL <https://aws.github.io/Fleet-Provisioning-for-AWS-IoT-embedded-sdk/v1.1.0/>.
- [16] Google. *Protocol Buffers - Google's data interchange format*, 2024. URL <https://protobuf.dev/>.
- [17] IBM. Simple network time protocol. 2024. URL <https://www.ibm.com/docs/en/i/7.5?topic=services-simple-network-time-protocol>.
- [18] InfluxData. *Get started with InfluxDB v2*, 2024. URL <https://docs.influxdata.com/influxdb/v2>.
- [19] Akashdeep Kashyap. *What you need to know about pkcs11?* 2024. URL <https://www.encryptionconsulting.com/what-you-need-to-know-about-pkcs11/>.
- [20] Aris S. Linas L. *What is a cron job: Understanding cron syntax and how to configure cron jobs*. 2024. URL <https://www.hostinger.com/tutorials/cron-job>.

- [21] Microsoft. Wi-fi problems in your home. . URL <https://support.microsoft.com/hr-hr/windows/problemi-s-wi-fijem-i-raspored-va%C5%Alega-doma-e1ed42e7-a3c5-d1be-2abb-e8fad00ad32a>.
- [22] Microsoft. What is softap? . URL <https://answers.microsoft.com/en-us/windows/forum/all/what-is-softap-what-does-it-do-where-do-i-get-it/e9e0385b-ad1a-446f-8b75-7973326c2629>.
- [23] Mouser. *DHT11 Humidity and Temperature Sensor.* URL <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Datasheet-Translated-Version-1143054.pdf>.
- [24] *MQTT.* MQTT.org, 2022.
- [25] A. S. Gillis N. Barney. Amazon web services (aws). URL <https://www.techtarget.com/searchaws/definition/Amazon-Web-Services>.
- [26] Carol Sliwa Robert Sheldon. Non-volatile storage (nvs). 2021. URL <https://www.techtarget.com/searchstorage/definition/nonvolatile-storage>.
- [27] Aleksandar Stojanović. *Osvrt na NoSQL baze podataka.* 2016. URL <https://hrcak.srce.hr/file/283391>.
- [28] *IEEE 802.11 Wireless Local Area Networks.* The Working Group for WLAN Standards, 2023. URL <https://www.ieee802.org/11/>.
- [29] Random Nerd Tutorials. Esp32 useful wi-fi library functions. 2021. URL <https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/>.

## **Sustav za udaljeni nadzor u poljoprivredi temeljen na platformi ESP32-C3 i AWS-uslugama**

### **Sažetak**

Ovo je moj hrvatski sažetak.

**Ključne riječi:** IoT, ESP32-C3-DevKitM-1, AWS, MQTT, InfluxDB, Grafana, računarstvo u oblaku, pametna poljoprivreda

## **System For Remote Monitoring In Agriculture Based On ESP32-C3 Platform and AWS Services**

### **Abstract**

This is my english abstract.

**Keywords:** IoT, ESP32-C3-DevKitM-1, AWS, MQTT, InfluxDB, Grafana, cloud computing, smart agriculture