BAB II
TINJAUAN PUSTAKA

2.1. State of the Art

No.	Judul Penelitian	Peneliti & Tahun	Hasil	Gap Penelitian
1	On Private *Server* Implementations and Data Visualization for LoRaWAN	Ahmed & Annamali Jr. (2023)	Membangun private LoRaWAN *server* dan dash*board* monitoring real-time dengan Things Stack CE, Grafana, *InfluxDB*, Node-RED, *MQTT*, Raspberry Pi, Dragino LPS8, dan ESP32.	Secara langsung menerapkan implementasi privat *server* namun belum membahas kemampuan *gateway* multikanal dengan penyesuaian frekuensi plan yang diterapkan di Indonesia (AS923) dan Chirpstack
2	BYOG: Multi-Channel, Real-time LoRaWAN *Gateway* Testbed using General-Purpose Software Defined Radio	Shahid & Krishnaswamy (2024)	Mengembangkan testbed LoRaWAN berbasis SDR (RTL-SDR, HackRF, USRP B210) sebagai *gateway* 10 kanal simultan.	Fokus pada pengembangan *gateway* berbasis SDR dan layer fisik LoRa. Tidak membahas implementasi *Network server*, *MQTT*, Chirpstack, *Docker*, maupun evaluasi performa.

No.	Judul Penelitian	Peneliti & Tahun	Hasil	Gap Penelitian
3	Alat Ukur Sinyal LoRa untuk Mengetahui Jangkauan antara *End device* dengan *Gateway*	Pebrian dkk (2025)	Mengembangkan alat ukur sinyal LoRa berbasis SenseCAP K1100 dan *gateway* Dragino PG1301 dengan *server* Chirpstack, *MQTT*, dan OpenRemote pada SF7 vs SF12.	Penelitian belum membahas pembangunan infrastruktur LoRaWAN berbasis *server* privat menggunakan *Docker*, *gateway SenseCAP M2* dan menyesuaikan regulasi frekuensi di Indonesia.
4	Analisis LoRa dengan Teknologi LoRaWAN dalam Ruangan di Lingkungan Politeknik Negeri Malang	Noprianto dkk. (2024)	Membangun sistem monitoring IoT berbasis LoRaWAN dengan LILYGO TTGO LoRa32, *gateway* RAK2247 915 MHz, TTN, *MQTT*, Node-RED, *InfluxDB*, Grafana, dan *Docker*.	Penelitian masih menggunakan TTN sebagai cloud *network server* sehingga bergantung pada layanan pihak ketiga. Implementasi belum membangun infrastruktur LoRaWAN privat menggunakan Chirpstack, *gateway SenseCAP M2* dan menyesuaikan regulasi frekuensi di Indonesia.

No.	Judul Penelitian	Peneliti & Tahun	Hasil	Gap Penelitian
5	Demo: A LoRaWAN Emulator Testbed	Akanova, Urazayev, Kadirzhanov, dan Zorbas (2023)	Mengembangkan testbed emulator LoRaWAN berbasis ESP32 dan Raspberry Pi dengan *gateway* single-channel dan single-SF.	Penelitian terdahulu masih terbatas pada emulator dan belum mengimplementasi infrastruktur LoRaWAN menggunakan *gateway* komersial multikanal, *server* Chirpstack, serta pengujian performa pada frekuensi AS923 sesuai regulasi Indonesia.

2.2. *Internet of Things* (IoT)

Internet of Things (IoT) merupakan sebuah proses menggambarkan bagaimana berbagai objek fisik dapat dihubungkan dengan sistem logika digital, sehingga objek-objek tersebut mampu saling terhubung dan berkomunikasi melalui jaringan internet tanpa memerlukan campur tangan manusia secara langsung. Dilengkapi dengan sensor dan perangkat lunak, objek-objek tersebut dapat mengumpulkan serta saling bertukar data secara otomatis, membentuk sebuah ekosistem jaringan yang memungkinkan informasi tersampaikan ke berbagai lokasi maupun platform yang membutuhkannya (Dauda & Flauzac, 2024).

2.3. Teknologi LoRa dan LoRaWAN

2.3.1. Konsep Dasar LoRa (*Long Range*)

LoRa (*Long Range*) adalah sebuah teknik modulasi radio proprietari yang dirancang untuk mendukung komunikasi nirkabel jarak jauh dengan konsumsi daya rendah, sehingga sangat sesuai digunakan pada perangkat Internet of Things (IoT). Pertama kali diperkenalkan ke dunia dan

dipatenkan tahun 2014 oleh perusahaan Semtech (Musonda & Ndiaye, 2024), LoRa mendapat sambutan yang antusias dengan kemampuan berkomunikasi di radio frekuensi mampu bertukar informasi mencapai jarak yang cukup yaitu 3-7 km bergantung pada keadaan lingkungan yang LoS (*Line of Sight)* dan N-LoS *(Non Line of Sight)* (Fahmida, 2026).

LoRa bekerja pada spektrum frekuensi tanpa lisensi (unlicensed bands) dan menggunakan teknik modulasi *Chirp Spread Spectrum* (*CSS*) yang memberikan sensitivitas penerimaan tinggi serta tahan terhadap interferensi sinyal dalam pengiriman data dijarak yang cukup jauh. Teknik *spread spectrum* pada LoRa memanfaatkan modulasi frekuensi linear berbasis pulsa *chirp* dengan *bandwidth* yang tinggi untuk membawa informasi data (Sendra et al., 2020)(Hadiningrum et al., n.d.).

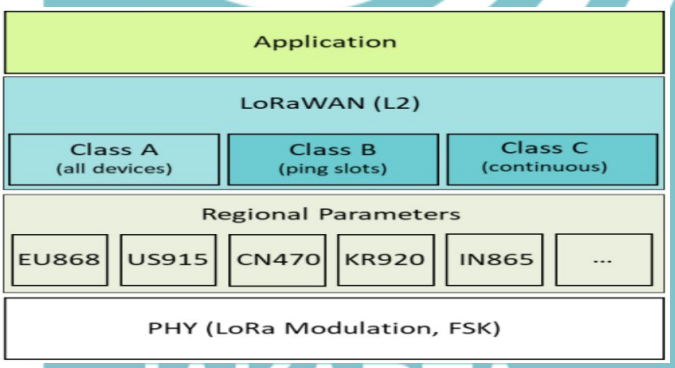
Teknologi ini berasal dari Semtech dan digunakan sebagai lapisan fisik (*physical layer*) dalam protokol komunikasi IoT kelas *Low Power Wide Area Network* (LPWAN) (Pebrian, 2025). *CSS* memungkinkan transmisi sinyal pada jarak yang jauh dengan risiko gangguan yang relatif rendah, sehingga perangkat dapat beroperasi dengan baterai dalam waktu yang cukup lama. Sinyal chirp dibentuk dari gelombang sinusoidal yang mengalami perubahan frekuensi secara bertahap, baik meningkat (*up-chirp*) maupun menurun (*down-chirp*) seiring waktu. Perubahan frekuensi tersebut digunakan untuk merepresentasikan simbol data dalam proses komunikasi (Judvaitis et al., 2025)(Sendra et al., 2020).

Berdasarkan Keputusan Menteri Komunikasi dan Informatika Nomor 1 Tahun 2019, alokasi frekuensi untuk teknologi LoRa di Indonesia ditetapkan pada rentang 920–923 MHz. Selain itu, pemanfaatan spektrum frekuensi tersebut diatur lebih lanjut melalui Peraturan Menteri Komunikasi dan Informatika (Permenkominfo) Nomor 2 Tahun 2023 tentang penggunaan spektrum frekuensi radio berdasarkan izin kelas (Hilyati, 2023) (Kominfo, 2023).

2.3.2. Protokol LoRaWAN

Protokol LoRaWAN (*Long Range Wide Area Network*) adalah sebuah standar komunikasi dan arsitektur jaringan yang dikembangkan oleh berbagai perusahaan, seperti IBM, Semtech, dan Actility, yang tergabung dalam organisasi LoRa Alliance. Melalui kolaborasi tersebut, LoRa Alliance merancang dan mengembangkan protokol komunikasi khusus untuk teknologi LoRa yang dikenal sebagai LoRaWAN (Rahmansyah & Murdiyat, n.d.).

LoRaWAN mengatur standar protokol MAC (*Media Access Control)* untuk *end node* dapat berkomunikasi dengan *gateway* dan informasinya bisa diterima oleh *server* serta menjaga kemanan proses routing data yang dapat dilihat pada Gambar 2.1. Pada lapisan MAC (*Medium Access Control*), transmisi setiap *end node* diatur untuk mencapai pengiriman data yang efisien.



Gambar 2.1 Arsitektur LoRaWAN

Sumber: (Ahmed, S. T., 2023)

Pada Gambar 2.1 menjelaskan bahwa LoRa bertanggung jawab sebagai modulasi fisik untuk mengirimkan sinyal radio, sedangkan LoRaWAN bertanggung jawab mengatur mekanisme akses kanal, prosedur join perangkat, enkripsi data, serta manajemen pengiriman ulang (*retransmission*).

2.3.3. Pita Frekuensi LoRa

Alokasi pita frekuensi ISM untuk LoRa berbeda-beda tergantung pada regulasi masing-masing wilayah. Beberapa pita frekuensi yang umum digunakan antara lain:

a. 433 MHz, pita frekuensi yang memiliki jangkauan sangat baik tetapi *bandwidth* yang dapat digunakan relatif cukup sempit membuat kapasitas data cukup terbatas.

b. 868 MHz, pita frekuensi yang paling utama untuk LoRaWAN bagi wilayah Eropa ini diatur oleh ETSI dengan batasan *duty cycle* mengurangi interferensi antarperangkat.

c. 915 MHz, pita frekuensi yang digunakan secara luas dengan mendukung pilihan kanal yang banyak dan menawarkan kapasitas jaringan yang lebih baik. Biasanya frekuensi ini banyak ditemukan penggunaannya pada wilayah Amerika Utara.

d. 923 MHz, pita frekuensi ini sudah mendapatkan regulasi lokal dari badan pemerintahan dan otoritas telekomunikasi nasional. Salah satunya di Indonesia, frekuensi 923 MHz telah ditetapkan sesuai Permenkominfo.

2.4. Arsitektur Jaringan LoRaWAN

Arsitektur jaringan LoRaWAN (*Long Range Wide Area Network*) merupakan standar protokol komunikasi nirkabel bertipe *Low Power Wide Area Network* (LPWAN) dirancang menggunakan topologi bintang bertingkat (*star-of-stars*) di mana perangkat akhir (*end devices*) berkomunikasi melalui *gateway* yang meneruskan data ke *server* pusat, sehingga arsitektur ini memisahkan fungsi radio dengan fungsi *backend* (*network server* dan *application server*) dalam lapisan yang berbeda.

Untuk setiap paket LoRa, beberapa *gateway* dapat secara bersamaan bertindak sebagai penerus (*forwarder*) ke *network server*, yang kemudian menyaring penerimaan data duplikat, melakukan pemeriksaan keamanan, menjadwalkan pengiriman *acknowledgment*, serta menyesuaikan konfigurasi

jaringan pada *end node* dan *gateway* apabila diperlukan. (Li & Cao, 2022)(Ahmed, 2023)(Sanubari et al., 2025).

a. *End node*

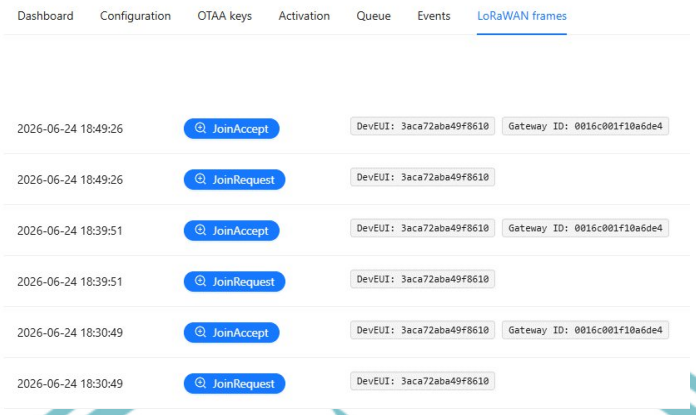
End node adalah perangkat IoT yang dilengkapi modul LoRa, sensor, aktuator, dan sumber daya baterai. Perangkat ini bertugas mengirimkan data sensor atau status aplikasi ke jaringan LoRaWAN melalui gelombang radio pada pita frekuensi ISM. *End node* biasanya bersifat low-power dengan kemampuan tidur (sleep mode) untuk mengoptimalkan masa pakai energi, serta hanya bertransmisi saat dibutuhkan (The Things Network).

b. *Gateway*

Gateway memegang peran sebagai penghubung antara *end node* dan *network server*. Dalam arsitektur LoRaWAN, *gateway* menerima paket radio dari *end node* tanpa melakukan pemrosesan data, kemudian meneruskannya ke *server* melalui koneksi IP (seperti Ethernet, Wi-Fi, atau seluler). Karena satu pesan dapat diterima oleh banyak *gateway*, arsitektur ini meningkatkan keandalan jaringan. (The Things Network)

c. *Network server*

Network server merupakan komponen backend yang menjadi titik pusat pengelolaan komunikasi dalam arsitektur LoRaWAN, termasuk pada implementasi Chirpstack yang digunakan dalam penelitian ini. Salah satu fungsi utamanya adalah memvalidasi keaslian (*authenticity*) perangkat akhir serta memastikan integritas pesan yang dikirimkan, yang menjadi bagian penting dalam proses autentikasi perangkat saat melakukan join ke jaringan melalui mekanisme OTAA yang dapat dilihat pada Gambar 2.2.



Gambar 2.2 Tampilan *Network server*

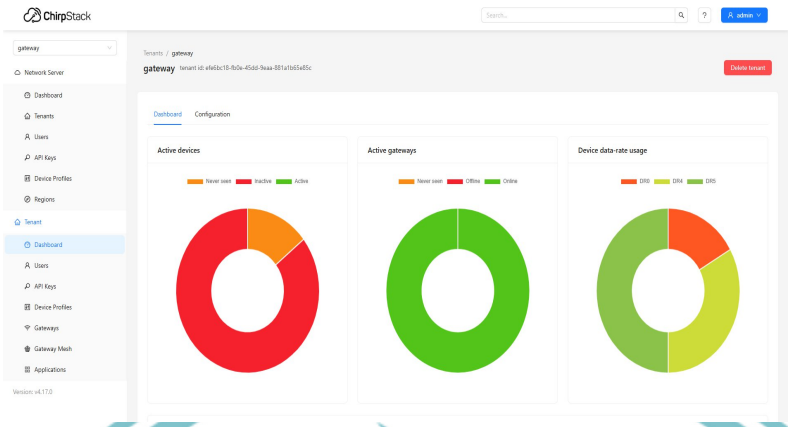
Sumber: Dokumentasi Pribadi

Gambar 2.2 menjelaskan Chirpstack berperan sebagai *network server* yang memverifikasi keberhasilan suatu perangkat bergabung ke jaringan serta menampilkan setiap data *uplink* yang diterima setelah proses join berhasil dilakukan melalui *payload Join Request* dan *Join Accept*

d. Application server

Application server bertanggung jawab untuk mengelola dan memproses data aplikasi akhir yang dikirim oleh perangkat IoT setelah difilter oleh *network server*. Di sini data didekripsi, diolah, disimpan, dan ditampilkan sesuai kebutuhan pengguna atau aplikasi, misalnya pada dash*board* monitoring atau sistem analitik. (The Things Network)

Dalam penelitian ini, Chirpstack berperan sebagai *Application server* yang menyediakan antarmuka berbasis web (Web UI) untuk menampilkan data secara real-time seperti pada Gambar 2.3.



Gambar 2.3 Tampilan *Application server*

Sumber: Dokumentasi Pribadi

Gambar 2.3 Menunjukkan tampilan dash*board* Chirpstack memudahkan proses analisis dan verifikasi keberhasilan autentikasi perangkat termasuk informasi waktu join (join time) dari setiap percobaan OTAA (*Active devices)*.

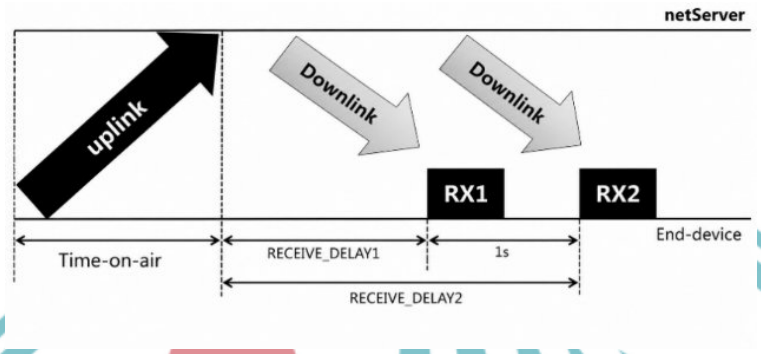
2.5. Kelas Perangkat LoRaWAN

Dalam spesifikasi protokol LoRaWAN, perangkat *end device* diklasifikasikan ke dalam tiga kelas operasional yang berbeda, yaitu Class A, Class B, dan Class C. Setiap kelas dirancang untuk memenuhi kebutuhan aplikasi yang berbeda dengan mempertimbangkan keseimbangan antara konsumsi daya, latensi komunikasi, dan kapabilitas penerimaan *downlink*. Pemilihan kelas perangkat yang tepat menjadi salah satu faktor penting dalam merancang sistem IoT berbasis LoRaWAN yang efisien. (LoRa Alliance, 2020)

2.5.1. Class A (Komunikasi Dua Arah dengan Konsumsi Daya Terendah)

Class A merupakan kelas perangkat paling dasar sekaligus wajib didukung oleh seluruh perangkat LoRaWAN. Perangkat Class A menggunakan mekanisme komunikasi yang sepenuhnya diinisiasi oleh *end device*, di mana perangkat hanya akan membuka jendela penerimaan *downlink* setelah menyelesaikan transmisi *uplink*. Prosesnya dimulai ketika perangkat mengirim data ke *gateway* (*uplink*). Setelah transmisi *uplink* selesai, perangkat membuka dua jendela penerimaan singkat yaitu RX1 (Receive Window 1) dan RX2 (Receive Window 2) dengan jeda waktu 1

detik dan 2 detik setelah akhir transmisi *uplink* (Al-Salim & Al-Rawi, 2020)(LoRa Alliance, 2020). Proses Receive RX1 dan RX2 dapat dilihat pada Gambar 2.4.

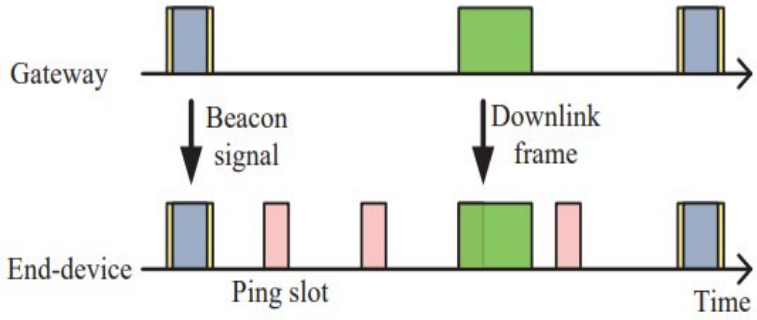


Gambar 2.4 Jendela Penerimaan Singkat (RX1 dan RX2)

Sumber: (Fehri et al., 2023)

2.5.2. Class B (Komunikasi Dua Arah dengan Jendela Penerimaan Terjadwal)

Class B merupakan pengembangan dari Class A yang menambahkan kemampuan penerimaan *downlink* terjadwal pada waktu-waktu tertentu yang telah disinkronisasi (LoRa Alliance, 2020). Perangkat Class B menerima sinyal beacon yang dikirimkan secara periodik oleh *gateway* setiap 128 detik untuk menyinkronkan jam internal perangkat dengan jaringan (Al-Salim & Al-Rawi, 2020). Mekanisme sinkronisasi berbasis beacon ini memungkinkan *network server* untuk mengetahui kapan perangkat aktif dan siap menerima data, sehingga latensi *downlink* dapat dikurangi secara signifikan dibandingkan Class A. Namun demikian, konsumsi daya perangkat Class B lebih tinggi daripada Class A karena perangkat harus tetap aktif untuk menerima sinyal beacon dan membuka ping slot secara berkala (Ron et al., 2020).



Gambar 2.5 Mekanisme *Downlink* Kelas B

Sumber: (Ron et al., 2020)

2.5.3. Class C (Komunikasi Dua Arah dengan Jendela Penerimaan Hampir Terus-Menerus)

Class C adalah kelas operasi dengan latensi *downlink* terendah di antara ketiga kelas LoRaWAN. Pada kelas ini, perangkat menjaga jendela penerimaan tetap terbuka hampir sepanjang waktu, kecuali ketika sedang melakukan transmisi *uplink*. Kondisi ini memungkinkan *network server* untuk mengirimkan data *downlink* kapan saja tanpa perlu menunggu jendela penerimaan yang terjadwal, sehingga latensi *downlink* pada Class C menjadi yang paling rendah di antara ketiga kelas perangkat LoRaWAN (LoRa Alliance, 2020).

2.5.4. Perbandingan Setiap Kelas Perangkat LoRaWAN

Ketiga kelas perangkat LoRaWAN memiliki karakteristik yang berbeda dalam hal konsumsi daya, latensi *downlink*, dan kasus penggunaan. Perbandingan ketiga kelas tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Perbandingan Tiga Kelas LoRaWAN

Sumber: (Seller, 2021)

Parameter	Class A	Class B	Class C
Jendela *Downlink*	2 Jendela setelah *uplink* (RX1 & RX2)	Terjadwal (ping slot) + RX1 & RX2	Memiliki kesamaan dengan Class A namun

Parameter	Class A	Class B	Class C
			dilakukan terus menerus
Latensi *Downlink*	Tinggi	Sedang	Rendah
Konsumsi Daya	Sangat Rendah	Sedang	Tinggi
Sumber Daya	Baterai	Baterai	Listrik
Dukungan Perangkat	Semua Perangkat	Opsional	Opsional

2.6. Parameter Jaringan LoRaWAN

LoRa (*Long Range*) merupakan teknik modulasi lapisan fisik yang dikembangkan oleh Semtech dan menggunakan metode *Chirp Spread Spectrum* (CSS). Berbeda dengan modulasi konvensional, *CSS* menyebarkan sinyal informasi ke seluruh *bandwidth* yang tersedia menggunakan sinyal chirp, yaitu sinyal yang frekuensinya meningkat (*up-chirp*) atau menurun (*down-chirp*) secara linier terhadap waktu (Sendra et al., 2020).

Karakteristik ini menjadikan LoRa tahan terhadap interferensi, efek multipath, dan kondisi kanal yang buruk, sehingga cocok digunakan pada lingkungan indoor maupun outdoor sangat dipengaruhi oleh empat parameter utama, yaitu *Spreading factor* (SF), *Bandwidth* (BW), *Coding rate* (CR), *Data rate* (DR), *Duty cycle* dan *Adaptive data rate* yang masing-masing memiliki karakteristik dan pengaruh berbeda terhadap kualitas transmisi data.

a. *Spreading Factor* (SF)

Spreading Factor (SF) merupakan parameter modulasi LoRa yang merepresentasikan jumlah bit yang dikodekan dalam setiap simbol transmisi. Nilai SF dapat dikonfigurasi dalam rentang SF7 hingga SF12, di mana setiap nilai menawarkan karakteristik yang berbeda. SF7 memberikan laju data yang tinggi namun dengan jangkauan yang lebih terbatas, sedangkan SF12 dirancang untuk komunikasi jarak jauh dengan sensitivitas penerima yang

lebih tinggi. Parameter SF secara langsung memengaruhi kecepatan transfer data dan time-on-air, di mana penggunaan nilai SF yang lebih tinggi terbukti dapat menekan tingkat *packet loss*, meskipun konsekuensinya adalah peningkatan durasi waktu pengiriman data. (Rahmansyah & Murdiyat, n.d.) (Pebrian, 2025)

Time-on-Air (ToA) merupakan durasi total yang dibutuhkan untuk mentransmisikan satu paket data secara penuh. ToA dipengaruhi secara langsung oleh nilai SF, BW, CR, dan ukuran *payload*. Hubungan antara SF dan ToA dapat dinyatakan melalui persamaan berikut:

$$T_{SIMBOL} = \frac{2^{SF}}{BW}$$

Maka, nilai Time On Air nya dapat dihitung dengan menggunakan rumus:

$$ToA = N_{Simbol}\ X\ T_{Simbol}$$

di mana T_{Simbol} adalah durasi satu simbol, BW adalah *bandwidth* yang digunakan, dan N_{Simbol} adalah jumlah total simbol dalam satu paket transmisi. Persamaan tersebut menunjukkan bahwa semakin tinggi nilai SF, maka durasi per simbol semakin panjang sehingga ToA keseluruhan paket semakin besar (Semtech, 2020).

Hubungan T_SIMBOL = 2^SF/BW dapat dilihat pada Tabel 2.2. Setiap peningkatan satu nilai SF akan melipatgandakan parameter 2^SF (Chips/Symbol). Pelipatgandaan ini menyebabkan durasi simbol (T_SIMBOL) ikut berlipat ganda, yang pada akhirnya memperpanjang Time-on-Air (ToA) total paket.

Tabel 2.2 Hubungan Rentang Setiap SF

Sumber: (https://heltec.org/project/WiFi-lora-32-v3/)

Spreading Factor (SF)	5	6	7	8	9	10	11	12
2^{SF}	32	64	128	256	512	1024	2048	4096
Nilai SNR (Signal-to-Noise Ratio) [dB]	-2.5	-5	-7.5	-10	-12.5	-15	-17.5	-20

Tabel 2.2 juga menunjukkan bahwa konsekuensi dari ToA yang lebih panjang adalah kemampuan demodulator LoRa untuk memproses sinyal dengan SNR (Signal-to-Noise Ratio) yang jauh lebih rendah, yaitu hingga -20 dB pada SF12, sehingga memperluas jangkauan komunikasi.

b. *Bandwidth* (BW)

Bandwidth (BW) pada jaringan LoRaWAN dapat dikonfigurasi pada tiga nilai, yaitu 125 kHz, 250 kHz, dan 500 kHz. *Bandwidth* yang lebih kecil menghasilkan laju data yang lebih rendah, namun memberikan ketahanan yang lebih baik terhadap derau (*noise*) sehingga sistem mampu beroperasi pada kondisi SNR yang lebih rendah (Pebrian, 2025)(Li & Cao, 2022).

Dalam penelitian ini, nilai BW yang digunakan adalah 125 kHz sesuai dengan konfigurasi standar LoRaWAN pada pita frekuensi AS923 yang berlaku di Indonesia berdasarkan Permenkominfo Nomor 2 Tahun 2023. Penggunaan BW 125 kHz dipilih karena memberikan keseimbangan optimal antara jangkauan komunikasi dan *data rate* pada lingkungan indoor NLOS (Kominfo, 2023). Selain BW 125 kHz, penelitian ini juga melakukan pengujian pembanding pada BW 250 kHz (DR6) dengan nilai SF yang sama (SF7), untuk mengevaluasi secara empiris pengaruh peningkatan *Bandwidth* terhadap Time on Air dan keandalan (*reliability*) proses Join OTAA.

c. *Coding Rate* (CR)

Coding Rate (CR) merupakan parameter yang menentukan tingkat koreksi kesalahan pada proses transmisi data melalui mekanisme *Forward Error Correction* (FEC). Pada mekanisme ini, setiap 4 bit data asli dikodekan bersama bit redundansi menjadi 5, 6, 7, atau 8 bit, yang menghasilkan nilai CR masing-masing sebesar 4/5, 4/6, 4/7, dan 4/8. Selama proses transmisi berlangsung, data yang dikirimkan rentan mengalami kerusakan akibat interferensi, seperti perubahan nilai bit dari 0 menjadi 1 atau sebaliknya (Wei & Transmission, 2021)(The Things Network).

d. *Data Rate* (DR)

Data Rate (DR) pada LoRaWAN merupakan kombinasi dari nilai SF, BW, dan CR yang secara bersama-sama menentukan kecepatan efektif pengiriman data. Dalam spesifikasi LoRaWAN, DR direpresentasikan dalam bentuk indeks numerik yang berbeda-beda tergantung pada regional parameters yang digunakan. Pada pita frekuensi AS923 yang berlaku di Indonesia, indeks DR dan konfigurasi parameter yang bersesuaian dapat dilihat pada Tabel 2.3 berikut.

Tabel 2.3 Indeks *Data rate* pada Frekuensi 923 MHz

Sumber: (https://heltec.org/project/WiFi-lora-32-v3/)

DR	SF	BW	*Data rate* (bps)
DR0	SF12	125	250
DR1	SF11	125	440
DR2	SF10	125	980
DR3	SF9	125	1.760
DR4	SF8	125	3.125
DR5	SF7	125	5.470
DR6	SF7	250	11.000

e. *Duty Cycle*

Pada jaringan LoRaWAN, *duty cycle* diterapkan untuk memastikan penggunaan spektrum frekuensi yang adil. Nilai *duty cycle* yang berlaku bervariasi tergantung pada regulasi wilayah, umumnya berkisar antara 0,1% hingga 10% per sub-band frekuensi. Dalam jaringan dengan kepadatan perangkat yang tinggi, pembatasan ini dapat menyebabkan kongesti dan peningkatan tingkat tabrakan paket (packet collision). Oleh karena itu, perancangan jaringan LoRaWAN harus mempertimbangkan *duty cycle* sebagai salah satu batasan fundamental.

Di Indonesia sendiri nilai persentase *duty cycle* untuk pita frekuensi 920-923 MHz sudah diatur pada Permenkominfo No. 2 tahun 2023 dimana maksimum nilainya 1% bagi proses *uplink* dan *downlink*. Nilai 1% jika diilustrasikan pada sebuah device yang bekerja mengirimkan data dalam 1 jam (3600 detik), perangkat hanya diperbolehkan mengirim selama maksimal 36 detik.

f. *Adaptive Data Rate* (ADR)

Adaptive Data Rate (ADR) merupakan mekanisme cerdas dalam protokol LoRaWAN yang dirancang untuk secara otomatis mengoptimalkan parameter transmisi (terutama SF dan daya pancar) berdasarkan kondisi kanal komunikasi. Algoritma ADR pada Chirpstack bekerja berdasarkan pengamatan terhadap nilai SNR tertinggi dari sejumlah paket *uplink* terakhir yang diterima (Khalilnasl et al., 2025)(Kufakunesu et al., 2023).

2.7. Mekanisme Aktivasi Perangkat LoRaWAN

Sebelum perangkat *end device* dapat berkomunikasi dan mengirimkan data dalam jaringan LoRaWAN, perangkat tersebut harus terlebih dahulu melalui proses aktivasi untuk memperoleh identitas jaringan dan kunci keamanan yang diperlukan. Spesifikasi LoRaWAN mendefinisikan dua mekanisme aktivasi yang dapat digunakan, yaitu *Over The Air Activation* (OTAA) dan *Activation by Personalization* (ABP). Kedua mekanisme ini memiliki karakteristik, prosedur,

dan tingkat keamanan yang berbeda, sehingga pemilihan metode aktivasi yang tepat perlu disesuaikan dengan kebutuhan dan skenario implementasi sistem.

2.7.1. *Over Time Air Activation* (OTAA)

Over-The-Air Activation (OTAA) merupakan mekanisme aktivasi yang direkomendasikan dalam spesifikasi LoRaWAN karena menawarkan tingkat keamanan yang lebih tinggi dibandingkan metode ABP. Sebelum perangkat bergabung ke dalam jaringan LoRaWAN terdapat tiga parameter identitas yang harus dikonfigurasi pada perangkat *end device* dan didaftarkan pada *network server* Chirpstack diantaranya DevEUI, AppEUI, dan AppKey.

a. DevEUI (*Device EUI*)

DevEUI adalah identifier unik 64-bit yang digunakan untuk mengenali setiap perangkat secara global di jaringan LoRaWAN. DevEUI bertindak sebagai identitas perangkat pada saat registrasi ke *server* dan menjadi dasar autentikasi awal untuk menunjukkan perangkat tujuan atau sumber (Theodorus, Visser, 2024).

b. AppEUI (*Application EUI*)

AppEUI adalah identifier unik untuk aplikasi yang terkait dengan perangkat tertentu. AppEUI membantu *server* mengelompokkan perangkat berdasarkan aplikasi atau domain layanan yang sama sehingga mempermudah manajemen dan routing data sesuai konteks aplikasi (Theodorus, Visser, 2024).

c. AppKey

AppKey adalah bentuk keamanan sesi yang digunakan antara perangkat dan *server* aplikasi LoRaWAN untuk menjamin bahwa hanya perangkat yang terdaftar dan terautentikasi bisa berkomunikasi dalam satu jaringan sama dan selama proses transmisi terjadi data akan terlindungi dari manipulasi (Theodorus, Visser, 2024).

2.7.2. *Activation By Personalization* (ABP)

Activation by Personalization (ABP) merupakan mekanisme aktivasi alternatif di mana seluruh parameter jaringan yang diperlukan, yaitu DevAddr, NwkSKey, dan AppSKey, dikonfigurasi secara manual dan disimpan secara permanen pada firmware perangkat *end device*. Parameter yang harus dikonfigurasi pada ABP meliputi DevAddr (alamat 32-bit perangkat dalam jaringan), NwkSKey (Network Session Key), dan AppSKey (Application Session Key).

a. DevAddr (*Device Address*)

DevAddr adalah sebuah pengenal yang dikonfigurasi secara statis untuk perangkat bergabung ke dalam jaringan LoRaWAN (Theodorus, Visser, 2024).

b. NwkSKey

NwkSKey adalah kunci yang digunakan dalam melakukan pengaman selama proses komunikasi berjalan. Fungsi dari NwsSKey berfungsi dalam mendekripsi dan enkripsi paket data yang dikirim dari perangkat ke *server* (Theodorus, Visser, 2024).

c. AppSKey

AppSKey adalah kunci yang digunakan dalam pengamanan selama proses komunikasi antara perangkat dan *server* di lapisan aplikasi (Theodorus, Visser, 2024).

2.8. Chirpstack

2.8.1. Pengertian dan Fungsi Chirpstack

Chirpstack adalah sebuah platform open-source LoRaWAN *Network server* yang digunakan untuk membangun dan mengelola jaringan LoRaWAN, baik secara privat (on-premise) maupun publik. Platform ini menyediakan antarmuka web untuk mengelola *gateway*, *end device*s, dan pengguna (tenants), serta mendukung integrasi dengan layanan eksternal

seperti basis data, *broker* pesan *MQTT*, dan layanan visualisasi data (Chirpstack).

Dalam arsitektur jaringan LoRaWAN, Chirpstack memiliki fungsi-fungsi yang mencakup menerima dan memproses data yang dikirim oleh *gateway*, mendaftar dan mengelola perangkat yang terhubung lalu meneruskan data ke sistem backend. Chirpstack memegang peran penting sebagai inti yang memfasilitasi jalur data dimulai dari *end node* menuju platform Chirpstack secara terstruktur dan efisien. (Chirpstack)

2.8.2. Komponen Chirpstack

Dalam platform Chirpstack, ada tiga komponen utama yang harus dipahami terkait peran dan fungsinya dalam jaringan LoRaWAN. Setiap komponen ini memiliki tugas berbeda tetapi saling bersinergi untuk menjamin kinerja jaringan secara keseluruhan.

a. Chirpstack *Gateway bridge*

Chirpstack *Gateway bridge* adalah komponen yang berfungsi sebagai jembatan antara *gateway* fisik dan *server* jaringan. Komponen ini menerima paket dari *gateway* LoRaWAN menggunakan *packet forwarder* standar seperti Semtech UDP atau BasicStation, kemudian mengubah format paket tersebut ke format *MQTT* (JSON/Protobuf) yang dapat dipahami oleh *network server* dan aplikasi *server*.

b. Chirpstack *Network server*

Chirpstack *Network server* merupakan komponen inti yang berperan dalam mengelola berbagai fungsi utama terkait pengaturan dan pemrosesan paket LoRaWAN. Komponen ini bertanggung jawab untuk menerima data *uplink* yang dikirimkan dari *gateway bridge*, menghapus duplikasi pesan yang mungkin diterima dari beberapa *gateway* sekaligus, serta mengatur penjadwalan *downlink* kepada perangkat. Selain itu, Chirpstack *Network server* juga berperan dalam mengelola mekanisme-mekanisme penting pada

jaringan, seperti *Adaptive data rate* (ADR) dan proses penanganan *Join Request* dari perangkat.

c. Chirpstack *Application server*

Chirpstack *Application server* bertindak sebagai lapisan atas dalam arsitektur Chirpstack yang bertanggung jawab atas pengelolaan serta penyampaian data aplikasi. Komponen ini menerima paket yang telah diproses oleh *network server*, kemudian mengekstrak dan menyimpan *payload* aplikasi tersebut. Selain itu, Chirpstack *Application server* juga menyediakan API serta integrasi untuk meneruskan data ke sistem eksternal, seperti dash*board* monitoring atau basis data, sekaligus menyajikan antarmuka bagi pengguna akhir untuk memantau perangkat dan aliran data yang berjalan.

2.9. Protokol Komunikasi LoRaWAN Berbasis Chirpstack

Dalam implementasi sistem Internet of Things (IoT) berbasis LoRaWAN, diperlukan protokol komunikasi pendukung untuk memastikan proses pertukaran data, pengelolaan layanan, serta integrasi antar komponen jaringan dapat berjalan secara efisien. Salah satu protokol utama yang digunakan adalah *MQTT* (Message Queuing Telemetry Transport), yang dirancang khusus untuk komunikasi data IoT dengan konsumsi *bandwidth* rendah dan latensi kecil. *MQTT* menerapkan mekanisme publish–subscribe melalui *broker* sehingga memungkinkan pertukaran data sensor secara asinkron dan skalabel. Pada platform Chirpstack, *MQTT* digunakan sebagai jalur komunikasi utama antara *gateway bridge*, *network server*, dan *application server* dalam meneruskan data *uplink* maupun perintah *downlink*. (Fikhri et al., 2025)

Selain *MQTT*, HTTP dengan pendekatan REST API digunakan sebagai antarmuka layanan untuk pengelolaan perangkat, konfigurasi jaringan, serta akses data oleh aplikasi eksternal. REST API memungkinkan sistem monitoring atau dash*board* mengambil data secara terstruktur dalam format standar seperti JSON. Seluruh data yang diterima melalui protokol *MQTT* dan REST API selanjutnya disimpan pada basis data (database) yang berfungsi sebagai media penyimpanan

terpusat bagi data sensor dan informasi jaringan. Integrasi antara *MQTT*, REST API, dan database dalam *server* LoRaWAN on-premise berbasis Chirpstack memungkinkan penyediaan layanan komunikasi data IoT yang terkelola, aman, dan mudah digunakan sebagai modul praktikum pembelajaran. (Syafa et al., 2026)

2.10. *MQTT Broker*

MQTT (*Message Queuing Telemetry Transport*) adalah protokol komunikasi berbasis publish-subscribe yang ringan dan efisien, dirancang untuk memenuhi kebutuhan komunikasi pada perangkat IoT dengan keterbatasan sumber daya. Arsitektur protokol *MQTT* mencakup sebuah *broker* sederhana yang menerima pesan yang dipublikasikan dan meneruskannya kepada subscriber yang terhubung.

Dalam infrastruktur Chirpstack, *MQTT broker* yang umumnya diimplementasikan menggunakan *Mosquitto* berperan sebagai jalur komunikasi antara Chirpstack *Gateway bridge* dan Chirpstack *Network server*. Pada implementasi LoRaWAN, komunikasi antara *gateway* dan *Network server* dilakukan menggunakan protokol *MQTT*, di mana *broker MQTT* berjalan pada mesin *Network server* dan bertindak sebagai perantara yang mengumpulkan pesan dari *gateway* serta mendistribusikannya kepada komponen yang berlangganan pada topik yang sesuai.

2.11. *Server*

Server merupakan perangkat keras atau perangkat lunak yang menyediakan layanan, sumber daya, atau data kepada perangkat lain yang disebut client melalui jaringan komputer. Jaringan komputer memungkinkan terjadinya komunikasi dan pertukaran data antar berbagai perangkat komputasi, termasuk *server*, *workstation*, komputer personal, dan perangkat mobile, dengan mengandalkan kombinasi perangkat keras dan perangkat lunak untuk membangun konektivitas. Berdasarkan fungsinya, server dalam arsitektur LoRaWAN dapat dibedakan menjadi beberapa jenis, di antaranya *Network Server*, *Application Server*, dan *Server On-Premise*.

2.11.1. *Network Server*

Network Server merupakan komponen inti dalam arsitektur LoRaWAN yang bertanggung jawab atas pengelolaan *gateway*, perangkat

akhir (*end device*), aplikasi, dan pengguna dalam keseluruhan jaringan LoRaWAN (The Things Network). Komponen ini bekerja pada lapisan jaringan dan menjadi penghubung utama antara *gateway* dengan *Application Server*. Adapun fungsi utama *Network Server* adalah sebagai berikut:

- Membangun koneksi aman menggunakan enkripsi AES 128-bit untuk menjamin keamanan *end to end* dalam pengiriman pesan antara perangkat akhir dan *Application Server*.
- Memvalidasi keaslian (*authenticity*) perangkat akhir serta memastikan integritas setiap pesan yang dikirimkan melalui jaringan.
- Melakukan penghapusan duplikasi (*deduplication*) terhadap pesan uplink yang diterima dari beberapa *gateway* secara bersamaan.
- Memilih *gateway* terbaik untuk melakukan perutean (*routing*) pesan downlink menuju perangkat akhir.
- Mengirimkan perintah ADR (*Adaptive Data Rate*) untuk mengoptimalkan laju data (*data rate*) perangkat secara otomatis.
- Melakukan pemeriksaan alamat perangkat (*device address checking*) serta memberikan tanda terima (*acknowledgement*) terhadap pesan uplink bertipe *confirmed*.
- Meneruskan *Join Request* dan *Join Accept* antara perangkat dan *join server* dalam proses autentikasi OTAA.
- Meneruskan *payload* aplikasi *uplink* ke *Application Server* yang sesuai serta merespons seluruh perintah pada lapisan MAC (*Medium Access Control*).

2.11.2. *Application Server*

Application Server merupakan komponen yang memproses pesan data spesifik aplikasi yang diterima dari perangkat akhir (The Things Network). Komponen ini bekerja pada lapisan yang lebih tinggi dari Network Server dan bertanggung jawab atas pengelolaan serta penyampaian data aplikasi secara menyeluruh. Selain menerima *data uplink*, *Application Server* juga bertugas menghasilkan seluruh *payload downlink* pada lapisan aplikasi dan mengirimkannya kembali ke perangkat akhir melalui *Network*

Server. Dalam satu jaringan LoRaWAN, dimungkinkan terdapat lebih dari satu *Application Server* yang beroperasi secara bersamaan sesuai kebutuhan sistem. Adapun fungsi utama *Application Server* adalah sebagai berikut:

- Mendekripsi *payload* aplikasi menggunakan kunci sesi yang telah dinegosiasikan saat proses OTAA (*Over The Air Activation*), kemudian menyimpan data tersebut sesuai kebutuhan sistem.
- Menghasilkan dan mengirimkan *payload downlink* lapisan aplikasi kepada perangkat akhir yang terhubung melalui *Network Server*.
- Menyediakan antarmuka berbasis web untuk memantau perangkat dan menampilkan data secara *real time*, termasuk mendukung integrasi dengan basis data seperti *PostgreSQL* untuk penyimpanan data historis.
- Mendukung integrasi dengan platform pihak ketiga melalui *REST API* dan *MQTT* untuk keperluan visualisasi maupun analisis data lebih lanjut.

2.11.3. *Server On-Premise*

Server on-premise adalah server yang diinstal, dikelola, dan dioperasikan secara fisik di lokasi pengguna, yaitu di dalam gedung atau laboratorium, berbeda dengan server cloud yang dihosting oleh pihak ketiga. Adapun karakteristik utama server on-premise adalah sebagai berikut:

- Memberikan kontrol penuh terhadap konfigurasi jaringan, pengelolaan perangkat, serta keamanan dan privasi data.
- Seluruh komponen layanan, mulai dari konfigurasi gateway, pengelolaan perangkat akhir, hingga penyimpanan data, dikelola secara langsung oleh pengelola jaringan tanpa perlu mengirimkan data ke infrastruktur eksternal.
- Memberikan fleksibilitas tinggi dalam melakukan eksperimen dan pengujian berbagai skenario jaringan, sehingga sangat sesuai untuk lingkungan akademik dan penelitian.

2.11.4. Perbedaan *Network Server*, *Application Server* dan *Server On-Premise*

Berikut ini disajikan sebuah tabel untuk mengetahui perbedaan antara ketiga jenis server tersebut:

Tabel 2.4 Perbedaan Ketiga Server

Sumber: Dokumentasi Pribadi

Aspek	*Network Server*	*Application Server*	*Server On-Premise*
Fungsi	Mengelola Komunikasi Lapisan Jaringan	Mengelola dan Memproses Data Aplikasi	Infrastruktur Tempat Server Dijalankan
Lapisan	MAC (*Medium Access Control*)	Lapisan Aplikasi	Lapisan Fisik / Infrastruktur
Output	Paket data yang telah divalidasi	Data yang telah dienskripsi	Lingkungan Operasional untuk Menjalankan Layanan
Contoh Implementasi	Chirpstack *Network Server*	Chirpstack *Application Server*	IBM System x3650 M4 dengan Proxmox

Dalam penelitian ini, Network Server dan Application Server diimplementasikan menggunakan platform ChirpStack, sedangkan pendekatan on-premise diterapkan dengan menjalankan seluruh komponen tersebut pada server fisik IBM System x3650 M4 yang mengoperasikan platform virtualisasi Proxmox.

2.12. Virtualisasi (*Virtual Machine*)

Virtualisasi adalah teknologi yang memungkinkan pembuatan representasi virtual dari sumber daya fisik seperti *server*, sistem operasi, perangkat penyimpanan, atau jaringan. Komponen inti dari virtualisasi adalah *hypervisor* atau *Virtual Machine Monitor* (VMM), yaitu lapisan perangkat lunak yang bertugas membuat dan mengelola satu atau lebih *virtual machine* (VM) di atas perangkat keras fisik (Ariyanto, 2023). *Hypervisor* bertugas mempresentasikan platform operasi virtual kepada sistem operasi tamu (guest OS) dan mengelola eksekusi sistem operasi tamu tersebut, termasuk pengelolaan prosesor, memori, dan sumber daya lainnya (Moratelli et al., n.d.)(Ariyanto, 2023).

Dalam penelitian ini, virtualisasi dipilih karena tiga alasan utama. Pertama, virtualisasi mengisolasi *server* Chirpstack dari perangkat keras fisik, sehingga kegagalan hardware tidak langsung menghilangkan data konfigurasi. Kedua, proses backup, restore, dan migrasi *server* menjadi lebih mudah. Ketiga, alokasi sumber daya (CPU, RAM, penyimpanan) dapat disesuaikan secara dinamis dengan kebutuhan beban jaringan LoRaWAN. Virtualisasi diimplementasikan menggunakan *Proxmox VE* yang menjalankan sebuah VM berbasis Ubuntu 22.04 LTS sebagai lingkungan operasional Chirpstack.

2.13. *Proxmox Virtual Environment*

Proxmox Virtual Environment (*Proxmox VE*) adalah platform manajemen virtualisasi *server open-source* berbasis QEMU/KVM dan LXC yang dikembangkan oleh Proxmox *Server* Solutions GmbH. Platform ini memungkinkan pengelolaan *virtual machine* (VM), kontainer, penyimpanan, serta jaringan melalui antarmuka web terintegrasi maupun CLI (Idrus, 2021)(Ariyanto, 2023).

Dalam penelitian ini, *Proxmox VE* dipilih karena dua alasan utama. Pertama, *Proxmox VE* mampu mendistribusikan beban kerja secara efisien antar beberapa VM, sehingga cocok untuk infrastruktur *server* dengan kebutuhan layanan yang beragam. Kedua, kemudahan pengelolaan melalui antarmuka web memungkinkan administrator memonitor dan mengatur sumber daya *server* (CPU, RAM, penyimpanan) secara terpusat tanpa perlu menguasai perintah CLI secara mendalam. Penelitian ini menggunakan *Proxmox VE* sebagai *hypervisor* yang

menjalankan VM Ubuntu 22.04.5 LTS sebagai lingkungan operasional Chirpstack untuk *server* LoRaWAN *on-premise*.

2.14. *Docker*

Docker merupakan platform open-source yang memungkinkan pengembang untuk mengemas, mendistribusikan, serta menjalankan aplikasi di dalam unit terisolasi yang disebut *container*. Kemampuan *Docker* dalam menyediakan proses deployment yang konsisten dan portabel di berbagai lingkungan menjadikannya solusi yang relevan untuk mengatasi kompleksitas pengelolaan perangkat yang heterogen dalam ekosistem IoT berskala besar. Dalam infrastruktur Chirpstack pada penelitian ini, seluruh komponen layanan, termasuk Chirpstack *Network server*, *PostgreSQL*, *Redis*, *InfluxDB*, dan *Mosquitto*, dijalankan dalam bentuk *container* yang dikelola menggunakan *Docker* Compose.

2.15. *Database*

Database atau basis data merupakan kumpulan data yang terorganisasi secara sistematis dan disimpan secara elektronis dalam suatu sistem komputer, sehingga dapat diakses, dikelola, dan diperbarui dengan mudah. Basis data yang umum digunakan dalam infrastruktur *server* LoRaWAN bergantung pada analisis kebutuhan fungsional dan non-fungsional dari sistem yang dibangun, dengan mempertimbangkan performa, skalabilitas, serta jenis data yang dikelola (Wolters & Riehle, 2023). Pada platform Chirpstack, kebutuhan yang beragam ini dijawab dengan menggabungkan tiga jenis basis data sekaligus, masing-masing menangani jenis data yang berbeda: *PostgreSQL* untuk data konfigurasi yang harus tersimpan secara permanen, *Redis* untuk data sesi yang sifatnya sementara namun harus diakses dengan sangat cepat, dan *InfluxDB* untuk data sensor yang terus mengalir seiring waktu.

2.15.1. *PostgreSQL*

PostgreSQL merupakan sistem manajemen basis data relasional (*Relational Database Management System*/RDBMS) yang bersifat *open-source* dan menggunakan bahasa *Structured Query Language* (SQL) sebagai antarmuka utama dalam pengelolaan data. Dalam ekosistem Chirpstack, *PostgreSQL* berperan sebagai basis data utama yang

menyimpan seluruh data konfigurasi dan operasional jaringan LoRaWAN, meliputi informasi tenant, aplikasi, *gateway*, perangkat *end node*, profil perangkat, serta kunci keamanan jaringan (Jeyaraj et al., n.d.)(Urnikien, 2026).

2.15.2. *Redis*

Redis (*Remote Dictionary Server*) adalah sistem penyimpanan struktur data in-memory yang bersifat *open-source* dan dapat difungsikan sebagai basis data, cache, maupun message *broker*. Dalam infrastruktur Chirpstack, *Redis* digunakan untuk menyimpan data sesi perangkat yang bersifat sementara namun memerlukan akses cepat, seperti sesi OTAA (*Over The Air Activation*), antrian *downlink*, serta informasi *frame counter* perangkat (Kanthed, 2023)(Privalov & Stupina, 2024).

2.15.3. *InfluxDB*

InfluxDB merupakan sistem basis data time-series (*Time-Series Database*/TSDB) yang dikembangkan oleh InfluxData dan dirancang secara khusus untuk menyimpan serta menganalisis data yang menjadikan atribut waktu (*timestamp*) sebagai komponen utamanya. Dalam ekosistem Chirpstack, *InfluxDB* diintegrasikan sebagai tujuan penyimpanan data telemetri yang dikirimkan oleh *end node*, sehingga data sensor dapat tersimpan secara kronologis dan selanjutnya divisualisasikan menggunakan platform seperti Node-RED. (Hendriks et al., 2024).

2.16. *SenseCAP M2 Multi-Platform* LoRaWAN *Gateway*

Gambar 2.6 *SenseCAP M2* LoRaWAN *Gateway*

Sumber: (https://wiki.seeedstudio.com/Network/SenseCAP_Network /SenseCAP_M2_Multi_Platform/SenseCAP_M2_Multi_Platform_Overview/)

SenseCAP M2 Multi-Platform LoRaWAN *Gateway* merupakan perangkat *gateway* LoRaWAN yang bersifat terbuka (*open source*) dan dapat dikustomisasi secara fleksibel. Perangkat ini mendukung koneksi ke berbagai *network server* serta kompatibel dengan berbagai rencana frekuensi LoRaWAN global pada rentang 865 MHz hingga 923 MHz.

Gateway ini dapat digunakan dalam berbagai aplikasi LoRaWAN, seperti smart building, sistem pemantauan lingkungan, dan pertanian presisi. Selain itu, perangkat ini memiliki cakupan komunikasi yang luas serta kemampuan transmisi sinyal yang kuat, sehingga cocok digunakan sebagai infrastruktur dalam pembangunan jaringan LoRaWAN (Seeed Studio, n.d.).

Tabel 2.5 Spesifikasi *SenseCAP M2 Multi-Platform Gateway* LoRaWAN

Sumber: (https://wiki.seeedstudio.com/Network/SenseCAP_Network /SenseCAP_M2_Multi_Platform/SenseCAP_M2_Multi_Platform_Overview/)

Spesifikasi	**Deskripsi**
Prosessor	MT7628(MIPS24KEc@580MHz)
RAM	DDR2 128 MB
Flash	32 MB
Long Range Gateway Chip	SX1302
Channel	8
LoRaWAN Protocol	V1.0.3 Kelas A/B/C
Frequency Plan	IN865/EU868/RU864/US915/AU915/KR920/AS923
Long Range Sensitivity	-125dBm @125KHz/SF7 -139dBm @125KHz/SF12
Long Range TX Power	Up to 26 dBm
Antenna	Long Range: 3 dBi External Antenna with Base Wi-Fi: Internal Antenna
Antenna Impedance	50 Ohm

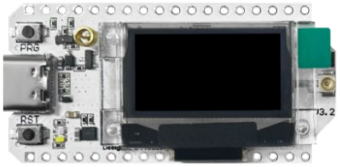
Berdasarkan Tabel 2.4, dapat diketahui perangkat *gateway* dilengkapi dengan prosesor MT7628 berbasis arsitektur MIPS24KEc dengan kecepatan 580 MHz, yang didukung oleh memori RAM DDR2 sebesar 128 MB dan penyimpanan flash sebesar 32 MB. Spesifikasi ini cukup memadai untuk menjalankan fungsi *gateway* dalam mengelola komunikasi data dengan tingkat sensitivitas penerimaan hingga -139 dBm pada konfigurasi tertentu, serta daya pancar maksimum mencapai 26 dBm. Hal ini memungkinkan perangkat untuk mendukung komunikasi jarak jauh dengan konsumsi daya yang relatif rendah, sesuai dengan karakteristik teknologi LoRaWAN.

2.17. *IBM System x3650 M4*

IBM System x3650 M4 merupakan *server* fisik kelas enterprise berformat rack-mounted 2U yang dikembangkan oleh IBM untuk memenuhi kebutuhan komputasi intensif pada lingkungan infrastruktur bisnis kritis maupun berbasis virtualisasi, dengan dukungan prosesor Intel Xeon low-voltage berdaya komputasi tinggi per watt, memori DDR3 1,35V yang lebih efisien hingga 19% dibandingkan DDR3 1,5V, serta teknologi *Calibrated Vectored Cooling* dengan ventilasi heksagonal untuk manajemen termal yang optimal, ditambah skalabilitas dari sisi konfigurasi multi-core, kapasitas memori besar, dan penyimpanan yang dapat dikembangkan sesuai kebutuhan, serta dilengkapi IBM Director sebagai solusi pemantauan sistem terintegrasi yang memungkinkan administrator memantau kondisi komponen kritis seperti prosesor, memori, dan hard disk drive secara remote dan real-time.

2.18. *Heltec WiFi LoRa 32 V3*

Heltec WiFi LoRa 32 V3 merupakan modul pengembangan (*development board*) berbasis Internet of Things yang dirancang dan diproduksi oleh Heltec Automation. Perangkat ini pertama kali diluncurkan pada tahun 2017 dan telah banyak digunakan oleh para pengembang serta praktisi IoT, didukung dengan pin yang lengkap serta layar OLED berukuran 0,96 inci yang terintegrasi langsung pada *board* yang terlihat pada Gambar 2.3 (Rev & Automation, 2022).



Gambar 2.7 *Heltec WiFi LoRa 32 V3*

Sumber: (https://heltec.org/project/WiFi-lora-32-v3/)

Heltec WiFi LoRa 32 V3 mendukung lingkungan pengembangan Arduino, serta telah menyediakan pustaka khusus bagi protokol LoRaWAN yang dapat berkomunikasi dengan *gateway* LoRa yang menjalankan protokol LoRaWAN standar, sehingga memudahkan proses integrasi perangkat ke dalam jaringan LoRaWAN seperti penelitian yang menggunakan infrastruktur Chirpstack sebagai *network server*.

Tabel 2.6 merangkum spesifikasi umum dari perangkat *Heltec WiFi LoRa 32 V3*.

Tabel 2.6 Spesifikasi *Heltec WiFi LoRa 32 V3*

Sumber: (https://heltec.org/project/WiFi-lora-32-v3/)

Spesifikasi	**Deskripsi**
MCU	ESP32-S3FN8 (dual-core 32-bit, hingga 240 MHz)
Chipset LoRa	SX1262
Frekuensi LoRa	470–510 MHz dan 863–928 MHz
Daya Tx Maks.	21 ± 1 dBm
Sensitivitas Rx Maks.	-137 dBm
Konektivitas nirkabel	Wi-Fi 802.11 b/g/n, Bluetooth 5
Memori	8MB Flash / 512KB SRAM

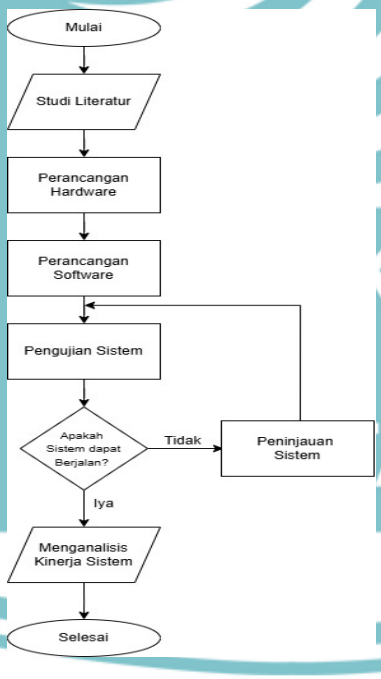
Tabel 2.6 memaparkan spesifikasi teknis dari perangkat *Heltec WiFi LoRa 32 V3* yang digunakan sebagai end-device dalam penelitian ini. Perangkat ini

mengandalkan mikrokontroler ESP32-S3FN8 berarsitektur dual-core 32-bit dengan kecepatan clock hingga 240 MHz sebagai unit pemrosesan utama, serta dilengkapi chipset LoRa SX1262 yang mampu beroperasi pada dua rentang frekuensi, yaitu 470–510 MHz dan 863–928 MHz, sehingga dapat disesuaikan dengan regulasi frekuensi LPWAN di berbagai negara, termasuk band AS923 yang digunakan dalam penelitian ini.

BAB III
PERENCANAAN DAN REALISASI

Pada bab perencanaan dan realisasi ini akan menjelaskan serta membahas mengenai tahapan perancangan dan realisasi pembangunan layanan *server* LoRaWAN On-Premise berbasis Chirpstack. Pembahasan mencakup beberapa aspek penting diantaranya deskripsi sistem, cara kerja sistem, spesifikasi sistem dan diagram blok.

Penelitian ini dilaksanakan melalui beberapa tahapan yang disusun secara sistematis, dimulai dari studi literatur, perancangan hardware, perancangan software, pengujian sistem, hingga analisis kinerja sistem. Tahapan tersebut digambarkan dalam bentuk diagram alur (flowchart) penelitian seperti yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Flowchart Penelitian

Berdasarkan Gambar 3.1, penelitian dimulai dengan tahap studi literatur untuk mengumpulkan referensi serta dasar teori yang relevan dengan topik penelitian. Tahapan tersebut kemudian dilanjutkan dengan perancangan hardware, yaitu proses penyiapan dan konfigurasi perangkat fisik yang digunakan, serta

perancangan software, yang meliputi proses instalasi dan konfigurasi sistem perangkat lunak yang dibutuhkan.

Setelah proses perancangan selesai, dilakukan pengujian sistem untuk memastikan seluruh komponen dapat berjalan sesuai dengan yang diharapkan. Apabila hasil pengujian menunjukkan bahwa sistem belum dapat berjalan dengan baik, maka dilakukan peninjauan sistem (review) untuk mengidentifikasi serta memperbaiki kendala yang ditemukan, sebelum dilakukan pengujian ulang. Proses ini berlangsung secara berulang (iteratif) hingga sistem dapat berfungsi sesuai dengan yang direncanakan. Apabila sistem telah berjalan dengan baik, penelitian dilanjutkan ke tahap analisis kinerja sistem untuk mengevaluasi performa keseluruhan sistem yang telah dibangun.

3.1. Rancangan Sistem

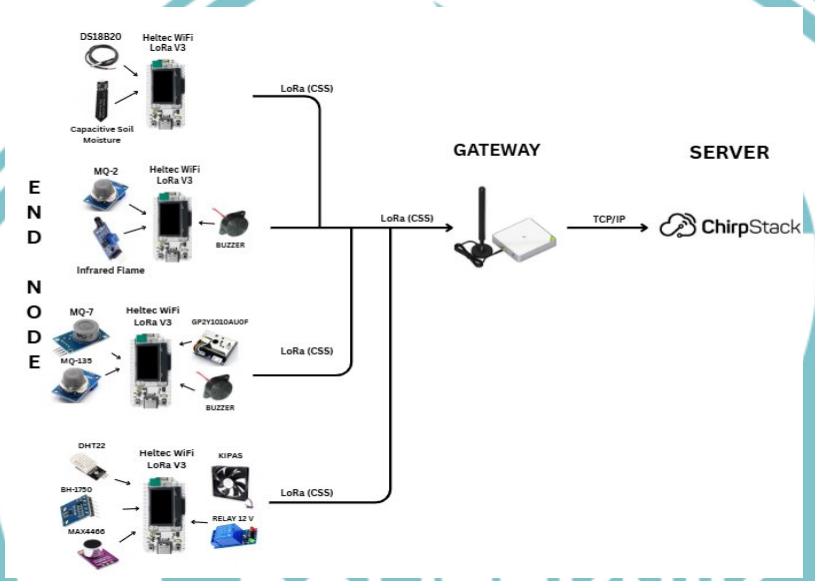
Pada penelitian ini dirancang sebuah sistem berbasis teknologi LoRaWAN yang berfokus pada pembangunan layanan *server* LoRaWAN secara on-premise sebagai modul praktikum. Sistem ini diharapkan dapat membantu mahasiswa untuk lebih mudah memahami bagaimana arsitektur LoRaWAN bekerja, bagaimana cara mengkonfigurasinya, serta bagaimana data dapat berpindah dari perangkat *end node* hingga sampai ke *server*, semua dilakukan secara langsung pada jaringan lokal.

Dalam perancangannya, sistem dibangun di atas sebuah *server* fisik yang dikonfigurasi menggunakan Proxmox sebagai *hypervisor*, kemudian di dalamnya dibangun sebuah *virtual machine* sebagai tempat Chirpstack diinstalasi dan dijalankan. Agar Chirpstack dapat berjalan dengan baik, sistem ini juga dilengkapi dengan beberapa layanan pendukung yaitu *Mosquitto* sebagai *MQTT broker*, *PostgreSQL* untuk menyimpan data, dan *Redis* untuk keperluan penyimpanan sesi sementara. Selain *server*, dirancang pula konfigurasi *gateway* yang bertugas menjembatani komunikasi antara *end node* dengan *server*, serta beberapa *end node* berbasis mikrokontroler yang dilengkapi sensor sebagai perangkat pengirim data. Sistem dinyatakan berhasil apabila *server* Chirpstack dapat berjalan secara stabil dan dapat dibuka melalui antarmuka web menggunakan alamat IP pada jaringan

lokal, serta *gateway* dan *end node* berhasil terhubung dan dapat berkomunikasi dengan *server*.

3.1.1. Deskripsi Sistem

Sistem yang dirancang merupakan sebuah arsitektur sistem berbasis LoRaWAN yang memiliki tiga komponen utama, yaitu *end node*, *gateway*, dan *network server* yang dapat dilihat pada Gambar 3.2.



Gambar 3.2 Arsitektur Sistem

Sumber: Dokumentasi Pribadi

Berdasarkan Gambar 3.2 yang menunjukkan keterkaitan setiap bagiannya, ketiga komponen arsitektur saling terhubung dalam satu jaringan lokal, di mana masing-masing komponen memiliki peran tersendiri agar data dari perangkat IoT dapat sampai ke *server* dengan baik. *Server* yang dibangun secara on-premise menggunakan Proxmox sebagai *hypervisor*, dengan Chirpstack berjalan di dalamnya melalui sebuah *virtual machine*. Agar Chirpstack dapat bekerja dengan optimal, sistem ini juga dilengkapi beberapa layanan pendukung, yaitu *Mosquitto* sebagai *MQTT broker* untuk menangani lalu lintas pesan, *PostgreSQL* untuk menyimpan data, serta *Redis* untuk kebutuhan penyimpanan sementara. selanjutnya *gateway* berperan sebagai jembatan komunikasi yang menerima paket data dari

end node melalui sinyal LoRa, lalu meneruskannya ke *server* melalui protokol *MQTT*.

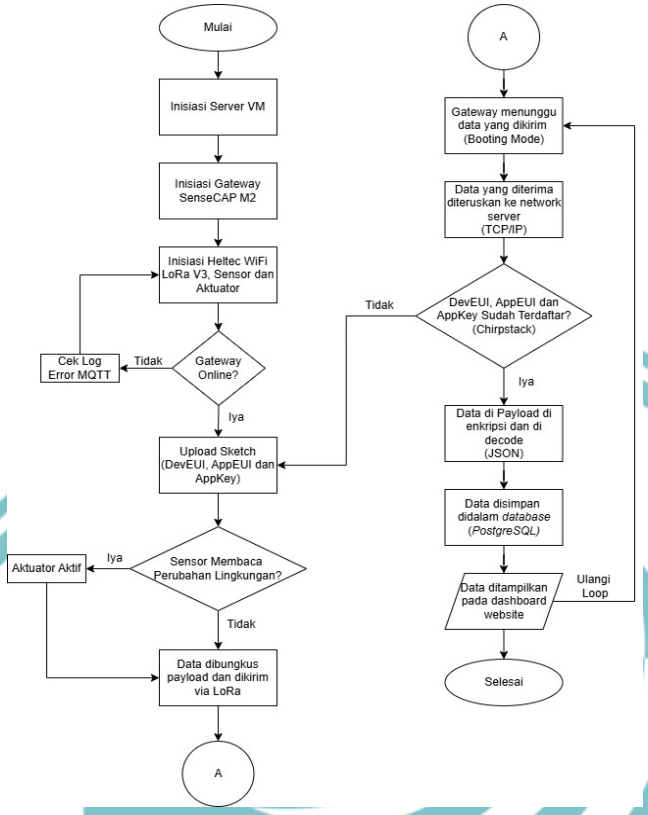
Di sisi *end node*, setiap perangkat dikonfigurasi dengan identitas jaringan yang unik dan akan terhubung ke *server* melalui mekanisme *Over-The-Air Activation* (OTAA). Setelah berhasil bergabung ke jaringan, *end node* akan secara berkala membaca dan mengirimkan data sensor dalam bentuk paket *uplink* yang kemudian diteruskan oleh *gateway* ke *network server*. Untuk mengetahui keberhasilan dari Chirpstack apabila *server* Chirpstack sudah dapat dibuka dan diakses melalui antarmuka web di jaringan lokal, serta *gateway* dan *end node* sudah terdaftar dan dapat berkomunikasi dengan *server*.

3.1.2. Cara Kerja Sistem

Sistem yang dirancang bekerja melalui integrasi antara *server*, *gateway*, dan *end node* dalam sebuah arsitektur LoRaWAN dapat dilihat pada Gambar 3.3.

1. Pada tahap pertama, *Server* VM menjalankan layanan Chirpstack yang telah dikonfigurasi sebelumnya menggunakan *Docker*, mencakup komponen Chirpstack *Network server*, Chirpstack *Gateway bridge*, *Mosquitto MQTT Broker*, *PostgreSQL*, *Redis*, dan *InfluxDB* yang telah disesuaikan dengan kebutuhan infrastruktur jaringan LoRaWAN yang dirancang.
2. Tahap kedua adalah menginisiasi dan mengkonfigurasi *gateway SenseCAP M2* untuk memeriksa status koneksi agar terhubung ke Chirpstack *Gateway bridge* melalui protokol *MQTT*.
3. Tahap berikutnya mengunggah sketch program ke perangkat Heltec LoRa WiFi V3 dengan kredibilitas Join OTAA disesuaikan DevEUI, AppEUI dan AppKey. Apabila sensor mendeteksi perubahan yang memenuhi kondisi tertentu, aktuator akan diaktifkan secara langsung.
4. Sistem kemudian memeriksa status koneksi *gateway*. Apabila *gateway* belum online, sistem akan mencatat error pada log *MQTT* dan kembali menunggu hingga koneksi berhasil terbentuk.
5. Apabila kondisi sensor tidak memicu aktuator, data hasil pembacaan sensor dibungkus dalam sebuah *payload* dan dikirimkan melalui modulasi LoRa pada frekuensi yang telah ditentukan menuju *gateway SenseCAP M2*.

6. *Gateway* yang berada dalam mode booting menunggu data yang dikirimkan oleh *end node*. Setelah sinyal LoRa diterima, *gateway* meneruskan paket data tersebut ke Chirpstack *Network server* melalui jalur TCP/IP menggunakan protokol *MQTT*.
7. Chirpstack *Network server* melakukan verifikasi terhadap kredensial perangkat berupa DevEUI, AppEUI, dan AppKey yang dikirimkan. Apabila kredensial tidak terdaftar dalam sistem, data akan ditolak dan *gateway* kembali menunggu transmisi berikutnya. Apabila kredensial valid, proses dilanjutkan ke tahap berikutnya.
8. Setelah verifikasi berhasil, Chirpstack mendekripsi frame LoRaWAN dan mengubah isi *payload* menjadi format JSON yang dapat dibaca oleh sistem. Data sensor yang telah didekode kemudian disimpan ke dalam database *PostgreSQL* untuk keperluan pencatatan frame log dan konfigurasi, serta ke *InfluxDB* untuk penyimpanan data time-series sensor.
9. Data yang telah tersimpan selanjutnya ditampilkan pada dash*board* website sebagai visualisasi monitoring kondisi lingkungan secara berkala. Setelah satu siklus selesai, sistem kembali mengulang loop dari tahap pembacaan sensor sehingga proses monitoring berjalan secara kontinu.



Gambar 3.3 Flow Chart Sistem

3.1.3. Spesifikasi Alat

Spesifikasi alat pada sistem yang dirancang bertujuan untuk menjelaskan karakteristik teknis dari setiap komponen utama yang digunakan dalam mendukung pembangunan layanan *server* LoRaWAN On-Premise Berbasis Chirpstack. Spesifikasi ini menjadi acuan dalam memahami kemampuan perangkat serta kesesuaian sistem dalam mendukung komunikasi data berbasis LoRaWAN sesuai pada tabel 3.1 dan table 3.2 dibawah ini.

Tabel 3.1 Spesifikasi Perangkat Keras Pendukung Layanan *Server* LoRaWAN On-Premise

No	Nama Perangkat	Spesifikasi
1	*SenseCAP M2* LoRaWAN *Gateway*	Prosesor : MT7628 (MIPS24KEc @ 580 MHz) RAM : DDR2 128 MB Flash Memory : 32 MB LoRa Chip : SX1302 Channel : 8 Channel

		Protokol LoRaWAN : V1.0.3 (Kelas A / B / C) Frekuensi LoRa : IN865 / EU868 / RU864 / US915 / AU915 / KR920 / AS923 Sensitivitas : -125 dBm @ 125 kHz (SF7) -139 dBm @ 125 kHz (SF12) TX Power : hingga +26 dBm Antena : LoRa : External 3 dBi (dengan base) Wi-Fi : Internal Impedansi Antena : 50 Ohm Konektivitas : Wi-Fi 2.4 GHz (802.11 b/g/n) Ethernet : 1x RJ45 (10/100 Mbps), mendukung PoE PD Seluler : Opsional (versi tertentu) Tegangan Input : DC 12V / 2A PoE (IEEE 802.3af, 40–57V DC) Interface : RJ45 Ethernet ×1
2	*IBM System x3650 M4*	Operating System: Proxmox Prosesor: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz; 32 Core RAM: 32 GB RAM SSD: 207.17 GB Total Storage Port: 8 USB 2.0 Ports 2 VGA Ports 4 Gigabit Ethernet RJ45 Ports

3	Heltec WiFi LoRa V3	MCU : ESP32-S3FN8 (Xtensa® 32-bit LX7 dual-core) Chip LoRa : SX1262 Frekuensi LoRa : 470–510 MHz dan 863–928 MHz Daya Tx Maks. : 21 ± 1 dBm Sensitivitas Rx : -134 dBm (SF12, BW=125KHz) Memori : 8MB Flash / 512KB SRAM Konektivitas Nirkebal : Wi-Fi 802.11 b/g/n, Bluetooth 5
4	Fiberhome HG6145F	Dual Band Frequency (2.4Ghz/5GHz) Channel *Bandwidth* 20MHz Transfer Rate Up to 200Mbps
5	Acer Swift SF313-51	OS: Windows 11 Home Processor: Intel Core i7-8550U @ 1.80GHz RAM: 8 GB
6	LAN CAT 6	RJ45 Connector

Tabel 3.2 Spesifikasi Perangkat Lunak Pendukung Layanan *Server* LoRaWAN On-Premise

No	Nama Perangkat Lunak	Spesifikasi
1	*Proxmox VE*	CPU: 32 Core RAM: 31.20 GiB Storage: 130 GB
2	Virtual Machine	vCPU: 2 Core (1 Socket) RAM: 4 GB Storage: 30 GB (VirtIO) OS Type: Ubuntu 22.04.5 LTS (Jammy

		Jellyfish)
3	*Docker*	*Docker* version 29.4.2, build 055a478
4	Chirpstack	Chirpstack version 4
5	*PostgreSQL*	*PostgreSQL* version 14
6	*Redis*	*Redis* version 7
7	*Mosquitto*	*Mosquitto* version 2
8	*InfluxDB*	*InfluxDB* version 2.7.12

Pada Tabel 3.1 dan 3.2 disajikan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam perancangan infrastruktur layanan *server* LoRaWAN on-premise berbasis Chirpstack. Sistem operasi utama yang digunakan pada *virtual machine* adalah Ubuntu 22.04.5 LTS (Jammy Jellyfish) yang menyediakan lingkungan operasional untuk seluruh layanan *server*. Seluruh komponen layanan dikemas menggunakan *Docker* versi 29.4.2 guna memudahkan manajemen dan isolasi antar layanan yang berjalan secara bersamaan.

Untuk implementasi *network server*, Chirpstack versi 4 digunakan sebagai perangkat lunak utama yang mencakup fungsi autentikasi perangkat, dekripsi *payload* LoRaWAN, serta manajemen aplikasi dan *gateway*. Chirpstack dikonfigurasi melalui laptop Acer Swift SF313-51 dan terhubung dengan *gateway SenseCAP M2* yang berfungsi sebagai perangkat penerima sinyal LoRa dari *end node*. Di sisi penyimpanan data, *PostgreSQL* versi 14 digunakan untuk menyimpan data konfigurasi jaringan dan frame log, *Redis* versi 7 berperan sebagai penyimpanan sementara berbasis memori untuk mendukung performa sistem, serta *InfluxDB* versi 2.7.12 digunakan untuk menyimpan data sensor berupa time-series yang diterima dari *end node*. Komunikasi antara *gateway* dan Chirpstack *Gateway bridge* dikelola menggunakan *Mosquitto* versi 2 sebagai *MQTT* message *broker*.

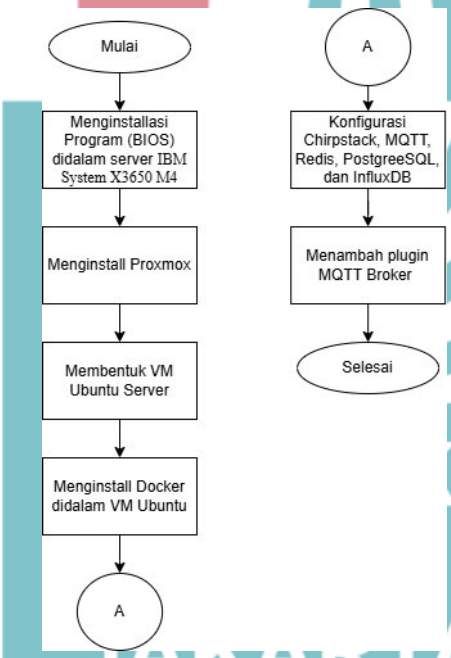
Pada sisi infrastruktur fisik, seluruh *virtual machine* dijalankan di atas *server IBM System x3650 M4* yang menggunakan Proxmox sebagai *hypervisor* dengan spesifikasi prosesor Intel Xeon E5-2670 32 core dan RAM 32 GB. *Virtual machine* yang dialokasikan untuk layanan Chirpstack dikonfigurasi dengan 2 vCPU, RAM 4 GB, dan storage 30 GB. Seluruh perangkat dihubungkan dalam satu

jaringan lokal menggunakan kabel LAN CAT 6 melalui router Fiberhome HG6145F. Dengan pengaturan ini, keseluruhan sistem dapat berfungsi secara optimal untuk mendukung pembangunan layanan *server* LoRaWAN on-premise yang dirancang.

3.1.4. Alur Perancangan Sistem

a. Perancangan Sistem layanan *server*

Dalam perancangan sistem layanan *server* LoRaWAN on-premise berbasis Chirpstack akan melewati 6 tahap yang akan dimulai dari alur perancangan sistem yang dapat dilihat pada gambar 3.4.



Gambar 3.4 Flowchart Perancangan Sistem

Berdasarkan flowchart pada Gambar 3.4, dijelaskan seluruh sistem yang dirancang melewati beberapa proses di awal diantaranya:

1. Tahap awal yang dilakukan adalah mengkonfigurasi BIOS (Basic Input/Output System) pada *server IBM System x3650 M4*. Konfigurasi ini meliputi pengaturan boot order agar *server* dapat melakukan booting dari media instalasi (USB atau DVD), pengaktifan virtualisasi (VT-x/AMD-V) untuk mendukung proses virtualisasi, serta penyesuaian pengaturan daya dan memori agar *server* berjalan optimal. Langkah ini penting dilakukan

karena menjadi fondasi agar sistem operasi dan perangkat lunak lain dapat diinstal dengan benar.

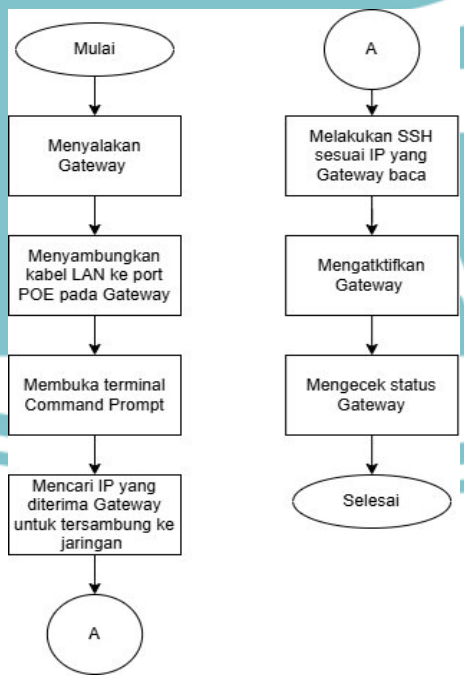
2. Setelah konfigurasi BIOS selesai, langkah berikutnya adalah menginstal Proxmox Virtual Environment (*Proxmox VE*) pada *server* fisik. *Proxmox VE* dipilih karena merupakan platform virtualisasi open-source yang memungkinkan pengelolaan *virtual machine* (VM) dan *container* secara efisien. Proses instalasi dilakukan melalui media bootable yang berisi image Proxmox, kemudian dilakukan konfigurasi awal seperti pengaturan alamat IP (dalam satu subnet dengan router Fiberhome), pengaturan storage, dan akses web-based untuk manajemen *server*.
3. Setelah *Proxmox VE* berhasil terinstal, langkah selanjutnya adalah membuat *virtual machine* (VM) yang akan menjadi lingkungan utama untuk menjalankan layanan *server* LoRaWAN. VM yang dibuat menggunakan sistem operasi *Ubuntu Server* dengan spesifikasi besarnya virtual RAM adalah 2 GB, jumlah core pada *virtual machine* berjumlah 2 core dan kapasitas storage pada sistem adalah 30 GB.
4. Setelah VM *Ubuntu Server* berhasil dibuat dan dijalankan, langkah berikutnya adalah menginstal *Docker* di dalam VM. *Docker* dipilih karena merupakan platform *container*isasi yang memungkinkan setiap layanan (Chirpstack, *MQTT*, *Redis*, PostgreeSQL dan *InfluxDB*) dijalankan dalam *container* yang terisolasi namun tetap dapat berkomunikasi satu sama lain. Penggunaan *Docker* menyederhanakan proses instalasi, konfigurasi, dan manajemen layanan.
5. Proses berikutnya adalah melakukan konfigurasi berbagai layanan yang dibutuhkan oleh sistem LoRaWAN, yaitu Chirpstack, *MQTT Broker*, *Redis*, *PostgreSQL*, dan *InfluxDB*. Chirpstack berperan sebagai *Network server* yang mengelola komunikasi perangkat LoRaWAN. *MQTT Broker* digunakan sebagai media pertukaran pesan antara *gateway* dan *server*. *PostgreSQL* berfungsi sebagai basis data utama untuk menyimpan konfigurasi perangkat, pengguna, dan informasi jaringan. *Redis* digunakan sebagai penyimpanan data sementara (in-memory database) untuk

meningkatkan performa sistem, sedangkan *InfluxDB* digunakan untuk menyimpan data telemetri dan data sensor yang bersifat time-series.

6. Langkah akhir dalam perancangan sistem, dilakukan penambahan atau konfirmasi plugin *MQTT Broker* pada Chirpstack agar *gateway* dapat terhubung ke *network server*. Plugin ini mengonfigurasi bagaimana Chirpstack berkomunikasi dengan *MQTT Broker*, termasuk pengaturan topic *uplink*, *downlink*, dan format pesan yang digunakan. Setelah plugin berhasil dikonfigurasi, *gateway SenseCAP M2* yang telah terdaftar dapat mulai mengirimkan data ke *server*.

b. Perancangan Aktivasi *Gateway*

Perancangan aktivasi *gateway* dilakukan untuk memastikan bahwa perangkat *gateway* dapat terhubung ke jaringan dan meneruskan paket menggunakan *broker MQTT*, keseluruhan alur dapat dilihat pada Gambar 3.5.



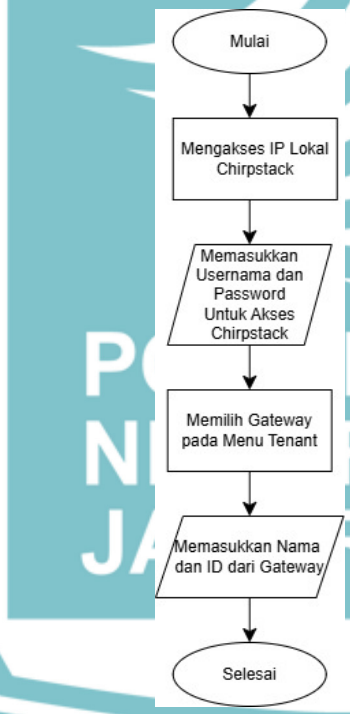
Gambar 3.5 Flowchart Aktivasi *Gateway*

1. Tahap awal dalam pengaktivasian *gateway* adalah dengan menyambungkan kabel power ke port power yang ada pada *Gateway SenseCAP M2*. Ketika *Gateway* sudah berhasil terhubung dengan power nantinya LED pada *Gateway* akan berkedip berwarna hijau kemudian LED akan tetap berwarna hijau.
2. Tahap kedua dimulai dengan menghubungkan kabel LAN (*Local Area Network*) dari router Fiberhome ke port Power on Ethernet yang tersedia pada *gateway SenseCAP M2*. Koneksi kabel ini berfungsi untuk menyediakan akses jaringan bagi *gateway* agar dapat berkomunikasi dengan *server* LoRaWAN yang telah dirancang sebelumnya.
3. Langkah berikutnya adalah membuka terminal Command Prompt melalui perangkat laptop yang digunakan untuk mencari IP yang dibaca oleh *Gateway* setelah tersambung oleh kabel LAN.
4. *Gateway* secara otomatis akan meminta alamat IP (*Internet Protocol*) dari DHCP *server* pada router Fiberhome. Setelah alamat IP *gateway* berhasil ditemukan, langkah selanjutnya adalah melakukan akses jarak jauh ke *gateway* menggunakan protokol SSH (*Secure Shell*). SSH digunakan untuk mengakses antarmuka command line *gateway* guna melakukan konfigurasi lebih lanjut.
5. Langkah selanjutnya dengan melakukan koneksi SSH ke *gateway* dengan memberi perintah ssh diikuti dengan username default *gateway* dan alamat IP yang telah ditemukan sebelumnya, misalnya ssh root@192.168.x.x. Setelah perintah dijalankan, sistem akan masuk ke dalam proses pengkonfigurasi untuk mengaktifkan *gateway*.
6. Setelah berhasil masuk ke antarmuka command line *gateway*, langkah selanjutnya adalah mengecek status *gateway* untuk memastikan bahwa perangkat berfungsi dengan baik melalui beberapa perintah seperti:
   uci set Chirpstack-*MQTT* forwarder.@*MQTT*[0].*server*=tcp://192.168.x.x,
   uci commit Chirpstack-*MQTT*-forwarder,
   /etc/init.d/Chirpstack-*MQTT*-forwarder restart,
   sleep 5,
   logread | grep-i "*MQTT*" | tail -10

7. Langkah terakhir adalah memastikan bahwa *gateway* benar-benar telah tersambung ke jaringan dan dapat berkomunikasi dengan *server* LoRaWAN. Indikator keberhasilan koneksi dapat dilihat dari munculnya log *MQTT* pada terminal *server* yang menampilkan pesan bahwa *gateway* telah berhasil melakukan publish/subscribe ke topic *MQTT*. Selain itu, pada dash*board* Chirpstack yang diakses melalui browser dengan alamat IP lokal *server*, status *gateway* akan berubah menjadi "connected" atau ditandai dengan indikator berwarna hijau.

c. Pendaftaran *Gateway* pada Platform Chirpstack

Proses pendaftaran *gateway* dilakukan pada platform Chirpstack yang dapat dilakukan sesuai pada Gambar 3.6.



Gambar 3.6 Flowchart Pendaftaran *Gateway*

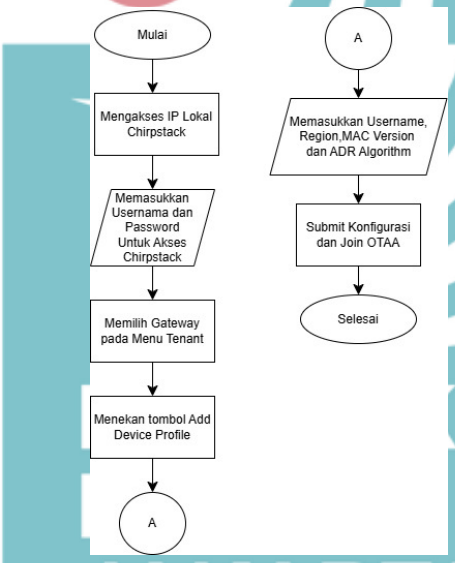
1. Proses pertama yang dilakukan adalah mengakses dash*board* Chirpstack melalui browser pada laptop atau PC yang terhubung ke jaringan lokal yang sama dengan *server*. Alamat yang dimasukkan adalah IP lokal *server* tempat Chirpstack berjalan, diikuti dengan port default Chirpstack yaitu 8080. Contoh alamat yang dimasukkan adalah http://192.168.x.x:8080.

2. Setelah halaman login Chirpstack muncul, langkah berikutnya adalah memasukkan kredensial username dan password yang telah ditentukan saat proses instalasi Chirpstack. Secara default, username administrator adalah admin dengan password admin. Setelah username dan password dimasukkan dengan benar, sistem akan mengarahkan pengguna ke halaman utama (dash*board*) Chirpstack.
3. Setelah berhasil masuk ke dash*board* Chirpstack, langkah selanjutnya adalah memilih menu Tenant yang tersedia pada antarmuka. Tenant dalam arsitektur Chirpstack berfungsi sebagai ruang kerja (workspace) yang memisahkan data dan konfigurasi antar pengguna atau organisasi yang berbeda. Pada umumnya, secara default terdapat tenant dengan nama tenant atau sesuai dengan konfigurasi awal saat instalasi. Setelah tenant dipilih, kemudian mengakses sub-menu *Gateway*s yang berada di dalam tenant tersebut. Menu *Gateway*s digunakan untuk mengelola semua *gateway* yang terdaftar pada *server* Chirpstack.
4. Pada halaman *Gateway*s, langkah berikutnya adalah menekan tombol Add *gateway* untuk menambahkan *gateway* baru ke dalam sistem. Setelah tombol tersebut ditekan, akan muncul formulir pendaftaran *gateway* yang harus diisi seperti name digunakan sebagai identitas nama dari *gateway*s dan memudahkan pengelolaan jika terdapat lebih dari satu *gateway*. Kemudian memasukkan *Gateway* EUI yang merupakan nomor identifikasi unik dari *gateway SenseCAP M2*. *Gateway* EUI ini biasanya tertera pada stiker yang ditempel di bodi *gateway* atau dapat ditemukan melalui antarmuka command line *gateway* dengan menjalankan perintah tertentu, seperti *gateway*-info atau melalui halaman administrasi *gateway*. *Gateway* EUI ini bersifat unik untuk setiap perangkat dan menjadi kunci utama bagi Chirpstack untuk mengenali *gateway* yang terhubung.
5. Setelah proses submit selesai, *gateway* akan muncul dalam daftar *gateway*s pada halaman *Gateway*s. Chirpstack akan mulai mendeteksi keberadaan *gateway* fisik yang sesuai dengan *Gateway* EUI yang telah didaftarkan. Jika *gateway* telah dikonfigurasi dengan benar (terhubung ke *MQTT broker* dengan alamat IP *server* yang tepat), maka status *gateway* pada Chirpstack

akan berubah menjadi "Connected" atau ditandai dengan indikator berwarna hijau. Status ini menandakan bahwa *gateway* telah berhasil terhubung ke *network server* dan siap digunakan untuk meneruskan komunikasi antara *end device* dan Chirpstack pada tahap pengujian selanjutnya.

d. Pendaftaran Device Profile pada Platform Chirpstack

Setelah *gateway* berhasil didaftarkan dan terhubung ke Chirpstack, langkah selanjutnya adalah membuat Device Profile. Profil ini akan digunakan sebagai acuan saat mendaftarkan *end device* (dalam penelitian ini adalah Heltec LoRa WiFi V3) ke dalam aplikasi. Berikut adalah langkah-langkah detailnya pada Gambar 3.7.



Gambar 3.7 Flowchart Pendaftaran Device Profiles

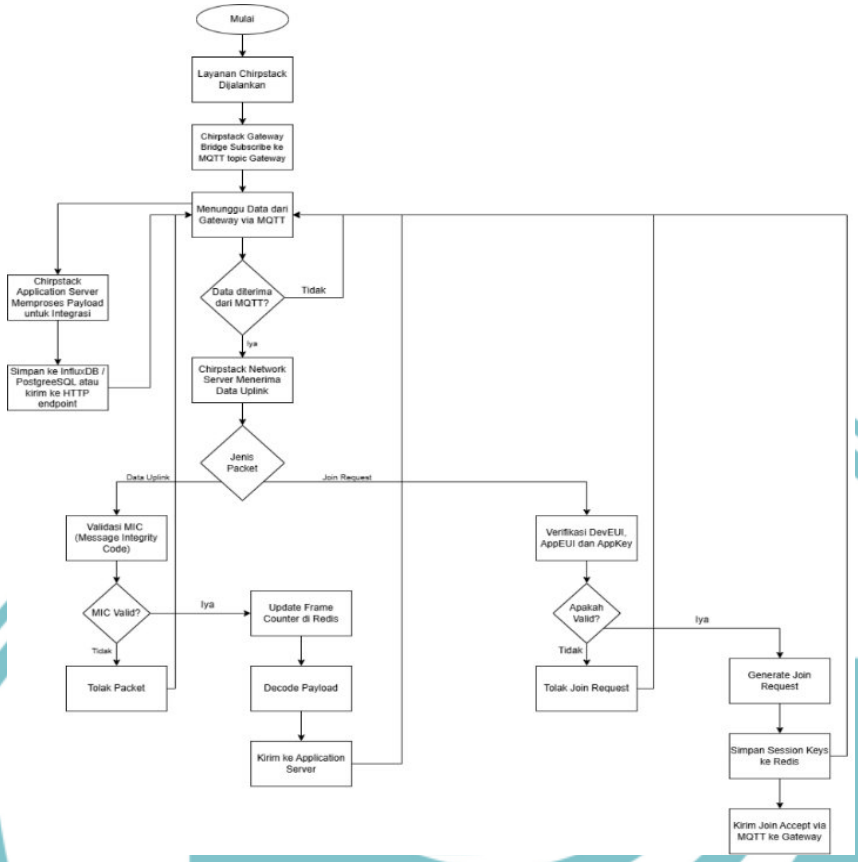
1. Langkah pertama yang dilakukan adalah mengakses dash*board* Chirpstack melalui browser pada laptop yang terhubung ke jaringan lokal yang sama dengan *server*. Alamat yang dimasukkan adalah IP lokal *server* dan port default Chirpstack yaitu 8080.
2. Setelah berhasil masuk ke dash*board*, pada antarmuka Chirpstack, administrator perlu memilih Tenant yang akan digunakan. Tenant berfungsi sebagai ruang kerja terpisah untuk mengelola berbagai entitas seperti *gateway*s, device profiles, dan applications. Setelah tenant yang tepat

dipilih, kemudian memilih sub-menu Device Profiles yang biasanya terletak di bagian bawah menu Tenant atau di sidebar navigasi.

3. Pada halaman Device Profiles, menekan tombol Add device profile untuk memulai proses pembuatan profil baru. Tombol ini biasanya berwarna biru dan terletak di sudut kanan atas halaman. Setelah ditekan, sistem akan menampilkan sebuah formulir konfigurasi yang harus diisi.
4. Pada tahap ini, mengisi formulir dengan parameter-parameter penting yang akan digunakan oleh semua perangkat yang memakai profil ini. Parameternya diantaranya Username, Region, MAC version dan ADR algorithm.
5. Setelah semua parameter terisi dengan benar, administrator menekan tombol Submit atau Add device profile untuk menyimpan konfigurasi. Profil yang baru dibuat akan muncul dalam daftar Device Profiles. Saat mendaftarkan device, administrator akan memilih profil yang baru saja dibuat. Dengan profil yang sudah tepat, proses Join OTAA antara *end device* dan *server* Chirpstack dapat berjalan dengan baik.

3.1.5. Cara Kerja Chirpstack

Pada subbab ini akan dijelaskan dan digambarkan cara kerja dari Chirpstack secara lengkap yang dapat dilihat pada Gambar 3.8.



Gambar 3.8 Flowchart Pendaftaran Device Profiles

Perancangan sistem layanan *server* LoRaWAN ini melibatkan bagian terkecil yaitu *end node* sebagai kumpulan *end device*, sensor dan aktuator. Bagian ini akan mempublikasikan data perubahan lingkungan menuju *gateway*, di mana data yang dikirim dibungkus oleh modulasi LoRa dengan lebar *bandwidth* 125 kHz pada setiap *Spreading factor* (SF7 – SF12). *Gateway* ketika sudah berhasil terhubung ke jaringan, statusnya menjadi online dan akan masuk ke dalam keadaan loop untuk menerima data yang berasal dari *end node*. Keseluruhan alur perancangan layanan dapat dilihat pada Gambar 3.8.

Paket yang diterima *gateway* kemudian diteruskan ke *server* melalui Chirpstack *Gateway bridge* menggunakan protokol *MQTT*, di mana *Mosquitto* berperan sebagai *broker* yang menangani lalu lintas pesan tersebut. Chirpstack *Gateway bridge* bertugas menerima data dari *packet forwarder* pada *gateway*, mengubah format data tersebut menjadi Protobuf atau JSON, kemudian mengirimkannya ke *MQTT Broker*. Selanjutnya, Chirpstack *Network server* yang merupakan inti dari sistem LoRaWAN telah melakukan subscribe ke *MQTT* topic

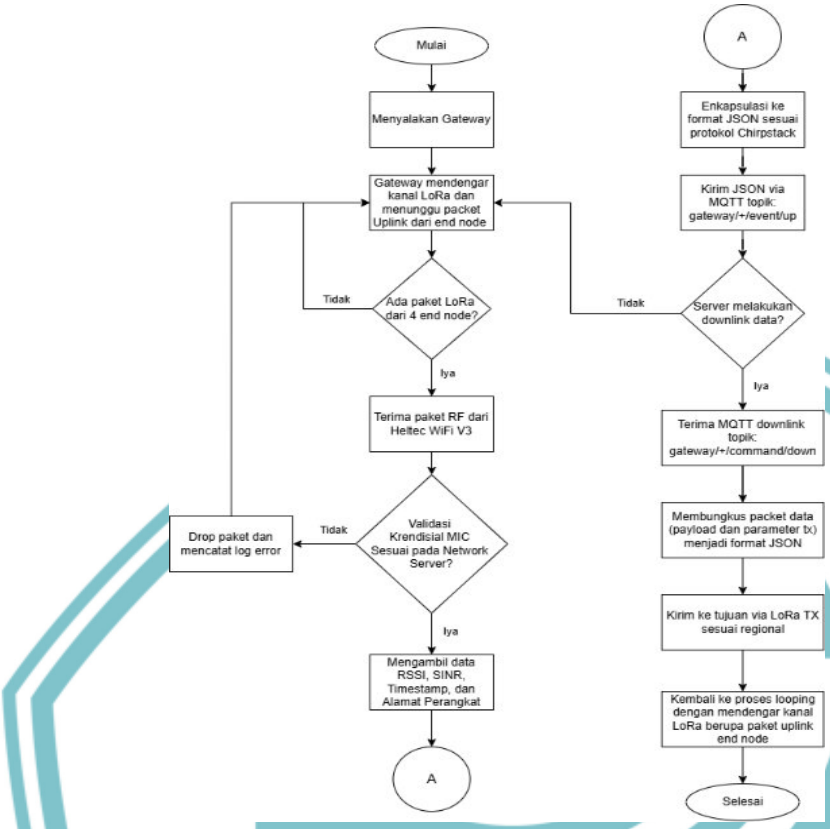
untuk menerima data *uplink* dari *gateway*. Ketika data diterima, *network server* akan menentukan jenis paket yang masuk.

Jika paket yang diterima adalah *Join Request* dari *end device* yang melakukan aktivasi OTAA, *network server* akan melakukan verifikasi kredensial perangkat (DevEUI, AppEUI, dan AppKey) yang sebelumnya telah didaftarkan pada database *PostgreSQL*. Apabila kredensial valid, *network server* akan menghasilkan *Join Accept* beserta session keys yang diperlukan untuk komunikasi selanjutnya. Session keys ini kemudian disimpan pada *Redis* (in-memory database) untuk mempercepat akses di masa mendatang. *Join Accept* kemudian dikirim kembali ke *gateway* melalui *MQTT broker* untuk diteruskan ke *end device*.

Jika paket yang diterima adalah data *uplink* biasa, *network server* terlebih dahulu memvalidasi MIC (Message Integrity Code) dari paket untuk memastikan keaslian dan integritas data, serta memverifikasi keaslian data menggunakan session key yang dihasilkan dari proses autentikasi OTAA sebelumnya. Setelah valid, *network server* melakukan proses de-duplication jika paket yang sama diterima oleh beberapa *gateway*. Selanjutnya, frame counter diperiksa dan diperbarui di *Redis* untuk mencegah replay attack. Apabila verifikasi berhasil, data akan didekripsi dan didekode menjadi format JSON yang dapat dibaca. Data kemudian diteruskan ke Chirpstack *Application server* yang bertugas mendekode *payload* (jika diperlukan) dan mengintegrasikan data ke berbagai platform seperti *InfluxDB* (untuk time series data) atau HTTP endpoint untuk kebutuhan visualisasi. Data yang telah diproses juga dapat dilihat melalui antarmuka web Chirpstack yang dapat diakses melalui alamat IP *server* pada jaringan lokal.

3.1.6. Cara Kerja *Gateway* didalam Sistem

Berikut penjelasan cara kerja *gateway* dalam perancangan layanan *server* LoRaWAN on-premise berbasis Chirpstack berdasarkan flowchart yang dapat dilihat pada Gambar 3.9.



Gambar 3.9 Flowchart Cara Kerja *Gateway*

1. Proses diawali dengan *gateway* dinyalakan. Pada tahap ini, *gateway* melakukan inisialisasi perangkat keras (modul LoRa dan koneksi Ethernet) serta menyiapkan koneksi *MQTT* ke *server* Chirpstack yang berada di lingkungan on-premise.
2. Dalam kondisi siap, *gateway* secara terus-menerus memindai kanal LoRa pada frekuensi yang telah ditentukan. *Gateway* melakukan booting, memuat konfigurasi regional (frekuensi LoRa), serta menghubungkan diri ke jaringan lokal melalui kabel LAN. *Gateway* bersifat pasif dalam mode ini, hanya menunggu adanya transmisi dari keempat *end node* Heltec WiFi LoRa V3 yang berada dalam jangkauannya.
3. *Gateway* melakukan pengecekan apakah ada paket LoRa yang masuk. Jika tidak ada paket yang terdeteksi, *gateway* akan tetap berada dalam mode mendengar (looping). Jika ada paket yang terdeteksi, *gateway* melanjutkan ke proses penerimaan paket RF.

4. Ketika sebuah *end node* mengirim data sensor (suhu, kelembaban, atau status aktuator), *gateway* menangkap sinyal radio frekuensi tersebut. Paket LoRa yang diterima masih dalam bentuk raw frame yang belum diolah.
5. *Gateway* memeriksa Message Integrity Code (MIC) dari paket yang diterima. Validasi ini bertujuan untuk memastikan bahwa paket tidak mengalami kerusakan atau perubahan selama perjalanan dari *end node* menuju *gateway*, serta menjamin bahwa paket benar-benar berasal dari *end node* yang sah. Jika kredensial tidak sesuai, maka paket dianggap tidak sah dan akan ditolak. Jika sesuai, *gateway* melanjutkan ke proses ekstraksi metadata.
6. *Gateway* mengekstrak informasi penting dari paket yang diterima, meliputi RSSI (kekuatan sinyal), SINR (rasio sinyal terhadap interferensi dan noise), timestamp (waktu penerimaan paket), serta alamat perangkat (DevAddr) untuk mengidentifikasi *end node* mana yang mengirim data.
7. Setelah validasi berhasil dan metadata diekstrak, *gateway* membungkus data *payload* dari *end node* bersama metadata (RSSI, SINR, timestamp, alamat perangkat) ke dalam format JSON. Format ini telah distandarisasi oleh Chirpstack sehingga dapat dipahami oleh komponen *Gateway bridge*.
8. *Gateway* mempublikasikan JSON yang telah terbentuk ke dalam protokol *MQTT* dengan topik *gateway*/+/event/up. Pesan ini akan dikirim melalui kabel LAN menuju router, lalu diteruskan ke *server* on-premise yang menjalankan Chirpstack *Gateway bridge*.
9. Setelah mengirim *uplink*, *gateway* memeriksa apakah ada perintah *downlink* dari *server* yang ditujukan ke *end node*. Perintah *downlink* biasanya berupa instruksi ke aktuator (misalnya menyalakan relay atau mengaktifkan buzzer). Jika tidak ada perintah *downlink*, *gateway* kembali ke mode mendengar kanal LoRa.
10. Apabila *server* mengirimkan perintah ke aktuator, *gateway* akan menerima pesan *MQTT* pada topik *gateway*/+/command/down. Pesan ini berisi perintah dari *Application server* Chirpstack yang sudah melalui *Network server* dan *Gateway bridge*.

11. *Gateway* mengambil *payload* perintah dan parameter transmisi (seperti frekuensi, daya pancar, dan *data rate*) dari pesan *MQTT* yang diterima, lalu membungkusnya ke dalam format JSON yang siap dikonversi menjadi paket LoRa.
12. *Gateway* mengubah JSON tersebut menjadi paket LoRa dan memancarkannya (LoRa TX) ke *end node* tujuan. Pengiriman ini mengikuti parameter regional yang telah dikonfigurasi, seperti AS923 untuk kawasan Asia Pasifik termasuk Indonesia.
13. Setelah proses *downlink* selesai, *gateway* kembali ke mode awal yaitu mendengarkan kanal LoRa untuk menangani paket *uplink* berikutnya dari keempat *end node*. *Gateway* akan terus melakukan perulangan proses ini selama perangkat masih menyala.

3.2. Realisasi Alat

Pada bagian subbab realisasi alat akan dijelaskan proses realiasi pembangunan layanan *server* LoRaWAN On-Premise berbasis Chirpstack meliputi implementasi set up *server* untuk sistem virtual machine, instalasi *docker* sebagai tempat menjalankan Chirpstack kemudian konfigurasi *gateway* dan komponen pendukung lainnya.

3.2.1. Realisasi dan Implementasi Setup *Server*

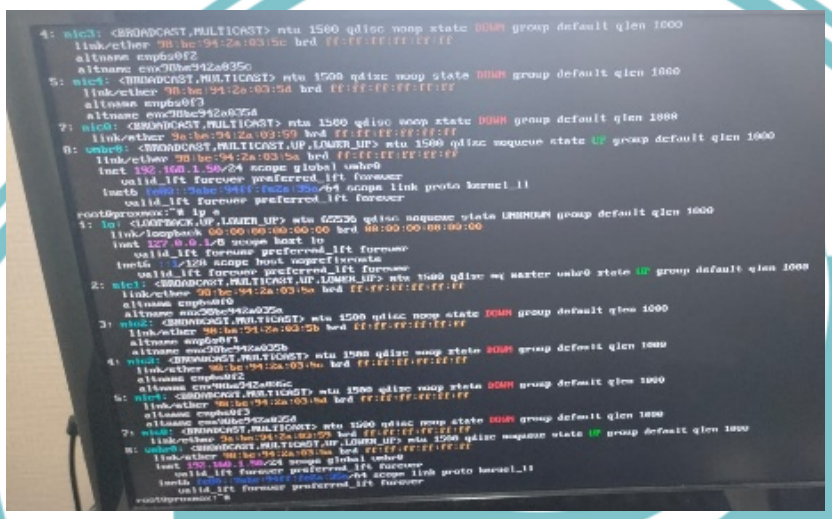
Slot kapasitas pada perangkat *IBM System x3650 M4* menggunakan 2 slot dengan masing-masing berkapasitas 128 GB. *Server* terhubung dengan konektivitas melalui kabel LAN yang dihubungkan ke router FiberHome agar bisa mengelola secara penuh infrastruktur seperti mendaftarkan perangkat *end device* melalui pengenerasian kunci deveui, appeui serta appkey di dalam *server*. Lainnya fitur Chirpstack yang dapat diaktif dengan penyambungan salah satunnya ADR (*Adaptive data rate*) dan integrasi protokol *MQTT* serta penerimaan paket secara real time melalui pengenalan data dari yang diteruskan oleh *gateway*. Keseluruhan setup perangkat keras dapat dilihat pada Gambar 3.10.

Gambar 3.10 Setup Slot SSD dan Konektivitas LAN *Server*

Setelah keseluruhan perangkat keras terpasang dan terkonfigurasi, langkah selanjutnya adalah melakukan verifikasi perangkat penyimpanan melalui BIOS *server IBM System x3650 M4*. Proses ini dilakukan untuk memastikan bahwa seluruh unit Solid State Drive (SSD) yang terpasang pada *server* telah terdeteksi dan dapat dikenali oleh sistem sebelum proses instalasi sistem operasi dimulai. Tampilan BIOS menampilkan informasi kapasitas serta status masing-masing SSD yang terpasang pada slot yang tersedia.

Proses berikutnya adalah melakukan instalasi *Proxmox VE* menggunakan media bootable (USB atau DVD). *Server* dinyalakan dan diatur agar melakukan booting dari media instalasi. Setelah masuk ke tampilan *install*er Proxmox, dilakukan konfigurasi awal meliputi pemilihan lokasi dan zona waktu, pengaturan password root, serta konfigurasi jaringan (IP address, netmask, *gateway*, dan DNS) secara langsung di dalam wizard instalasi. Pada tahap pemilihan target disk, dilakukan pembagian partisi penyimpanan dengan membuat direktori terpisah untuk masing-masing pengguna. Direktori pertama dialokasikan sebagai ruang penyimpanan untuk kebutuhan sistem Proxmox dan sisa ruang penyimpanan akan digunakan dalam penelitian pembangunan layanan *server* LoRaWAN, sedangkan direktori kedua dialokasikan untuk rekan peneliti yang menggunakan infrastruktur *server* yang sama. Pembagian direktori ini bertujuan untuk memisahkan lingkungan kerja antar pengguna sehingga tidak terjadi konflik data maupun konfigurasi sistem selama proses penelitian berlangsung. Instalasi kemudian dilanjutkan dengan menyalin paket-paket Proxmox ke disk target hingga proses selesai.

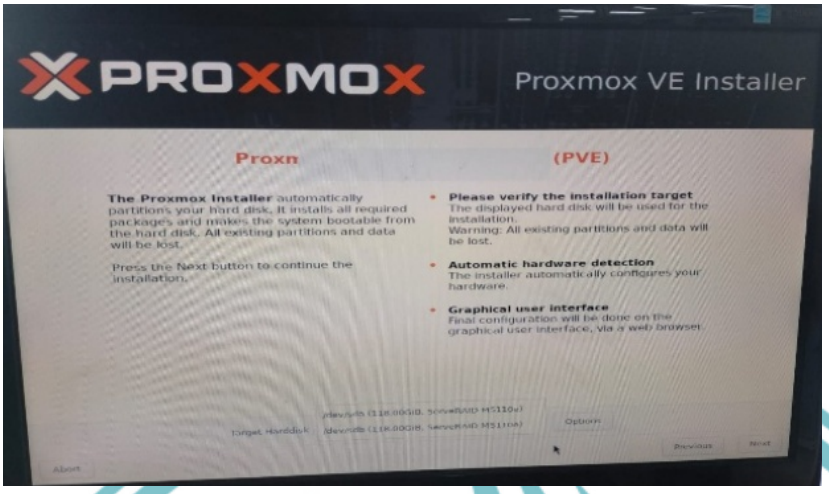
Setelah proses partisi dan pembentukan direktori selesai dilakukan, sistem melanjutkan proses booting hingga berhasil masuk ke antarmuka Command-Line Interface (CLI) seperti pada Gambar 3.11. Keberhasilan proses ini ditandai dengan munculnya shell prompt pada layar terminal, yang mengindikasikan bahwa sistem operasi telah berhasil dimuat dan *server* siap untuk dikonfigurasi lebih lanjut. Pada tahap inilah proses instalasi Proxmox sebagai *hypervisor* dan pembentukan *virtual machine* untuk platform Chirpstack dapat dimulai.



Gambar 3.11 Tampilan Command Line

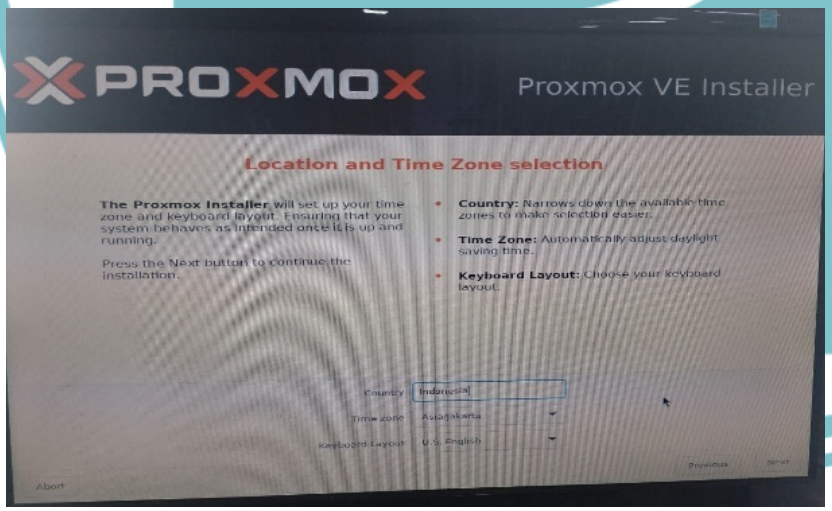
3.2.2. Realisasi Setup Proxmox

Proses instalasi *Proxmox VE* pada *server IBM System x3650 M4* diawali dengan menyiapkan media instalasi berupa USB flash drive yang telah ditanamkan image ISO *Proxmox VE*. Media instalasi tersebut kemudian dihubungkan ke port USB *server*, lalu *server* dinyalakan. Pada tahap booting awal, tombol F12 ditekan untuk mengakses boot menu guna mengarahkan sistem agar melakukan booting dari USB, bukan dari harddisk yang terpasang. Ketika *server* berhasil booting dari media instalasi, layar menampilkan menu awal *Proxmox VE* dengan beberapa opsi yang tersedia. Opsi "*Install Proxmox VE*" dipilih untuk memulai proses instalasi melalui wizard berbasis teks yang akan memandu seluruh tahapan konfigurasi secara bertahap. Tampilan layer menu *Proxmox VE* terdapat pada Gambar 3.12.



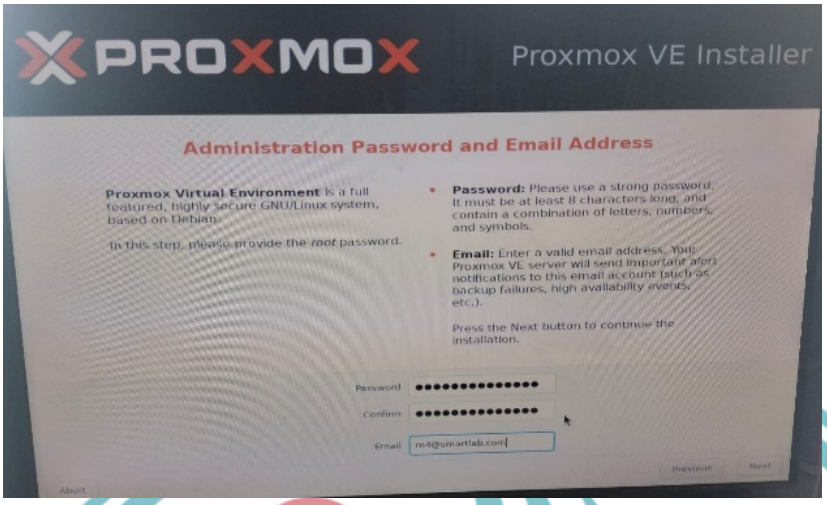
Gambar 3.12 Tampilan Menu *Installer Proxmox VE*

Pada tahap berikutnya wizard, tampilan layer akan berubah untuk diminta menyetujui lisensi perangkat lunak Proxmox yang bersifat open source, dilanjutkan dengan pemilihan zona waktu Asia/Jakarta sesuai lokasi operasional *server* di Politeknik Negeri Jakarta. Pemilihan zona waktu ini krusial untuk memastikan keakuratan timestamp pada seluruh log sistem dan data yang diteruskan ke platform Chirpstack terlihat pada Gambar 3.13.



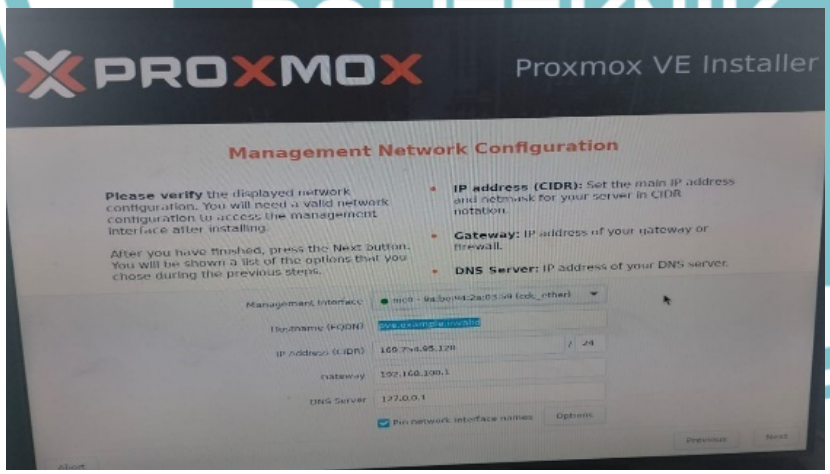
Gambar 3.13 Tampilan Menu Pemilihan Zona Waktu

Selanjutnya, mengatur kata sandi akun root sebagai administrator tertinggi sistem beserta alamat email yang digunakan untuk menerima notifikasi kondisi *server* seperti pada Gambar 3.14.



Gambar 3.14 Tampilan Menu Proxmox Memasukkan Email dan Kata Sandi

Pada tahap konfigurasi jaringan, *install*er secara otomatis mendeteksi antarmuka Ethernet yang tersedia, kemudian pengguna mengisikan parameter jaringan secara manual meliputi hostname, alamat IP statis, netmask, *gateway*, dan alamat DNS *server*. Penetapan alamat IP statis pada tahap ini sangat penting mengingat Proxmox dikelola melalui antarmuka web berbasis HTTPS yang mensyaratkan alamat *server* dapat dijangkau secara konsisten dari komputer lain dalam jaringan yang sama. Proses pengaturan dapat dilihat pada Gambar 3.15.

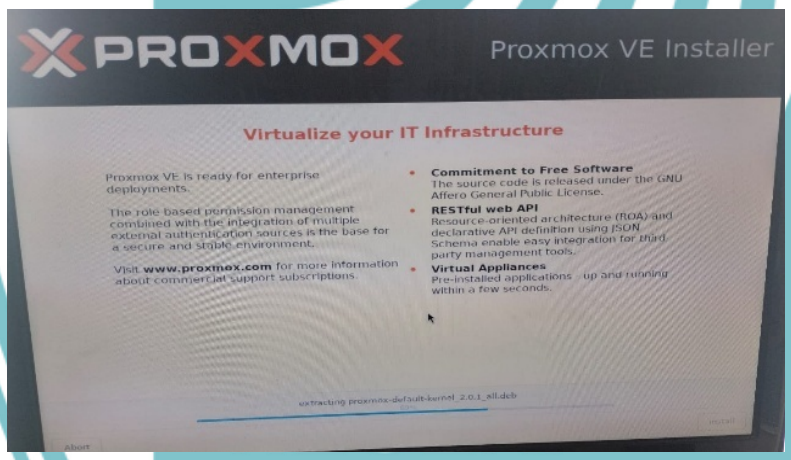


Gambar 3.15 Tampilan Menu Proxmox untuk Mengatur Jaringan

Setelah konfigurasi jaringan selesai, *install*er menampilkan daftar media penyimpanan yang telah terdeteksi, termasuk SSD yang sebelumnya telah diverifikasi melalui BIOS. SSD yang dituju dipilih sebagai target instalasi, di mana *install*er secara otomatis membentuk skema partisi optimal untuk kebutuhan

virtualisasi yang terdiri dari partisi root berformat ext4 dan partisi swap yang disesuaikan dengan kapasitas RAM *server*. Untuk kebutuhan pemisahan ruang penyimpanan antar pengguna, konfigurasi partisi tambahan dapat dilakukan melalui opsi Advanced atau melalui Logical Volume *Manager* (LVM) setelah instalasi selesai.

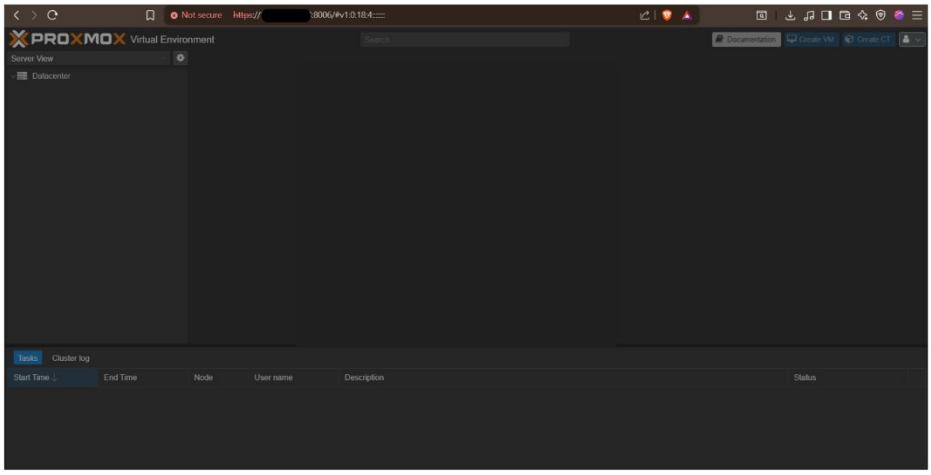
Proses instalasi sesungguhnya dimulai setelah seluruh konfigurasi dikonfirmasi. Sistem secara otomatis memformat SSD target dan menyalin seluruh paket Proxmox yang mencakup sistem operasi Debian, kernel virtualisasi KVM, dukungan *container* LXC, serta antarmuka web administrasi. Proses ini berlangsung selama lima hingga lima belas menit bergantung pada performa SSD dan kecepatan media instalasi USB yang digunakan. Tampilan layer ketika proses *install*asi dilakukan pada Gambar 3.16.



Gambar 3.16 Tampilan Proxmox Ketika *Install*asi Dijalankan

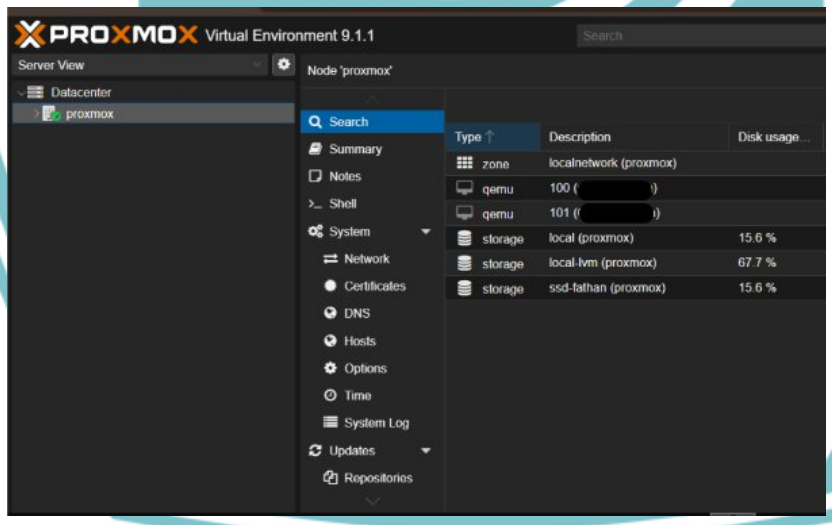
Setelah instalasi selesai, media instalasi USB dilepas dan *server* melakukan restart otomatis. *Server* kemudian melakukan booting dari SSD yang telah terinstalasi Proxmox, ditandai dengan munculnya antarmuka command-line sistem operasi Debian beserta pesan "Welcome to *Proxmox VE*" dan prompt login. Pengguna memasukkan kredensial root untuk mengakses sistem, yang menandakan bahwa Proxmox telah berhasil terinstalasi dan berjalan dengan baik.

Verifikasi akhir dilakukan melalui komputer lain dalam jaringan laboratorium dengan mengakses antarmuka web Proxmox melalui peramban menggunakan alamat https://[IP_*Server*]:8006.



Gambar 3.17 Tampilan Dash*board* Proxmox Diakses Melalui Browser

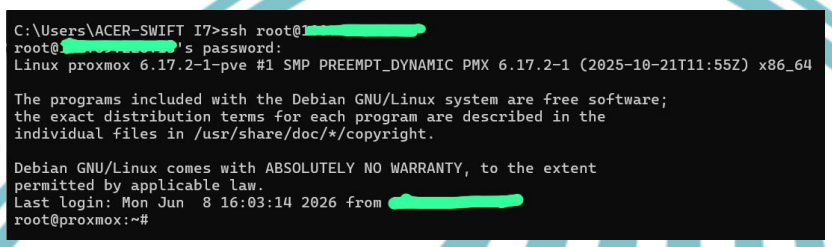
Peringatan sertifikat SSL yang muncul dapat diabaikan karena Proxmox menggunakan sertifikat yang diterbitkan secara mandiri. Setelah autentikasi berhasil, dash*board* Proxmox menampilkan ringkasan sumber daya *server* mencakup utilisasi CPU, konsumsi memori RAM, kapasitas penyimpanan, serta daftar *virtual machine* dan *container* yang aktif.



Gambar 3.18 Tampilan Ringkasan Informasi Sistem Proxmox

Keberhasilan akses ke antarmuka web ini menandakan bahwa tahap instalasi dan konfigurasi awal *hypervisor* telah selesai, dan *server IBM System x3650 M4* siap difungsikan sebagai platform virtualisasi untuk seluruh layanan Chirpstack on-premise.

Selain melalui antarmuka web, *server* Proxmox beserta *virtual machine* yang berjalan di dalamnya dapat diakses secara remote menggunakan protokol *Secure Shell* (SSH) melalui *Command Prompt* pada sistem operasi Windows maupun terminal pada Linux/macOS. SSH merupakan protokol jaringan terenkripsi yang memungkinkan pengguna untuk mengakses dan mengelola *server* dari jarak jauh secara aman melalui koneksi jaringan lokal maupun internet seperti pada Gambar 3.19.

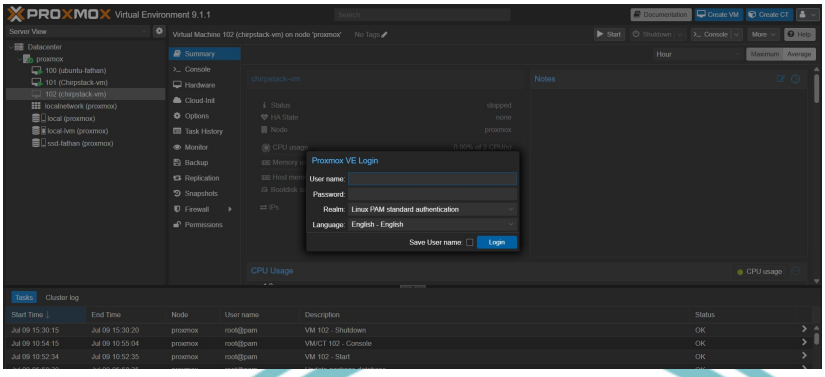


Gambar 3.19 Tampilan Command Prompt untuk melakukan Remote

Akses remote dilakukan dengan mengetikkan perintah ssh[username]@[ip_address] pada terminal, di mana pengguna akan diminta untuk melakukan autentikasi sebelum dapat masuk ke dalam sistem. Setelah sesi SSH berhasil terbentuk, pengguna memiliki kendali penuh terhadap *server* melalui antarmuka berbasis teks, mencakup instalasi perangkat lunak, modifikasi file konfigurasi, pemantauan kinerja sistem, serta debugging layanan Chirpstack yang berjalan pada virtual machine.

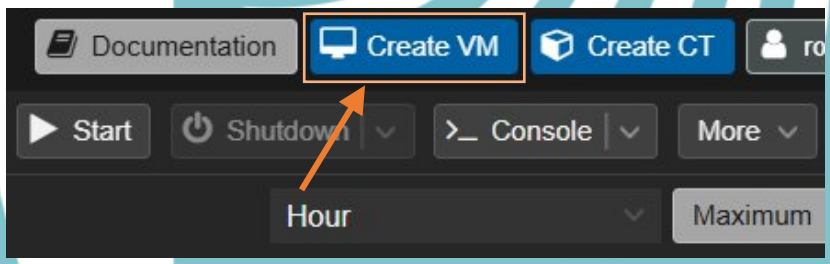
3.2.3. Realisasi Setup Virtual Machine

- Mempersiapkan bahwa layanan Proxmox sudah aktif dan dapat diakses melalui laman pencarian dengan mengakses http://IP_PROXMOX:8006.
- Saat laman berhasil diakses, akan muncul tampilan login ke dalam proxmox dengan memasukkan username dan password yang sesuai pada saat proses pengaturan administrator. Tampilan login terlihat pada Gambar 3.20.



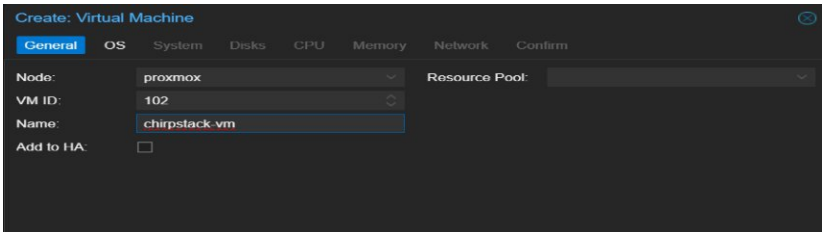
Gambar 3.20 Halaman Login Proxmox

- Langkah selanjutnya adalah memulai proses pembuatan *virtual machine* dengan mengklik tombol Create VM yang terletak di pojok kanan atas antarmuka web Proxmox yang dapat dilihat pada Gambar 3.21. Tombol ini akan memunculkan sebuah jendela yang terdiri dari beberapa tahap konfigurasi berurutan.



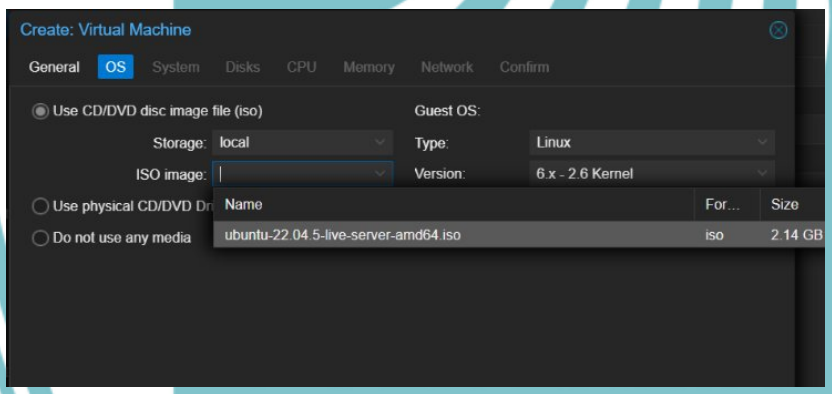
Gambar 3.21 Navigasi Tombol Create VM

- Tahap pertama ketika jendela sudah muncul adalah pengaturan umum atau General. Pada tahap ini, wajib mengisi nama untuk *virtual machine* yang akan dibuat. Pemberian nama akan membantu dalam identifikasi di kemudian hari, terutama ketika *server* Proxmox sudah mengelola banyak virtual machine. Proxmox secara otomatis akan memberikan nomor ID unik untuk *virtual machine* tersebut, biasanya dimulai dari 100 dan bertambah secara berurutan setiap kali *virtual machine* baru dibuat. Terlihat pada Gambar 3.22, bahwa VM ID nya 102 menunjukkan bahwa sudah lebih dari 2 VM yang ada didalamnya.



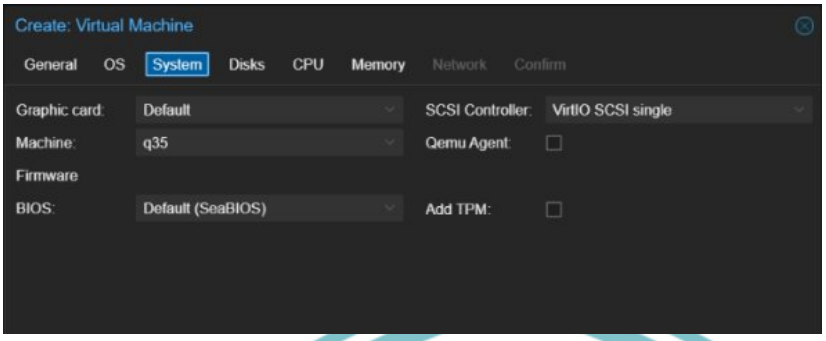
Gambar 3.22 Halaman General yang Harus Di isi

- Pada tahap kedua menunjukkan pemilihan media apa yang akan digunakan untuk menginstal sistem operasi ke dalam virtual machine. Setelah itu, memilih storage local dari menu drop-down, kemudian memilih file ISO *Ubuntu Server* yang telah diunggah pada menu ISO Image seperti pada Gambar 2.23. Pemilihan tipe sistem operasi tamu juga dilakukan pada tahap ini, misalnya memilih Linux sebagai tipe dan memilih versi kernel yang sesuai.



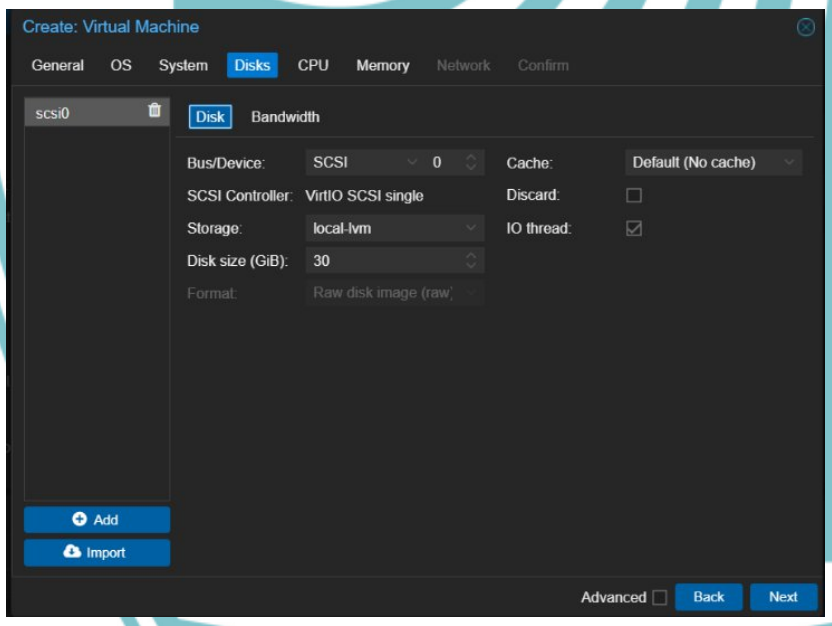
Gambar 3.23 Halaman Konfigurasi Sistem Operasi

- Tahap ketiga adalah konfigurasi sistem atau System. Pada umumnya, pengaturan default pada tahap ini sudah cukup baik dan tidak perlu diubah. Opsi Machine dipilih q35 karena chipset ini lebih modern dibandingkan i440fx dan mendukung perangkat keras PCI Express (PCIe) secara native. Pemilihan konfigurasi sistem dapat dilihat pada Gambar 3.24.



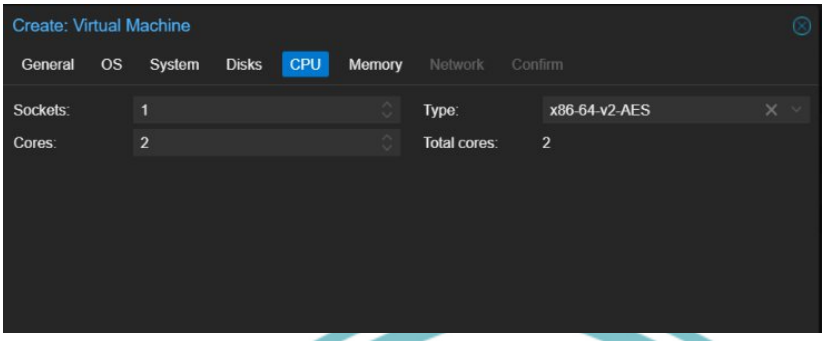
Gambar 3.24 Tampilan Konfigurasi Sistem

- Tahap keempat adalah konfigurasi hard disk atau Disks untuk menentukan ukuran penyimpanan yang akan dialokasikan untuk virtual machine. Ukuran hard disk yang disarankan untuk menjalankan platform Chirpstack beserta seluruh komponen pendukungnya seperti database *PostgreSQL*, *Redis*, dan *MQTT broker* adalah 30 gigabyte.



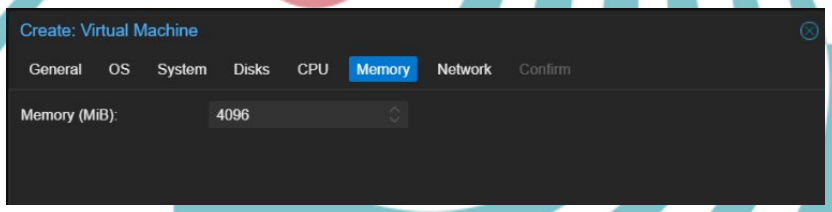
Gambar 3.25 Tampilan Konfigurasi Penyimpanan

- Tahap kelima adalah mengkonfigurasi CPU atau processor untuk menjalankan platform Chirpstack yang terdiri dari beberapa komponen yang berjalan secara bersamaan (*Network server*, *Application server*, *Gateway bridge*, serta *MQTT broker* dan database), alokasi 2 inti virtual sudah mencukupi sedangkan jumlah soket sebaiknya diatur ke 1.



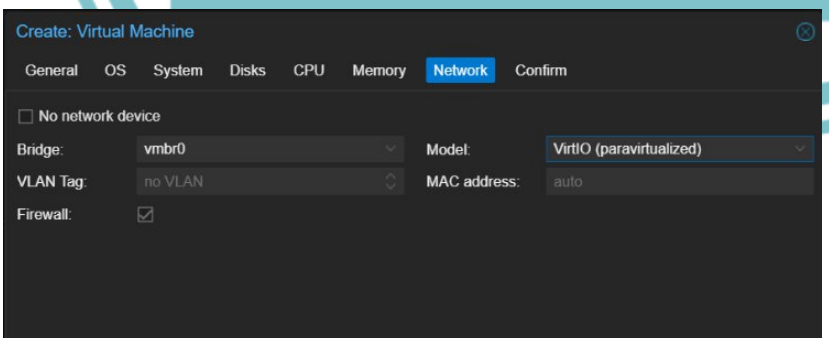
Gambar 3.26 Tampilan Konfigurasi CPU

- Tahap keenam adalah mengatur alokasi dari memori RAM untuk *Virtual machine* bagi Chirpstack. Untuk pengalokasian RAM cukup 4 gigabyte dengan kebutuhan yang menyesuaikan dengan penelitian.



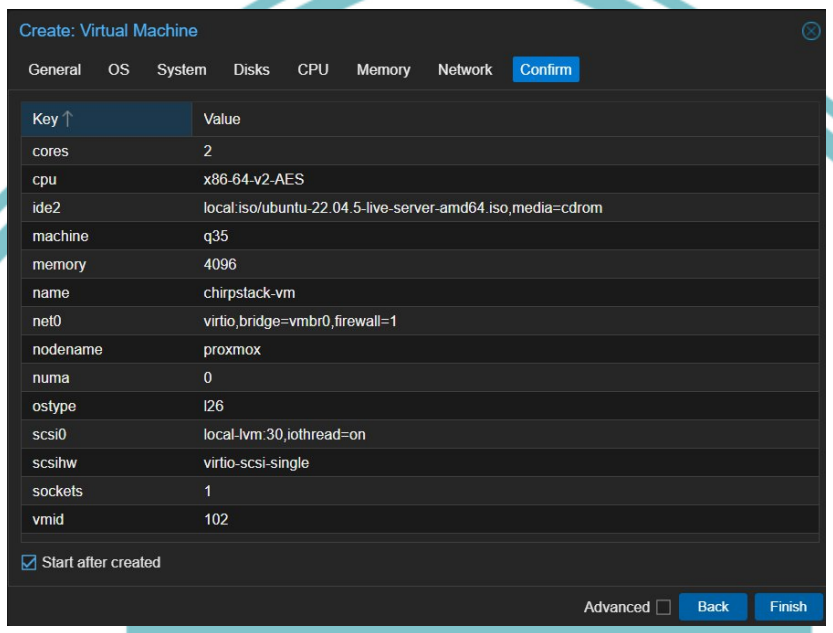
Gambar 3.27 Tampilan Konfigurasi RAM Virtual Machine

- Tahap ketujuh adalah konfigurasi jaringan yang akan digunakan oleh *virtual machine* untuk berkomunikasi dengan dunia luar. Proxmox secara default menyediakan bridge bernama vmbr0 yang berfungsi seperti saklar virtual yang terhubung ke antarmuka fisik ethernet *server* IBM X3650 M4. Untuk Bridge ini umumnya telah dikonfigurasikan pada saat instalasi Proxmox dan memiliki alamat IP yang sama dengan host Proxmox.



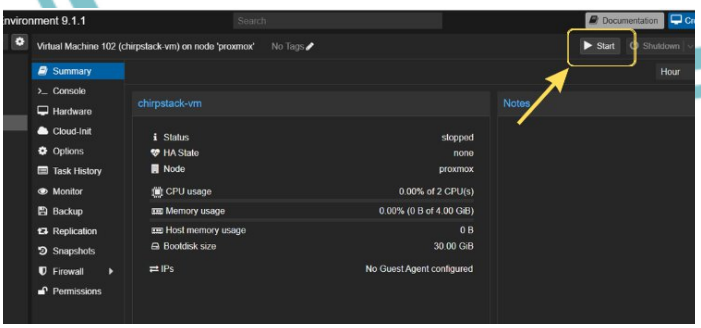
Gambar 3.28 Tampilan Konfigurasi Jaringan

- Setelah semua tahap konfigurasi sudah dilakukan, halaman konfirmasi atau Confirm akan berisi ringkasan seluruh pengaturan yang telah dilakukan. Halaman ini menjadi titik penting apakah seluruh konfigurasi sudah menyesuaikan dengan spesifikasi bagi setiap parameter. Jika dirasa sudah yakin dengan pengaturan dapat menekan tombol Finish untuk memulai proses pembuatan virtual machine.



Gambar 3.29 Tampilan Halaman Konfirmasi

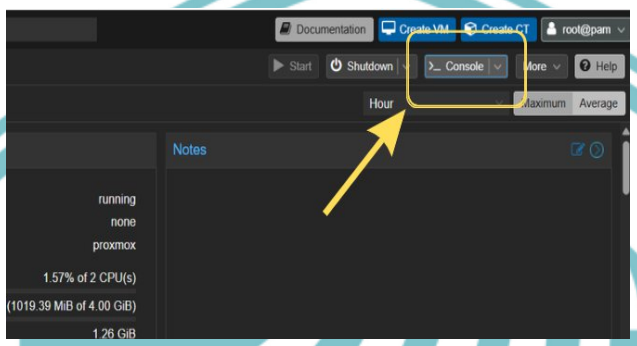
- Untuk menyalakan virtual machine, pilih nama *virtual machine* tersebut pada daftar, kemudian menekan tombol Start yang terdapat pada toolbar atas halaman pengelolaan virtual machine. Proxmox akan merespons dengan pesan bahwa *virtual machine* sedang dalam proses booting.



Gambar 3.30 Navigasi Tombol Start Virtual Machine

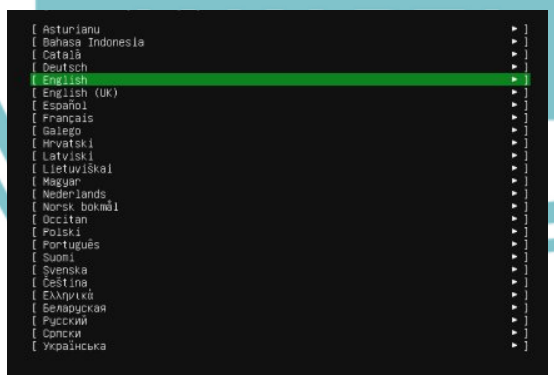
3.2.4. Realisasi Instalasi *Ubuntu Server* di Dalam Virtual Machine

- Setelah *virtual machine* berhasil dibuat dan dinyalakan melalui antarmuka Proxmox, langkah selanjutnya adalah melakukan instalasi sistem operasi *Ubuntu Server* 22.04 LTS ke dalam *virtual machine* tersebut. Proses instalasi diakses melalui jendela konsol *virtual machine* yang dibuka dengan mengklik tombol Console pada halaman pengelolaan virtual machine



Gambar 3.31 Navigasi Pengaksesan Console

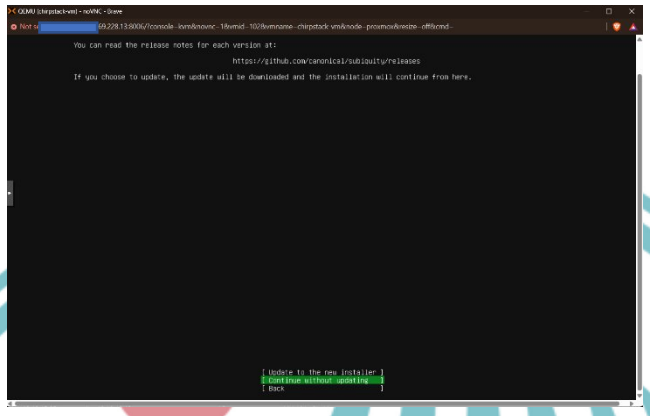
- Tahap pertama yang muncul pada layar konsol adalah tampilan pemilihan bahasa seperti yang terlihat pada gambar, di mana pengguna diminta untuk memilih bahasa yang akan digunakan selama proses instalasi. Pemilihan bahasa yang digunakan dalam tahap konfigurasi tidak akan mempengaruhi hasil akhir instalasi. Bahasa Inggris dipilih sebagai bahasa default seperti pada Gambar 3.32. Untuk mengkonfirmasi pemilihan bahasa, tekan tombol Enter pada key*board* laptop untuk mengonfirmasi pilihan.



Gambar 3.32 Tampilan Pemilihan Bahasa

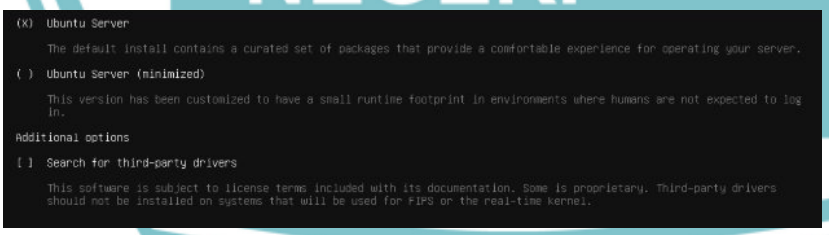
- Tahap berikutnya setelah pemilihan bahasa, layar akan menampilkan sebuah pemberitahuan bahwa tersedia versi baru dari *install*er Ubuntu. Laman akan menanyakan apakah pengguna ingin memperbarui ke *install*er baru atau

melanjutkan instalasi tanpa memperbarui. Pilih *Continue Without Updating* karena saat ini sudah cukup stabil dan mampu menyelesaikan proses instalasi dengan baik.



Gambar 3.33 Tampilan Tawaran Pembaharuan *Install*er

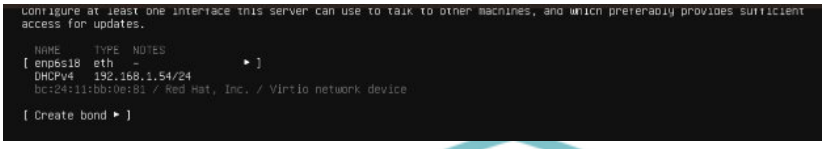
- Tahap selanjutnya tampilan layar akan menanyakan jenis instalasi yang diinginkan. Terdapat dua opsi utama yaitu *Ubuntu Server* (instalasi penuh dengan paket-paket standar *server*) dan Minimized (instalasi ringan hanya dengan paket-paket minimal). Untuk keperluan menjalankan platform Chirpstack beserta *Docker* dan komponen pendukungnya, pilihan *Ubuntu Server* direkomendasikan karena menyertakan paket-paket dasar yang diperlukan seperti alat jaringan dan manajemen paket. Setelah menentukan jenisnya pilih opsi Done untuk melanjutkan.



Gambar 3.34 Tampilan Jenis *Install*asi

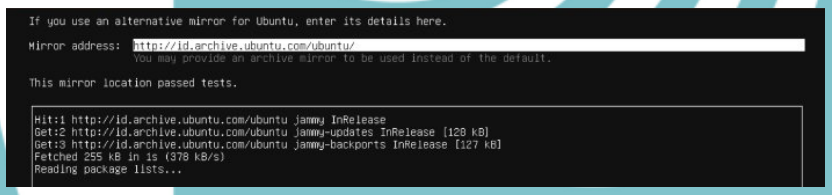
- Tahap selanjutnya adalah konfigurasi jaringan. *Install*er secara otomatis akan mendeteksi antarmuka jaringan virtual yang tersedia pada *virtual machine* berupa antarmuka ens18 sesuai pada konfigurasi *virtual machine* di Proxmox. Penentuan jaringan sebaiknya diatur dengan IP secara statis karena pengaksesan *server* dan *gateway* secara konsisten dan menghindari adanya perubahan IP

setiap kali *Virtual machine* di restart. Ketika semua parameter sudah diisi, pilih opsi Done untuk melanjutkan.



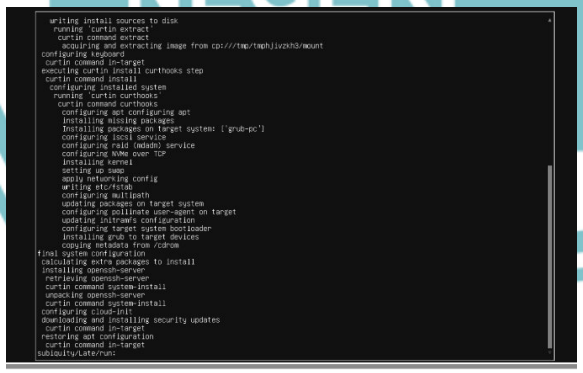
Gambar 3.35 Tampilan Konfigurasi Jaringan

- Pada tahap setelah menentukan konfigurasi jaringan, menentukan alamat mirror repository Ubuntu yang akan digunakan untuk mengunduh paket-paket selama instalasi dan setelah sistem berjalan. Secara default, *install*er menggunakan mirror resmi Ubuntu yaitu *http://archive.ubuntu.com/ubuntu*. Pilih secara default dan tekan Enter sebagai bentuk konfirmasi untuk melanjutkan proses instalasi Repositori kemudian halaman akan diarahkan menuju tahap selanjutnya.



Gambar 3.36 Tampilan Pemilihan Repositori Ubuntu

- Proses *Install*asi sudah selesai dan berhasil dilakukan.

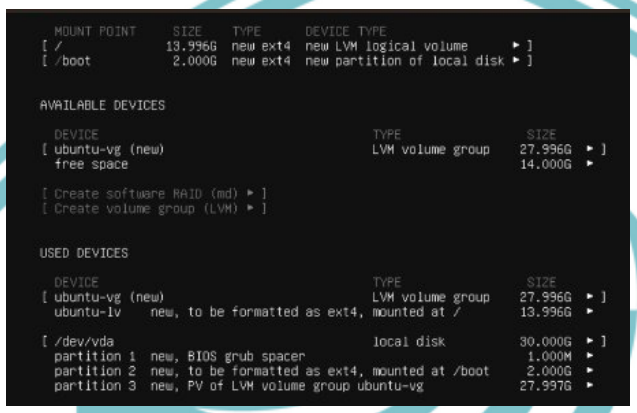


Gambar 3.37 *Install*asi Repositori Ubuntu

- Menuju tahap akhir, bagian partisi disk menjadi titik penting apakah seluruh kapasitas hard disk digunakan atau dapat melakukan kostumisasi untuk kebutuhan yang lebih spesifik. Pada penelitian perilaku partisi disk memilih

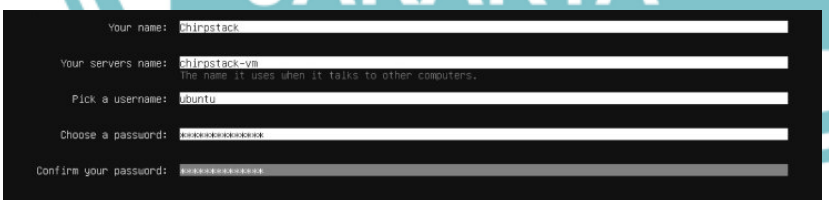
opsi Use An Entire Disk, *install*er *Ubuntu Server* secara otomatis menampilkan ringkasan skema partisi yang akan diterapkan pada hard disk virtual.

- Berdasarkan kapasitas 30 Gigabyte tersebut, *install*er secara otomatis membuat skema partisi yang terdiri dari tiga bagian utama, yaitu partisi untuk bootloader BIOS, partisi untuk direktori /boot, dan partisi untuk Logical Volume *Manager* (LVM) yang kemudian dibagi menjadi logical volume untuk direktori root /.



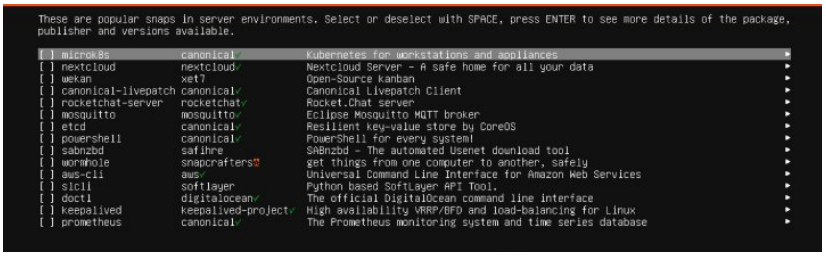
Gambar 3.38 Tampilan Partisi Disk

- Setelah partisi selesai dikonfigurasi, *install*er akan meminta untuk mengatur profil pengguna. Informasi yang perlu diisi meliputi: Your name: Nama lengkap pengguna (misalnya: Chirpstack); Your *server*'s name: Nama host untuk *server* (misalnya: Chirpstack-vm); Pick a username: Nama pengguna untuk login (ubuntu); Choose a password: Kata sandi untuk pengguna tersebut dan Confirm your password: Konfirmasi kata sandi.



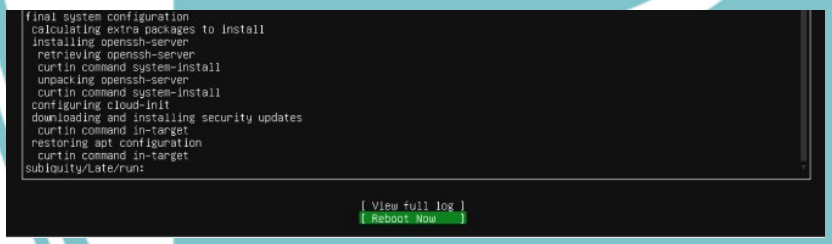
Gambar 3.39 Tampilan Pengisian Data Profil

- Untuk tahap selanjutnya ada proses pengunduhan fitur tambahan yang dapat disematkan pada sistem, pada penelitian akan mengikuti secara default fitur tambahan dengan langsung menekan tombol Enter pada key*board* laptop sebagai tanda konfirmasi.



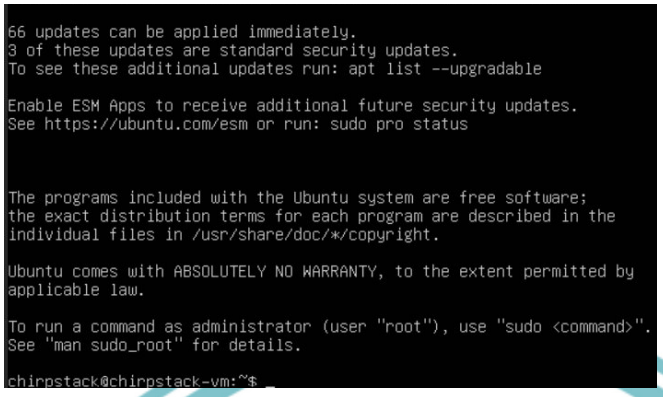
Gambar 3.40 Tampilan Layar *Install*asi Fitur Tambahan

- Setelah semua konfigurasi selesai, *install*er memulai proses instalasi dengan menyalin file-file sistem ke dalam hard disk virtual. Pada layar akan terlihat progres bar yang menunjukkan kemajuan instalasi, mulai dari menyalin file sistem (Copying files), menginstal bootloader (*Install*ing GRUB bootloader), hingga membersihkan dan menyelesaikan (Cleaning up and finishing). Proses instalasi ini biasanya memakan waktu antara 5 hingga 15 menit tergantung pada kecepatan hard disk virtual dan koneksi internet yang digunakan untuk mengunduh paket-paket tambahan.
- Ketika Proses *Install*asi telah selesai, *virtual machine* perlu restart terlebih dahulu dengan memilih opsi Reboot Now.



Gambar 3.41 Tampilan Layar Proses *Install*asi Berhasil dan Mulai Ulang

- Setelah proses booting selesai, tampilan konsol akan berubah menjadi layar login *Ubuntu Server* seperti yang terlihat pada gambar 3.42. Tampilan ini dikenal sebagai Message of the Day (MOTD) yang secara otomatis muncul setiap kali pengguna berhasil login ke sistem. Perubahan yang paling signifikan dari sebelumnya adalah bahwa *virtual machine* sekarang tidak lagi berada di lingkungan *install*er, melainkan sudah menjalankan sistem operasi *Ubuntu Server* yang telah terinstal di hard disk virtual.

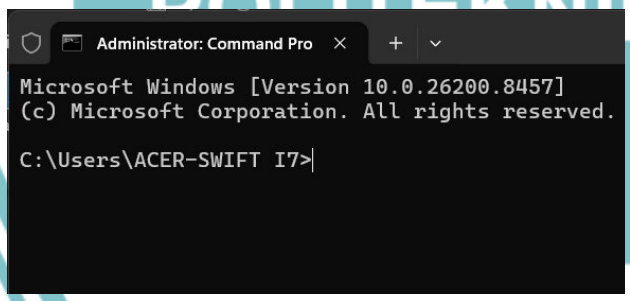


Gambar 3.42 Tampilan MOTD

3.2.5. Realisasi Setup Chirpstack *Server*

Ketika proses setup *virtual machine* sudah dilakukan melalui GUI proxmox, berikutnya adalah melakukan setup Chirpstack *server* dengan meng*install* beberapa layanan didalam VM yang diakses melalui terminal command prompt laptop dengan memanfaatkan layanan tailscale. Alasan digunakannya terminal melalui laptop

- Membuka terminal command prompt yang dapat dicari pada perangkat laptop. Ketika command prompt sudah terbuka akan muncul tampilan terminal yang terlihat pada Gambar 3.43.

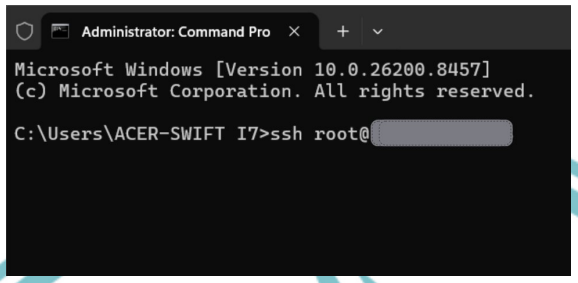


Gambar 3.43 Tampilan Terminal Command Prompt

- Untuk mengakses *virtual machine* didalam proxmox, langkah awal yang dapat dilakukan dengan masuk ke dalam sistem proxmox melalui ip tailscale yang didapatkan ketika disematkan pada sistem proxmox. Dengan menyambungkan tailscale didalam proxmox membuat sistem proxmox dapat diakses di luar jaringan lokal dimana jaringan proxmox tersambung. Perintah yang diberikan sebagai berikut:

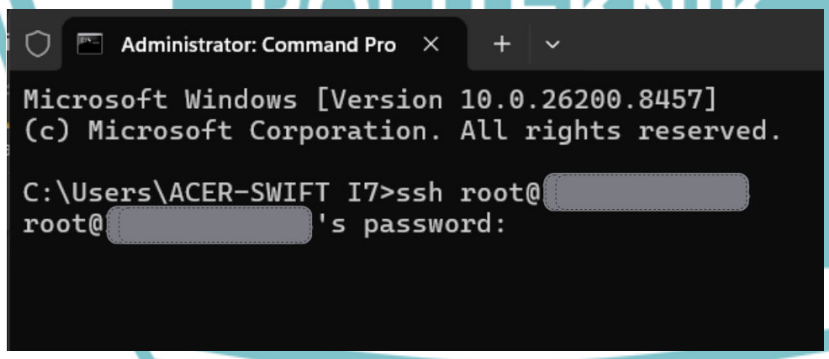
ssh username@IP_Tailscale

Proses memasukkan perintah pada terminal dapat dilihat pada Gambar 3.44.



Gambar 3.44 Tampilan Terminal Menunjukkan Perintah Akses Tailscale

- Ketika perintah di running oleh sistem nantinya secara langsung proxmox dapat diakses dan nantinya tampilan akan berubah lalu diminta untuk memasukkan kata sandi proxmox yang sudah dibuat pada proses setup *server* sebelumnya. Masukkan kata sandi yang benar dengan memperhatikan besar kecilnya sebuah huruf, tanda, angka dan lainnya. Password yang dimasukkan secara otomatis tidak ditampilkan oleh terminal sebagai bentuk keamanan data pengguna agar sistem yang dirancang tidak mudah diakses oleh orang lain. Memasukkan kata sandi dapat dilihat pada Gambar 3.45.

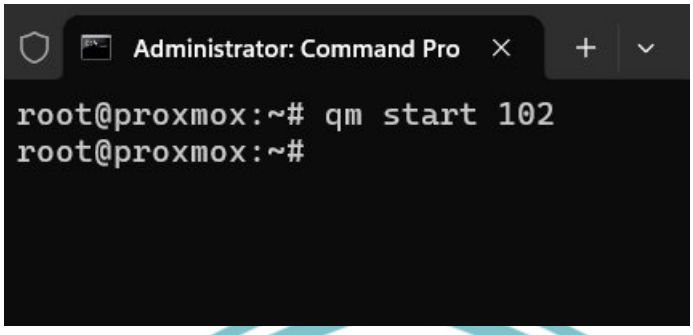


Gambar 3.45 Tampilan Terminal Ketika Memasukkan Kata Sandi

- Selanjutnya setelah berhasil mengakses *Proxmox VE* melalui terminal dapat ditandai dengan perubahan user menjadi:

username@username_server

- Nyalakan *Virtual machine* yang telah di konfigurasi pada tahap realisasi setup *virtual machine* dengan memberi perintah: *qm start 102* seperti pada Gambar 3.46.



Gambar 3.46 Tampilan Terminal Ketika Memulai VM

- *Virtual machine* berhasil dinyalakan melalui perintah *qm start 102* pada terminal *server* Proxmox. koneksi remote ke *virtual machine* dilakukan melalui protokol SSH dari laptop administrator dengan menggunakan format perintah *ssh username_vm@alamat_ip_vm* (Contoh *ssh root@192.168.1.100)* di mana alamat IP *virtual machine* telah ditetapkan secara statis pada saat instalasi *Ubuntu Server* sebelumnya. Proses dapat dilihat pada Gambar 3.47.

```
root@proxmox:~# ssh chirpstack@192.168.1.54
```

Gambar 3.47 Contoh Perintah Pengaksesan VM

- Pada koneksi pertama kali, sistem menampilkan peringatan authenticity of host yang mengindikasikan bahwa laptop belum pernah terhubung ke *virtual machine* tersebut sebelumnya. Administrator mengonfirmasi koneksi dengan mengetikkan yes untuk menyimpan sidik jari kunci host (host key fingerprint) ke dalam file known_hosts pada laptop, sehingga peringatan serupa tidak akan muncul pada koneksi berikutnya.

```
The authenticity of host '192.168.1.54 (192.168.1.54)' can't be established.
ED25519 key fingerprint is SHA256:pyPIn62x6pNvz3B6/jVqeejM0Uxs/htGd2fC5O9kgt0.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Gambar 3.48 Tampilan Proses Penyambungan ke Virtual Machine

- Sistem kemudian meminta autentikasi dengan menampilkan prompt password:, di mana administrator memasukkan kata sandi virtual machine. Sebagai fitur keamanan bawaan SSH, karakter kata sandi yang dimasukkan tidak ditampilkan pada layar.

- Apabila kredensial yang dimasukkan valid, koneksi SSH berhasil terjalin yang ditandai dengan berubahnya tampilan prompt terminal menjadi `username@nama_vm:‘(misalnya‘admin@ubuntu−Chirpstack:)’.
- Perubahan prompt ini mengindikasikan bahwa administrator kini telah berada di dalam lingkungan *virtual machine* dan memiliki kendali penuh terhadap sistem. Perubahan tampilan prompt dapat dilihat pada Gambar 3.49.

```
chirpstack@chirpstack-vm:~$
```

Gambar 3.49 Tampilan Prompt Ketika Berhasil Mengakses VM

- Proses pembangunan layanan Chirpstack dimulai dengan melakukan peningkatan dan memperbarui sistem dengan memberi perintah *sudo apt update* dan *sudo apt upgrade -y* untuk memastikan semua dependensi berada dalam versi terbaru sebelum proses instalasi dimulai. Proses pembaruan dan peningkatan dilihat pada Gambar 3.50.

```
chirpstack@chirpstack-vm:~$ sudo apt update & sudo apt upgrade -y
[1] 990
[sudo] password for chirpstack:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following NEW packages will be installed:
  netplan-generator python3-netplan python3-packaging
The following packages have been kept back:
  fwupd libjcat1
```

Gambar 3.50 Tampilan Proses Peningkatan dan Pembaharuan Sistem VM

- Sebelum instalasi komponen Chirpstack dimulai, *Docker* diinstalasi terlebih dahulu sebagai platform kontainerisasi yang akan mengelola seluruh layanan pendukung dalam satu lingkungan terisolasi. Pendekatan ini dipilih karena *Docker* memungkinkan seluruh layanan didefinisikan, dikonfigurasi, dan dijalankan secara bersamaan melalui satu file *docker-compose.yml*, sehingga proses manajemen layanan menjadi lebih efisien dan terstruktur. *Install* dependensi awal dengan perintah *sudo apt install -y ca-certificates curl gnupg lsb-release*.

```
chirpstack@chirpstack-vm:~$ sudo apt install -y ca-certificates curl gnupg lsb-release
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
lsb-release is already the newest version (11.1.0ubuntu4).
lsb-release set to manually installed.
ca-certificates is already the newest version (20240203~22.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.81.0-1ubuntu1.24).
curl set to manually installed.
gnupg is already the newest version (2.2.27-3ubuntu2.5).
gnupg set to manually installed.
```

Gambar 3.51 Tampilan Proses *Install*asi Dependasi *Docker*

- *Install Docker* menggunakan skrip resmi dengan perintah *curl -fsSL https://get.docker.com -o get-docker.sh* diikuti *sudo sh get-docker.sh*.

```
chirpstack@chirpstack-vm:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 2687d91ddeb3bd6aeae37a90947761efdee87030
+ sh -c apt-get -qq update >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install ca-certificates curl >/dev/null
+ sh -c install -m 0755 -d /etc/apt/keyrings
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" -o /etc/apt/keyrings/docker.asc
+ sh -c chmod a+r /etc/apt/keyrings/docker.asc
+ sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu jammy sta
ble" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get -qq update >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install docker-ce docker-ce-cli containerd.io docker-compose-plugi
n docker-ce-rootless-extras docker-buildx-plugin docker-model-plugin >/dev/null
```

Gambar 3.52 Tampilan Proses *Install*asi *Docker*

- *Install Docker* Compose dengan perintah *sudo apt install docker-compose -y*

```
chirpstack@chirpstack-vm:~$ sudo apt install docker-compose -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  python3-docker python3-dockerpty python3-docopt python3-dotenv python3-texttable python3-websocket
Recommended packages:
  docker.io
The following NEW packages will be installed:
  docker-compose python3-docker python3-dockerpty python3-docopt python3-dotenv python3-texttable python3-websocket
0 upgraded, 7 newly installed, 0 to remove and 2 not upgraded.
Need to get 290 kB of archives.
```

Gambar 3.53 Proses *Install*asi *Docker* Compose

- Tambahkan user ke grup *Docker* dengan perintah *sudo usermod -aG docker $USER*. Fungsi penambahan user dimaksudkan agar bisa mengakses *Docker*.

```
chirpstack@chirpstack-vm:~$ sudo usermod -aG docker $USER
```

Gambar 3.54 Proses Penambahan User

- *Install Docker* Compose dengan perintah *sudo apt install docker-compose -y*

```
chirpstack@chirpstack-vm:~$ sudo apt install docker-compose -y

[sudo] password for chirpstack:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-compose is already the newest version (1.29.2-1).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
```

Gambar 3.55 Tampilan Terminal Untuk Proses *Install*asi *Docker*

- Melakukan cloning repository Chirpstack *docker* yang berisikan file *docker-compose.yml*

```
chirpstack@chirpstack-vm:~$ git clone https://github.com/chirpstack/chirpstack-docker.git
Cloning into 'chirpstack-docker'...
remote: Enumerating objects: 583, done.
remote: Counting objects: 100% (428/428), done.
remote: Compressing objects: 100% (141/141), done.
remote: Total 583 (delta 374), reused 287 (delta 287), pack-reused 155 (from 2)
Receiving objects: 100% (583/583), 99.74 KiB | 2.70 MiB/s, done.
Resolving deltas: 100% (404/404), done.
```

Gambar 3.56 Tampilan Terminal untuk Proses Cloning Repository

- Menjalankan *docker* compose up -d dengan tujuan menjalankan seluruh layanan

```
chirpstack@chirpstack-vm:~/chirpstack-docker$ docker compose up -d
[+] up 7/7
 ✔Container chirpstack-docker-chirpstack-gateway-bridge-basicstation-1 Running                0.0s
 ✔Container chirpstack-docker-chirpstack-1                             Running                0.0s
 ✔Container chirpstack-docker-chirpstack-gateway-bridge-1              Running                0.0s
 ✔Container chirpstack-docker-chirpstack-rest-api-1                    Running                0.0s
 ✔Container e187f75dea08_chirpstack-docker-mosquitto-1                 Started                0.5s
 ✔Container 41f4c546813d_chirpstack-docker-postgres-1                  Started                0.5s
 ✔Container 3c6519ae8031_chirpstack-docker-redis-1                     Started                0.5s
```

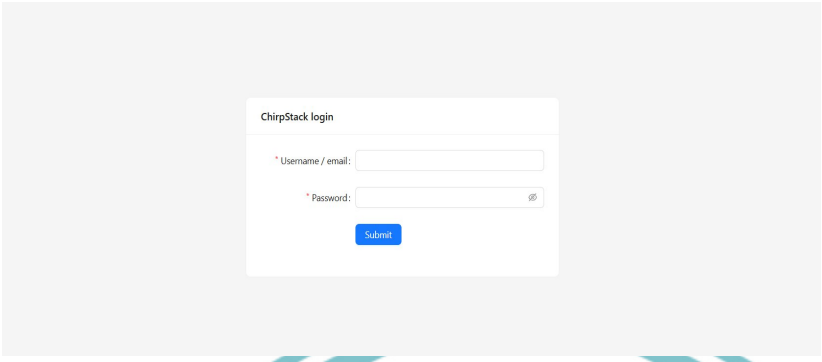
Gambar 3.57 Tampilan Terminal untuk Proses Pengaktifan Layanan *Docker*

- Kemudian untuk mengecek layanan sudah berhasil dengan *docker ps*

```
chirpstack@chirpstack-vm:~/chirpstack-docker$ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED        STATUS              PORT
S                                      NAMES
33e9d12466a7   chirpstack/chirpstack-rest-api:4       "/usr/bin/chirpstack…"   12 hours ago   Up 36 minutes       0.0.
0.0:8090->8090/tcp, [::]:8090->8090/tcp   chirpstack-docker-chirpstack-rest-api-1
70bec63b58d4   chirpstack/chirpstack:4                "/usr/bin/chirpstack…"   12 hours ago   Up 36 minutes       0.0.
0.0:8080->8080/tcp, [::]:8080->8080/tcp   chirpstack-docker-chirpstack-1
484fd46d6ccb   chirpstack/chirpstack-gateway-bridge:4   "/usr/bin/chirpstack…"   12 hours ago   Up 36 minutes       0.0.
0.0:3001->3001/tcp, [::]:3001->3001/tcp   chirpstack-docker-chirpstack-gateway-bridge-basicstation-1
5de8df6d773c   chirpstack/chirpstack-gateway-bridge:4   "/usr/bin/chirpstack…"   12 hours ago   Up 36 minutes       0.0.
0.0:1700->1700/udp, [::]:1700->1700/udp   chirpstack-docker-chirpstack-gateway-bridge-1
41f4c546813d   postgres:14-alpine                     "docker-entrypoint.s…"   12 hours ago   Up About a minute   5432
/tcp                                   41f4c546813d_chirpstack-docker-postgres-1
3c6519ae8031   redis:7-alpine                         "docker-entrypoint.s…"   12 hours ago   Up About a minute   6379
/tcp                                   3c6519ae8031_chirpstack-docker-redis-1
e187f75dea08   eclipse-mosquitto:2                    "/docker-entrypoint.…"   12 hours ago   Up About a minute   0.0.
0.0:1883->1883/tcp, [::]:1883->1883/tcp   e187f75dea08_chirpstack-docker-mosquitto-1
```

Gambar 3.58 Tampilan Terminal Berjalannya Setiap Layanan

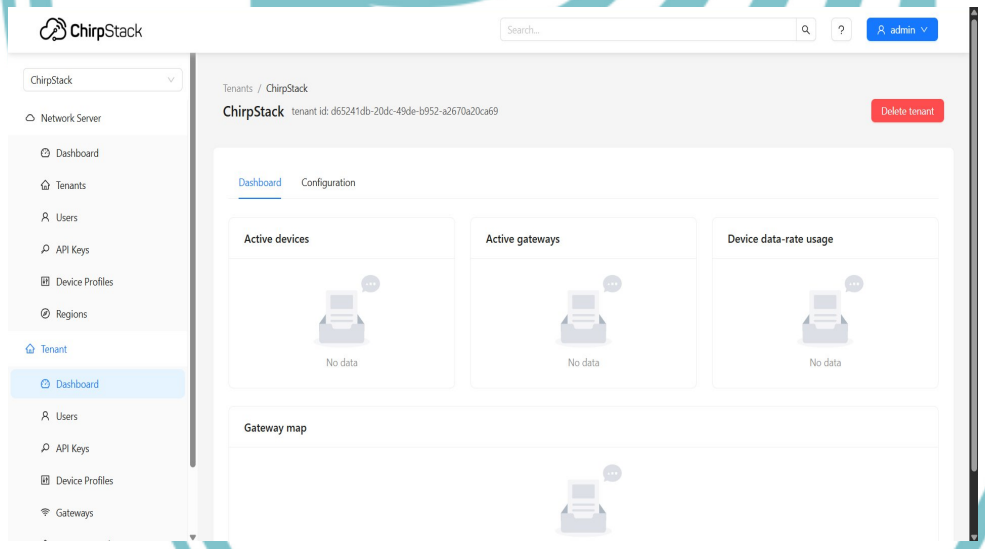
- Kemudian mengakses Chirpstack melalui browser dengan *http://IP_VM:8080*



Gambar 3.59 Tampilan Browser mengakses Chirpstack

Secara default untuk mengakses laman Chirpstack dengan memasukkan username dan passwordnya admin.

- Tampilan halaman akan berubah menunjukkan dash*board* dari Chirpstack untuk mengetahui perangkat aktif, *gateway* yang aktif dan penggunaan *data rate* dari perangkat.



Gambar 3.60 Tampilan Awal Halaman Chirpstack

3.2.6. Realisasi Konfigurasi *Gateway* pada Platform SenseCAP

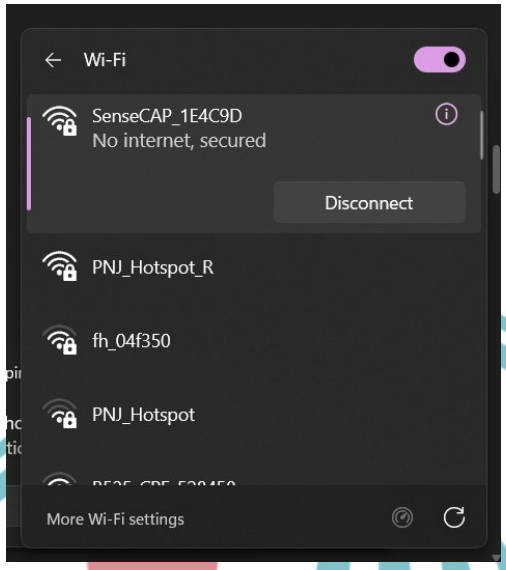
- Persiapkan antenna LoRa disambung ke port ANT sebelum *gateway* dihidupkan. Perlu diingat jangan menyalakan *gateway* jika antena belum tersambung ke perangkat karena akan menyebabkan kerusakan terhadap modul RF.
- Colokkan adaptor power ke *gateway*. Tunggu hingga LED berwarna biru berkedip atau menyala solid.

Gambar 3.61 Tampilan LED sudah berwarna biru solid

- Tekan dan tahan tombol di badan *gateway* selama ±5 detik hingga LED biru berkedip lambat. *Gateway* akan menyebarkan SSID hotspot.

Gambar 3.62 Tombol *Gateway* untuk Konfigurasi

- Di laptop, scan Wi-Fi dan sambungkan ke SSID bernama SenseCAP_XXXXXX (6 digit terakhir adalah bagian dari MAC address *gateway*). Password default: 12345678.

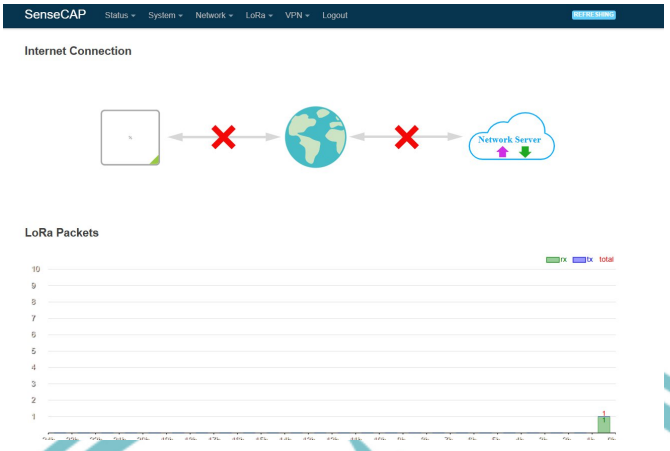


Gambar 3.63 Tampilan Pembacaan Jaringan WiFi

- Mengakses laman dengan mengunjungi melalui http://192.168.168.1. Untuk bisa menggunakan fitur pengaturan perlu memasukkan username default yaitu *admin* dan password yang dapat dilihat di belakang perangkat *Gateway SenseCAP M2*.

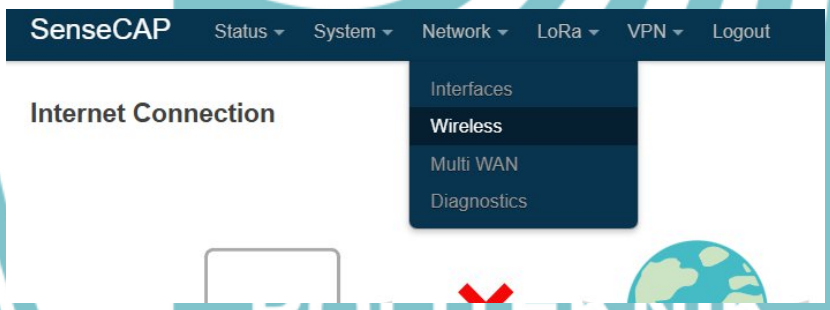
Gambar 3.64 Tampilan Belakang dari *Gateway*

- Ketika berhasil mengakses lama akan muncul tampilan konektivitas antara *gateway* ke internet untuk tersambung ke *network server*.



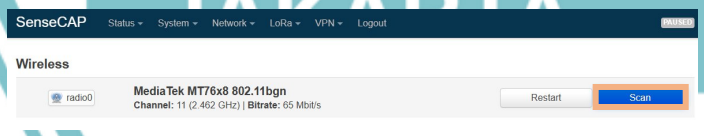
Gambar 3.65 Tampilan Dash*board* SenseCAP

- *Gateway* perlu terhubung ke jaringan yang sama dengan *server* Chirpstack agar bisa berkomunikasi. Di laman SenseCAP, untuk menemukan menu Wireless klik menu Network dan pilih Wireless.



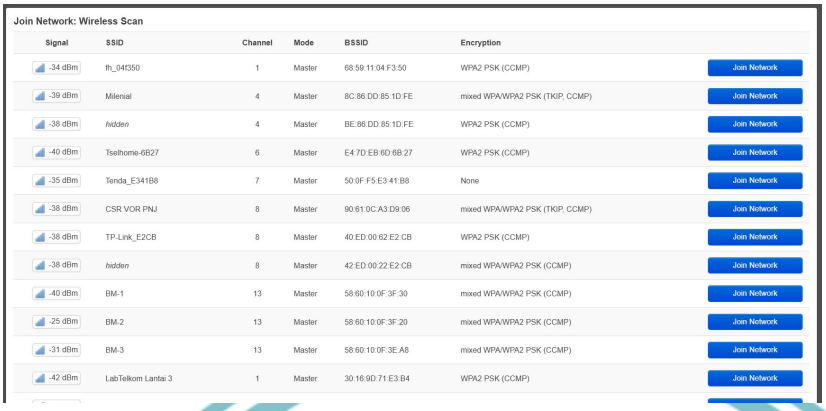
Gambar 3.66 Navigasi Halamanuntuk Wireless

- Klik tombol Scan pada interface radio0.



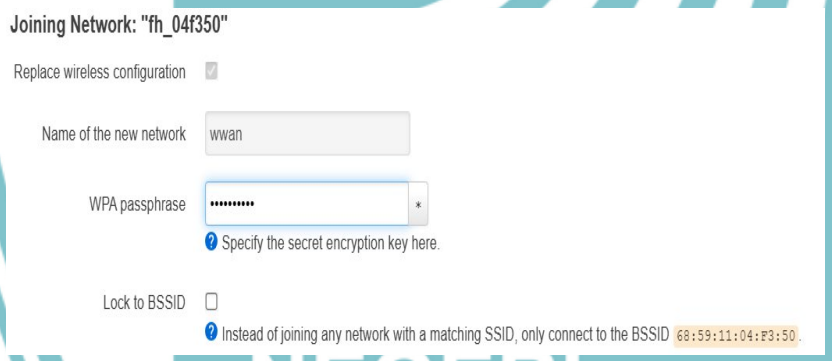
Gambar 3.67 Navigasi Tombol Scanning

- Tunggu hingga daftar SSID muncul.



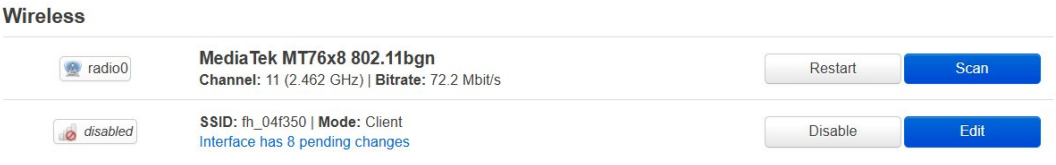
Gambar 3.68 Tampilan Hasil Pembacaan Wireless Scan

- Konektivitas *Gateway* yang digunakan itu adalah fh_04f350, lalu tekan Join Network nanti akan diarahkan ke halaman untuk mengisi password dari fh_04f350 itu sendiri dan tekan tombol submit yang berwarna hijau yang letaknya di kanan bawah seperti pada Gambar 3.69.



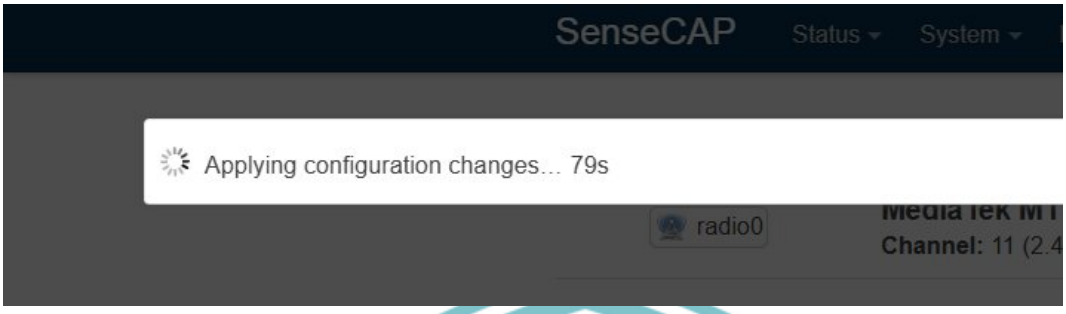
Gambar 3.69 Proses Pengisian Password Network

- Tampilan halaman akan seperti pada Gambar 3.70, dimana koneksi bertambah 1 yaitu jaringan yang disambungkan melalui proses pembacaan jaringan.



Gambar 3.70 Penambahan Konektivitas *Gateway*

- Lalu ubah jaringan ke fh_04f350 dengan tekan tombol save & apply



Gambar 3.71 Tampilan Loading Perubahan Jaringan

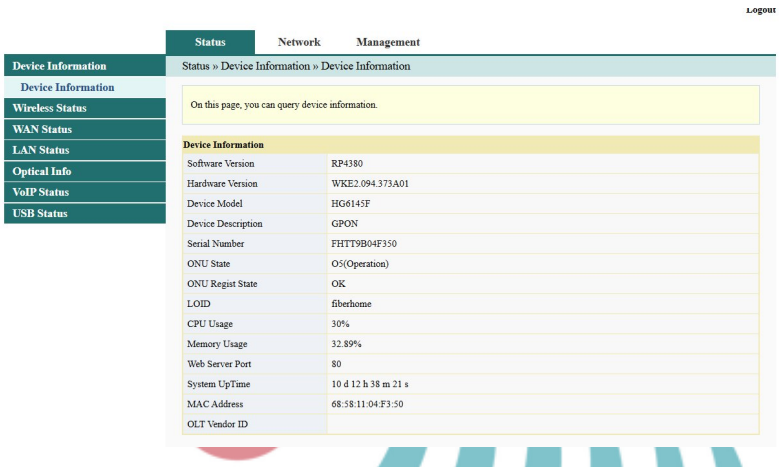
Gambar 3.72 Tampilan Setelah Konfigurasi Berubah

Sekarang, karena *gateway* sudah beralih fungsi menjadi Client, *gateway* beralih dari mode access point dan mencoba mencari alamat IP baru dari router/WiFi kampus (fh_04f350).

- Dengan berubahnya arah konektivitas dari *gateway* maka untuk mengakses laman kembali untuk kebutuhan konfigurasi lebih lanjut perangkat laptop harus bergabung ke jaringan fh_04f350.
- Cari Alamat IP baru *Gateway* dengan membuka terminal atau command prompt lalu ketik perintah *ipconfig*.
- Buka browser, ketikkan alamat IP baru yang didapatkan di address bar (default *gateway*), lalu tekan Enter.
- Berdasarkan hasil yang ditampilkan ipconfig pada command prompt, ip untuk mengakses pada browser ialah 192.168.1.1

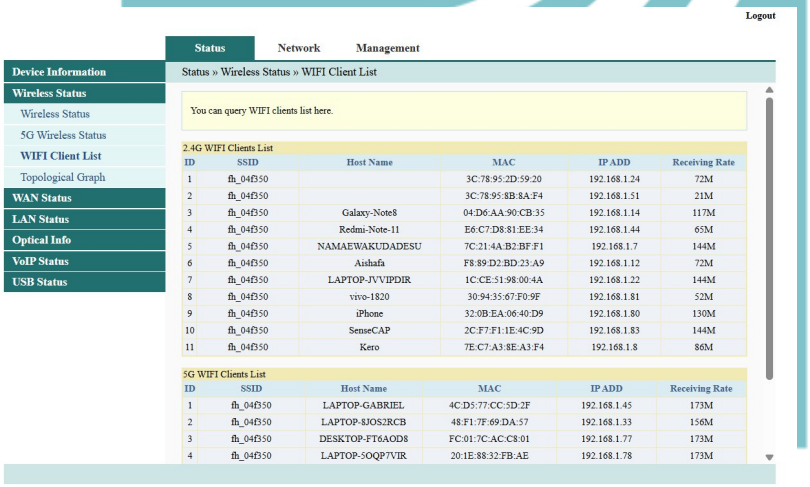
Gambar 3.73 Halaman Login Konfigurasi Router

- Masukkan username dan password yang sesuai dengan router yang digunakan oleh fh_04f350.
- Tampilan dash*board* ketika berhasil mengakses



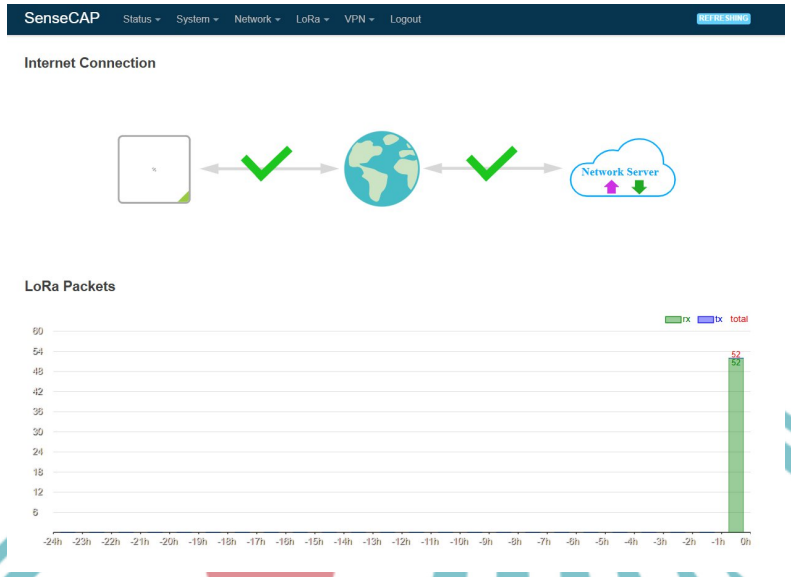
Gambar 3.74 Tampilan Halaman Dash*board*

- Kemudian untuk mengetahui ip dari *gateway* setelah terhubung ke dalam jaringan dengan mengakses wireless status lalu tekan WIFI Client List



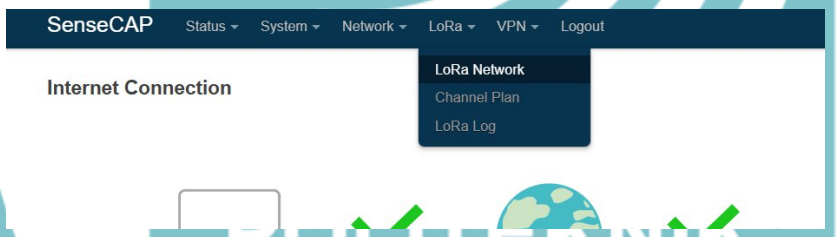
Gambar 3.75 Tampilan Halaman Device yang Tergabung di Jaringan

- Pada Gambar 3.75, Perangkat *gateway* sudah bergabung berdasarkan Namanya SenseCAP, dan IP yang diakses untuk mengkonfigurasi lebih lanjut adalah 192.168.1.83. Masukkan kembali username dan password.
- Tampilannya koneksi internet pada lama *gateway* akan berubah menjadi tanda ceklis yang menandakan bahwa *gateway* berhasil terhubung ke internet dan tersambung ke *network server* nantinya Chirpstack.



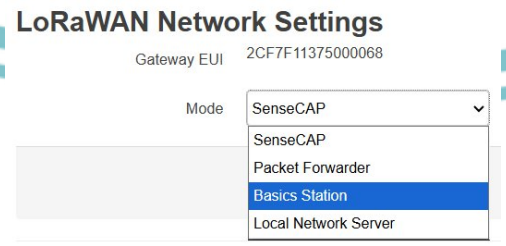
Gambar 3.76 Tampilan Perubahan Sambungan Koneksi *Gateway*

- Buka Menu Konfigurasi LNS, klik submenu LoRa dan pilih LoRa Network untuk mengkonfigurasi fungsi dari *gateway*



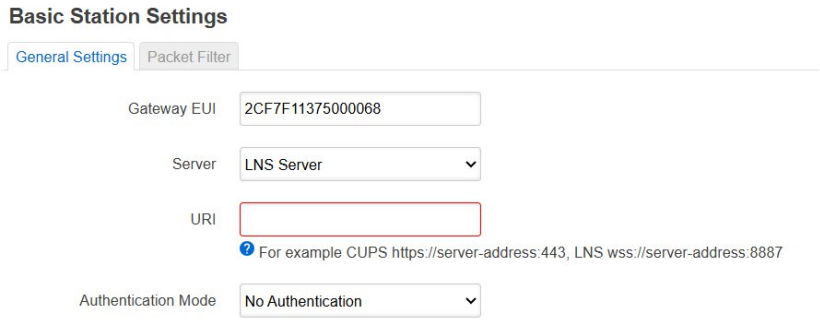
Gambar 3.77 Navigasi Menuju Halaman LoRa Network

- Mode *gateway* diubah dari *Packet forwarder* ke Basics Station untuk mendukung koneksi WebSocket (ws://) yang lebih stabil dengan *server* Chirpstack on-premise.



Gambar 3.78 Perubahan Fungsi *Gateway*

- Kolom URI diisi dengan alamat *server* Chirpstack yang berjalan pada VM dengan port Basics Station 3001.



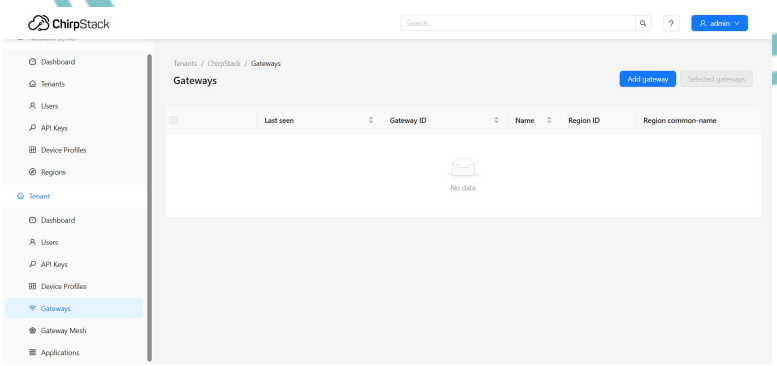
Gambar 3.79 Formulir Pengisian Alamat *Server*

- Setelah seluruh parameter konfigurasi pada halaman Basic Station Settings terisi dengan benar, langkah selanjutnya adalah menekan tombol Save & Apply untuk menyimpan konfigurasi.
- Ketika sudah menekan tombol save & apply maka akan muncul tampilan loading sebagai indikator bahwa sistem sedang memproses konfigurasi yang dapat dilihat pada Gambar 3.80.

Gambar 3.80 Tampilan Loading Proses Konfigurasi

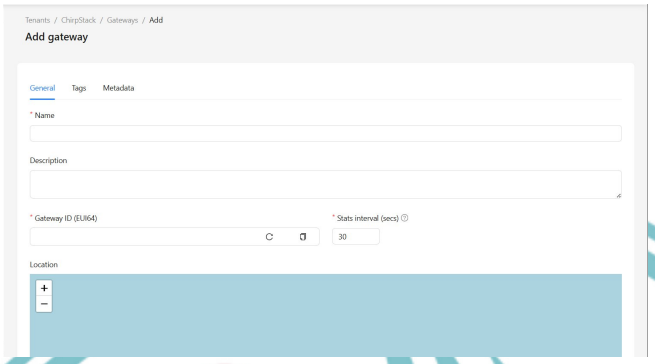
3.2.7. Realisasi Pendaftaran *Gateway* ke dalam Chirpstack

- Akses Web UI Chirpstack: Buka browser, lalu akses alamat: *http://192.168.1.54:8080* sebagai ip lokal yang dikonfigurasi pada VM.
- Masukkan username dan password default (admin/admin)
- Buka Menu *Gateway*s: Pada menu di sisi kiri Tenants, cari dan klik menu *Gateway*s



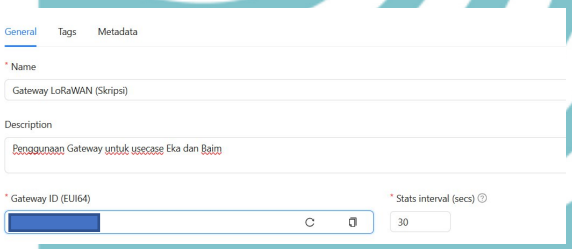
Gambar 3.81 Halaman Dash*board* Chirpstack

- Tambah *Gateway* Baru: Klik tombol + CREATE (atau Add *gateway*)
- Setelah memilih tampilan laman akan diarahkan seperti Gambar 3.82.



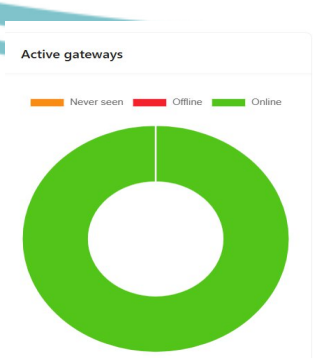
Gambar 3.82 Tampilan Halaman Penambahan Gateway

- Isi formulir dengan benar agar perangkat tidak mengalami kesalahan pengiriman atau gagal untuk bergabung ke dalam Chirpstack.



Gambar 3.83 Pengisian Data Penting Untuk Penyambungan

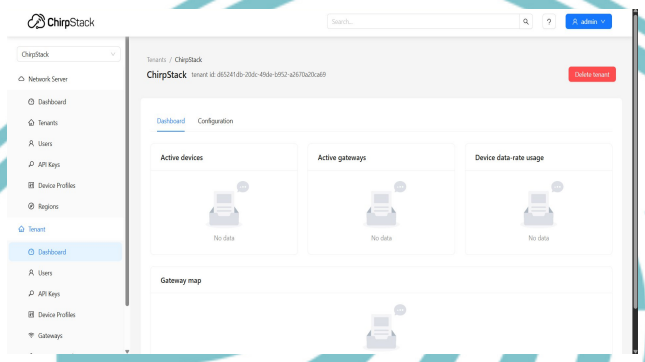
- Tekan submit sebagai bentuk konfirmasi bahwa pengisian formulir dilakukan dengan tepat.
- Kemudian halaman akan diarahkan kembali ke dash*board*, dan akan muncul indikator bahwasannya *gateway* sudah terhubung ke *network server* Chirpstack. Indikator akan berwarna hijau dengan keterangan online.



Gambar 3.84 Indikator *Gateway* Berhasil Disambungkan

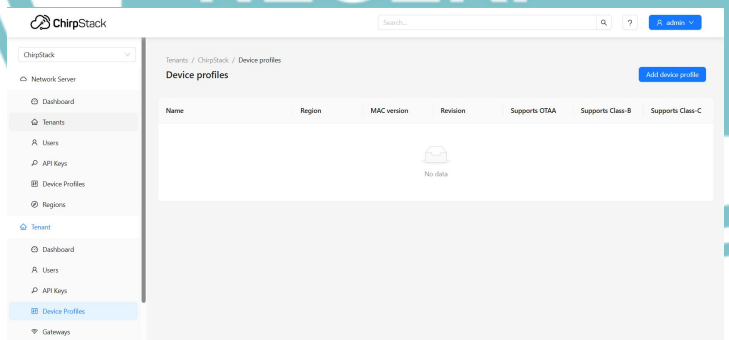
3.2.8. Realisasi Pengelompokkan *End device* ke Dalam Chirpstack

- Memastikan bahwa *server* VM sudah menyala terlebih dahulu agar layanan dan proses login ke Chirpstack Web UI dapat dilakukan dengan benar.
- Kunjungi Laman Chirpstack melalui *http://IP_VM_:8080*, lalu login ke dalam layanan dengan kembali memasukkan username dan password (admin/admin)
- Tampilan halaman akan berubah menunjukkan dash*board* dari Chirpstack



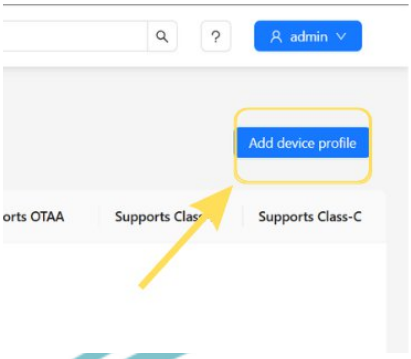
Gambar 3.85 Tampilan Dash*board* Chirpstack

- Pada menu sebelah kiri terdapat Tenant yang harus kita klik bagian Device Profilesnya. Pada halaman device profiles terdapat informasi yang dapat diketahui pengguna ketika keseluruhan proses penyambungan dilakukan diantaranya name atau nama profile perangkat, kemudian ada region atau wilayah yang dapat diatur sesuai frekuensi plan, MAC Version, Revision, Supports OTAA, Class B dan Class C.



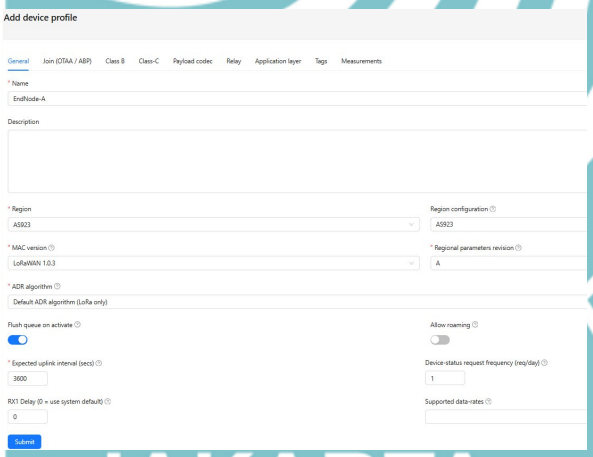
Gambar 3.86 Halaman Device Profile

- Untuk menambahkan perangkat bisa menekan tombol pada bagian kanan atas yang bertuliskan “Add Device Profiles”.



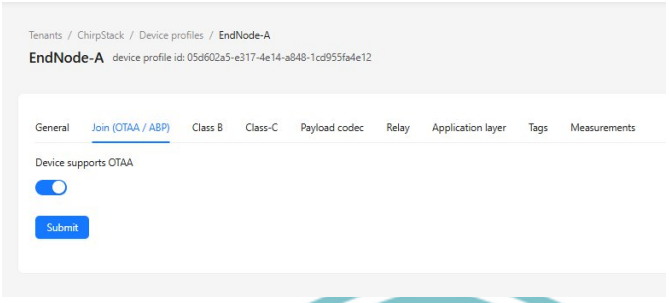
Gambar 3.87 Navigasi Add Device Profile

- Selanjutnya didalam halaman Add Device Profiles akan berisi formulir yang harus diisi dengan benar sesuai dengan arah penelitian diantaranya mengisi *name, region, MAC version, regional parameters,* dan *ADR algorithm*. Setelah semua formulir sudah diisi maka dapat menekan tombol submit sebagai bentuk konfirmasi yang dapat dilihat pada Gambar 3.86.



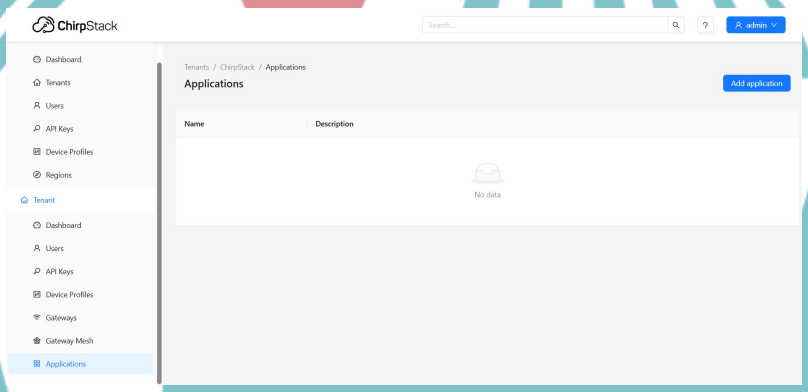
Gambar 3.88 Tampilan Formulir Device Profiles

- Disebelah menu General sebagai tempat pengisian formulir data perangkat, tekan Join OTAA/ABP untuk sistem mengetahui bahwa akan ada paket *uplink* yang dikirim oleh *end device* untuk bisa bergabung ke dalam sistem LoRaWAN. Untuk menyetujui proses Join OTAA perangkat dapat menekan tombol *Submit*.



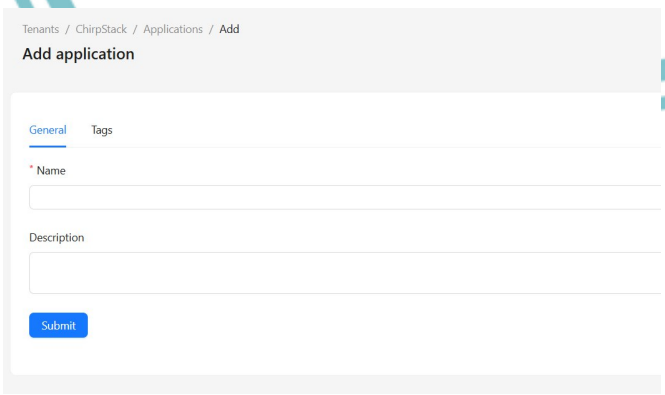
Gambar 3.89 Tampilan Halaman Join OTAA

- Proses selanjutnya dalam menyambungkan *end device* ke sistem layanan adalah dengan membuat *Application* atau tempat mengelompokkan perangkat sejenis. Menu *Application* terletak di sudut kiri paling bawah dari bagian Tenants.



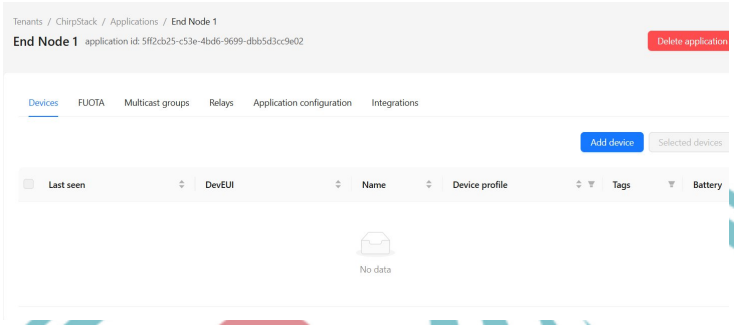
Gambar 3.90 Tampilan Application

- Untuk menambahkan *Application* pada Chirpstack dengan menekan tombol "*Add application*" yang berada di bagian kanan atas.
- Halaman akan diarahkan ke formulir yang harus di isi diantaranya itu Name dan Description (optional).



Gambar 3.91 Tampilan Formulir Penambahan Application

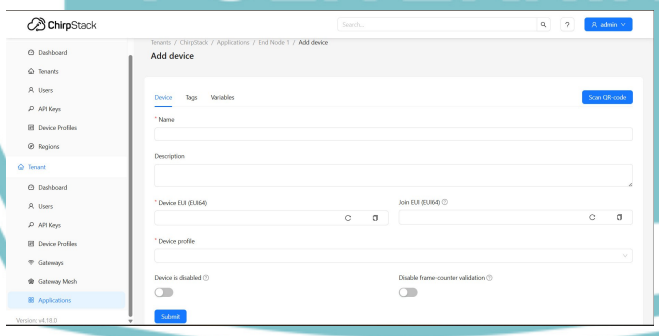
- Simpan formulir yang sudah diisi dan nantinya halaman diarahkan menuju tempat pengumpulan perangkat sesuai kebutuhannya dengan memperhatikan Name, Deveui, Last Seen, Device Profile, Tags, dan Battery.



Gambar 3.92 Tampilan Pengelompokkan Perangkat Akhir

3.2.9. Realisasi Pendaftaran *End device* ke Chirpstack

- Proses awal untuk melakukan pendaftaran perangkat akhir ke dalam sistem dengan mempersiapkan perangkatnya yaitu Heltec WiFi LoRa V3 dan *Applications* yang sebelumnya telah dibentuk.
- Pada halaman Applications yang sudah didaftarkan akan muncul nama yang telah di isi pada formulir, pilih nama tersebut dan halaman akan diarahkan pada pendaftaran perangkat akhir.
- Selanjutnya pilih *"Add Device"*

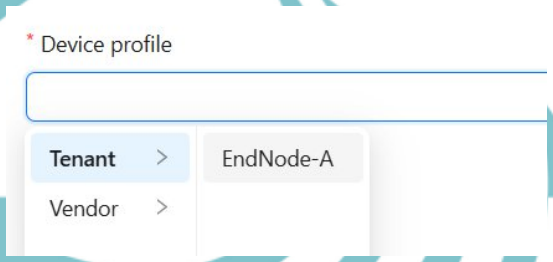


Gambar 3.93 Tampilan Halaman Pendaftaran Perangkat Akhir

- Lakukan pengisian pada formulir diantaranya Name, Device EUI (EUI64), Join EUI (EUI64) dan Device Profile. Untuk Device EUI dan Join EUI akan mengenerate secara otomatis karakter hexadesimalnya sebagai informasi yang harus ditambahkan pada sketch koding. Perilaku generate ini dapat dilakukan dengan menekan tombol refresh.

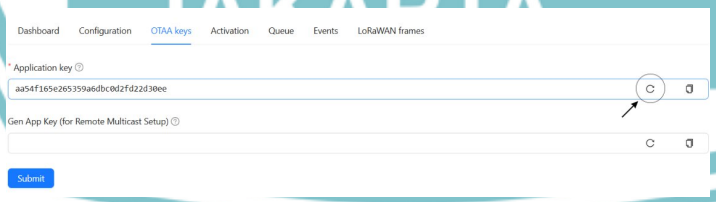
Gambar 3.94 Navigasi untuk Tombol Refresh

- Untuk pemilihan Device Profile bisa menekan kolom nantinya akan terdapat pilihan Tenant dan bisa menentukan dimana perangkat ini akan ditempatkan dan dikelompokkan.



Gambar 3.95 Tampilan Penentuan Pengelompokkan *End device*

- Ketika seluruh proses pengisian formulir pendaftaran perangkat akhir pada *Applications* sudah selesai dilakukan, klik bagian submit.
- Langkah berikutnya ketika pensubmitan formulir, halaman akan diarahkan di bagian OTAA keys. Halaman ini akan mengenerate kembali karakter hexadesimal untuk kunci enkripsi perangkat dengan tujuan bergabung ke dalam perangkat dengan cara menekan tombol refresh seperti perilaku Device EUI dan Join EUI.



Gambar 3.96 Tampilan Pengenerate Karakter Join OTAA

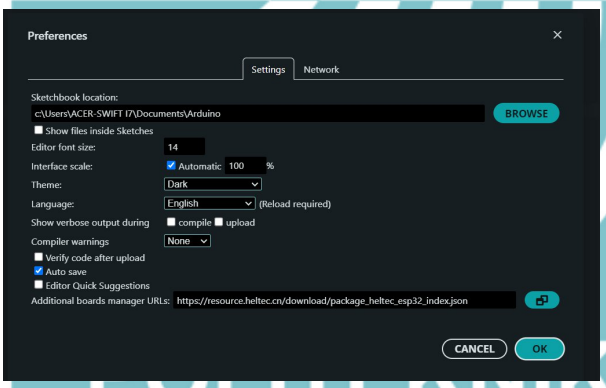
- Pilih submit untuk mengkonfirmasi bahwa proses ini sudah selesai dilakukan.

3.2.10. Realisasi Penyambungan *End device*

- Mempersiapkan perangkat *End device* Heltec WiFi LoRa V3 dan memastikan layanan Chirpstack sudah aktif.

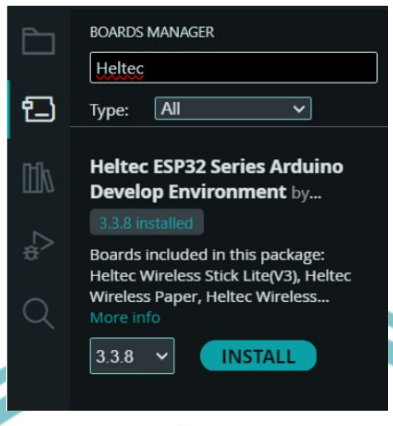
- Menginstal driver resmi Silicon Labs berupa file ZIP kemudian mengekstraknya agar perangkat laptop dapat mengenali dan berkomunikasi dengan chip USB-to-serial pada Heltec V3. Di dalam folder tersebut, terdapat file berekstensi .exe. Untuk Windows 64-bit jalankan file *CP210xVCPInstaller_x64.exe.*
- Kemudian menambahkan *board* Heltec melalui alamat repository online yang berisi informasi tentang *board* Heltec ESP32.

  https://resource.heltec.cn/download/package_heltec_esp32_index.json
- Membuka Platform koding sketch seperti *Arduino IDE* untuk melakukan beberapa *install*asi dan konfigurasi terhadap perangkat *end device*.
- Membuka File yang posisi berada pada kiri atas dan memilih *Preferences* untuk menyalin alamat repository *board* Heltec.



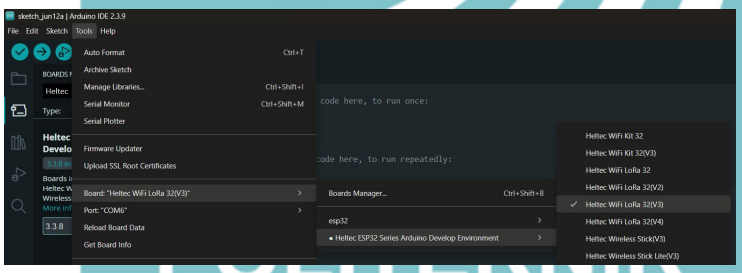
Gambar 3.97 Tampilan *Preferences Arduino IDE*

- Secara otomatis, *Arduino IDE* akan mengunduh alamat yang disematkan pada kolom Additional *Board Manager* URL's
- Proses berikutnya adalah mengunduh semua file yang diperlukan (*toolchain*, *compiler*, *library*, dan file konfigurasi) untuk memprogram *board* Heltec ESP32 dengan membuka Tools yang posisinya ada di sebelah kiri atas dari help. Kemudian Pilih *Board* dan cari *Boards Manager*, ketik kata kunci "Heltec ESP32". Sistem akan menampilkan paket bernama "Heltec ESP32 Series Dev-*boards*". Klik tombol *Install* yang berada di sebelah kanan nama paket. Jangan menutup *Arduino IDE* selama proses berlangsung.



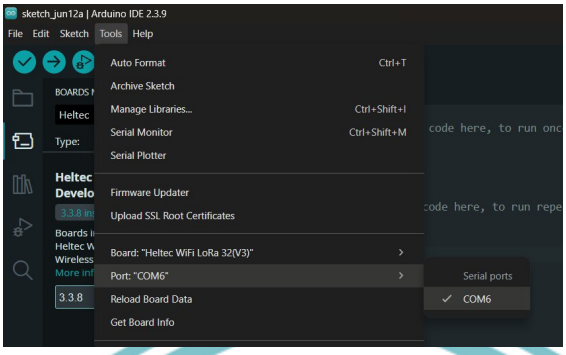
Gambar 3.98 Tampilan Pencarian *Board* Heltec

- Setelah *library board* Heltec ESP32 terinstal, *Arduino IDE* masih belum tahu *board* mana yang gunakan dari sekian banyak *board* yang didukung (ada Heltec V2, V3, WiFi Kit, Wireless Shell, dll). Pemilihan *board* heltec dapat dilakukan dengan cara memilih tools, tekan *board* dan pilih Heltec ESP32 Series Dev-*boards* dengan nama *board*nya Heltec WiFi LoRa 32 (V3).



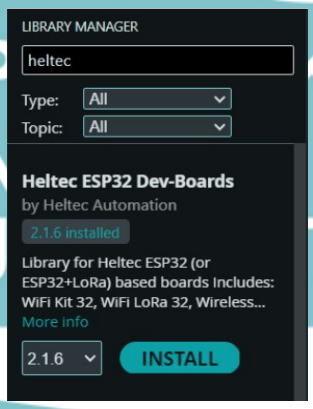
Gambar 3.99 Pemilihan dan Pengenalan *Board* Heltec WiFi LoRa V3

- Menentukan jalur komunikasi antara *Arduino IDE* dan *board* Heltec V3 dengan menghubungkan Heltec V3 ke komputer menggunakan kabel USB-C yang mendukung data. Buka menu Tools dan pergi ke Port. Pilih port yang sesuai dengan Heltec V3. Di Windows, biasanya port yang benar memiliki nama "COM" dengan nomor yang cukup besar (COM3, COM4, COM5) dan tidak ada di daftar sebelum Heltec dicolokkan.



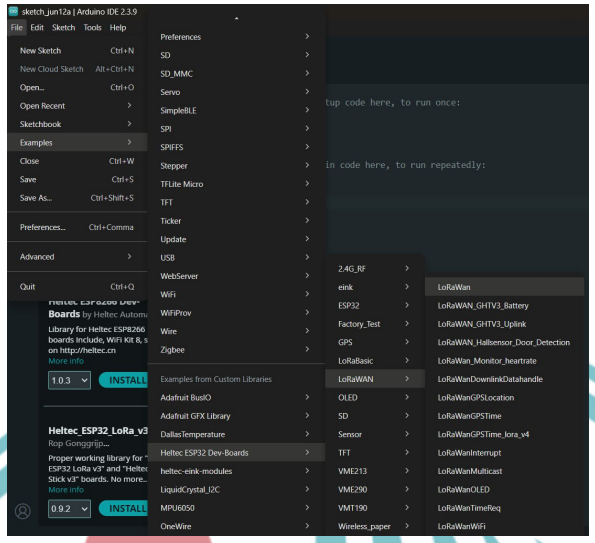
Gambar 3.100 Tampilan Pemilihan Port

- Langkah berikutnya adalah menginstal Heltec ESP32 Extended *Library*. *Library* extended inilah yang menyediakan semua contoh program (examples) dan fungsi pendukung (helper functions) untuk perangkat Heltec. Dengan melakukan proses menginstal *library* akan membantu saat proses percobaan karena isi dari *library* sangat penting saat proses #define dan kode contoh dapat berjalan.
- Navigasikan ke menu Sketch lalu pilih Include *Library* dan Manage Libraries. Setelah jendela *Library Manager* terbuka, ketikkan kata kunci "Heltec ESP32", lalu klik tombol *Install*.



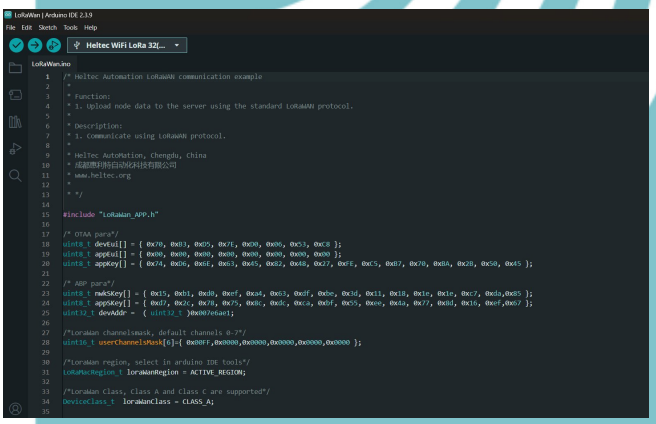
Gambar 3.101 Tampilan *Library Manager*

- Setelah instalasi selesai, sangat disarankan untuk merestart *Arduino IDE*. Hal ini penting dilakukan agar IDE dapat membaca ulang daftar *library* yang baru saja ditambahkan. Setelah restart, kode contoh LoRaWAN dapat diakses dengan membuka menu File > Examples > Heltec ESP32 > LoRaWAN.



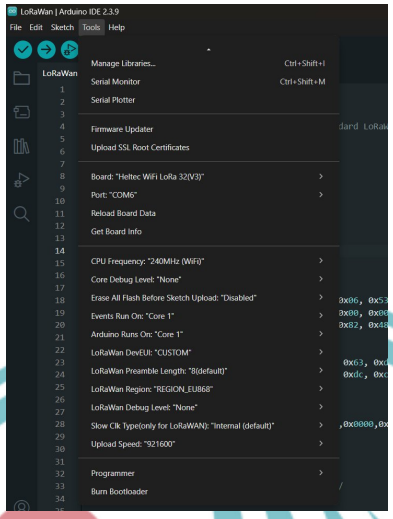
Gambar 3.102 Pengaksesan Example LoRaWAN Untuk Heltec WiFi V3

- Dengan membuka file contoh LoRaWAN, sebuah jendela baru akan terbuka berisi kode lengkap yang sudah ditulis oleh Heltec.



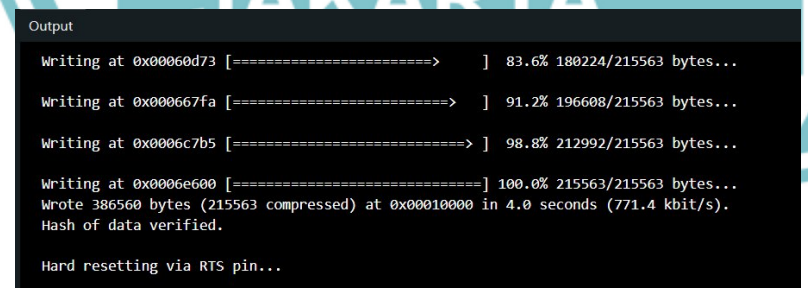
Gambar 3.103 Tampilan Example di Jendela Arduino Baru

- Sebelum melakukan sketching terhadap kodingan contoh LoRaWAN lebih mendalam, pengaturan parameter LoRaWAN di *Arduino IDE* perlu dilakukan dengan tujuan sketching dan frekuensi plan bisa tertuju dengan benar. Pengaturan dapat dilakukan melalui dropdown Tools. Perhatikan untuk Regionnya, rubah menjadi “REGION_AS923”.



Gambar 3.104 Tampilan Tools Pengaturan LoRaWAN

- Setelah merubah region pada menu Tools, kode pada example difokuskan untuk merubah bagian DevEUI, AppEUI, dan AppKey nya. Karena keseluruhan sketch pada example sudah dibuat berdasarkan default perangkat. Kode DevEUI, AppEUI dan AppKey didapatkan dari proses pendaftran perangkat akhir ke Chirpstack.
- Sesudah memasukkan karakter hexadesimal ke dalam sketch example LoRaWAN dan memastikan tidak ada kesalahan saat penginputan, berikutnya mengeksekusi program tersebut melalui tombol Upload (ikon panah kanan di pojok kiri atas) atau tekan Ctrl+U. Perhatikan output di area hitam di bawah jendela *Arduino IDE*. Jika berhasil, Anda akan melihat pesan *Leaving... Hard resetting via RTS pin... Done uploading* yang dapat dilihat pada Gambar 3.105.



Gambar 3.105 Tampilan Serial Monitor Setelah Upload Sketch

3.2.11. Realisasi Pembuatan Peran Pengguna Chirpstack (*Owner, Editor* dan *Viewer*)

- Login ke Web UI Chirpstack melalui browser dengan mengakses alamat *http://IP_VM:8080* sebagai *Default* Admin.
- Pada menu di sisi kiri, akses menu *Users* pada jendela *Network Server*.
- Klik tombol *Add User* dan Isi formulir pendaftaran pengguna dengan memasukkan alamat email pengguna yang valid (misalnya: *owner@chirpstack.com*, *editor@chirpstack.com*, *viewer@chirpstack.com*) dan mengatur kata sandi. Pastikan opsi *Is Active* dalam keadaan aktif.
- Klik tombol *Submit* untuk menyimpan akun pengguna yang telah dibuat.
- Setelah akun pengguna berhasil dibuat, pada menu yang sama, akses menu *Tenants*.
- Di dalam halaman tenant, cari menu *Tenant Users* kemudian untuk menambahkan pengguna yang terdiri dari 3 peran pengguna dapat menekan tombol yang bertuliskan *Add tenant user*.
- Masukkan alamat *email* pengguna yang telah dibuat sebelumnya pada kolom yang tersedia. Pada bagian *Roles*, terdapat beberapa opsi yang menentukan peran pengguna dalam tenant tersebut.
- Untuk peran *Owner* dapat memilih *User is tenant admin;* peran *Editor* dapat memilih *User is gateway admin* dan *User is device admin*; sedangkan untuk peran *Viewer* setelah memasukkan *Email* cukup langsung mensubmitnya.
- Untuk memverifikasi bahwa konfigurasi role telah berhasil diterapkan, lakukan pengujian dengan cara login menggunakan akun pengguna yang baru saja dikonfigurasi dan pastikan hak aksesnya sesuai dengan role yang diberikan. Pengguna dengan role *Viewer* hanya dapat melihat data tanpa tombol pengelolaan, *Editor* dapat melakukan perubahan konfigurasi perangkat, sedangkan *Owner* memiliki akses penuh terhadap seluruh fitur aplikasi.

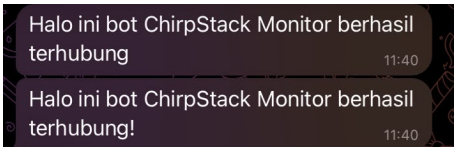
3.2.12. Realisasi Pembuatan Sistem Peringatan Bagi Layanan ChirpStack

- Tahap awal dalam membuat sistem peringatan adalah dengan pembuatan bot melalui *platform Telegram* sebagai media notifikasi kepada pengguna. Pembuatan bot melalui akun resmi @BotFather.
- Sistem @BotFather saat awal dikunjungi akan memberikan beberapa petunjuk kebutuhan yang akan ditujukkan bagi pengguna. Pembuatannya dengan menuliskan perintah */newbot.*
- Kemudian BotFather akan meminta pengguna untuk memberikan nama untuk bot yang akan digunakan, setelah proses pemberian nama sudah dilakukan, BotFather memberikan sebuah token API yang berfungsi sebagai kredensial autentikasi setiap kali bot melakukan permintaan ke Telegram Bot API.
- Langkah berikutnya adalah memperoleh Chat ID, yaitu identitas unik dari akun Telegram penerima notifikasi. Chat ID ini didapatkan dengan cara mengirimkan pesan ke bot yang telah dibuat.
- Setelah itu, buka browser dan akses URL berikut: *https://api.telegram.org/bot TOKEN/getUpdates* (ganti TOKEN dengan token yang didapat dari BotFather).
- Untuk memastikan bot telah terhubung dengan benar, dilakukan pengujian awal dengan mengirim pesan uji coba melalui pengaksesan URL yang disertai dengan TOKEN beserta Chat ID yang sudah didapatkan.

{"ok":true,"result":[{"message_id":6,"from":{"id":8862862889,"is_bot":true,"first_name":"Chirpstack_Monitor","username":"chirpstack_alert_bot"},"chat":{"id":7873579389,"first_name":"firli","type":"private"},"date":1783572019,"text":"Halo ini bot ChirpStack Monitor berhasil terhubung!"}]}

Gambar 3.106 Tampilan Verifikasi Bot Berhasil Terhubung

- Bentuk verifikasi berikutnya jika bot yang dibuat telah berhasil dan dapat digunakan akan muncul pesan di aplikasi Telegram di perangkat telepon gengam seperti pada Gambar 3.107.



Gambar 3.107 Tampilan Pesan dari Aplikasi Telegram

- Setelah bot dipastikan aktif, tahap selanjutnya adalah merancang script pemantauan yang diberi nama chirpstack-monitor.sh dan ditempatkan pada direktori */usr/local/bin/*. Script ini bertugas memeriksa status empat kontainer Docker yang menyusun layanan ChirpStack, yaitu kontainer ChirpStack itu sendiri, Mosquitto sebagai broker MQTT, PostgreSQL sebagai basis data, serta Redis sebagai penyimpanan sementara. Script dapat dilihat pada halaman Lampiran.
- Agar script pemantauan dapat berjalan secara terus-menerus tanpa henti dan tetap aktif meskipun terjadi kegagalan pada proses itu sendiri, script tidak dijadwalkan melalui crontab, melainkan dikonfigurasi sebagai layanan sistem (systemd service). Konfigurasi dilakukan dengan membuat berkas layanan pada direktori /etc/systemd/system/chirpstack-monitor.service, yang mendefinisikan bahwa layanan akan dijalankan setelah layanan Docker aktif dan akan otomatis dijalankan kembali (restart) apabila proses berhenti karena sebab apa pun.
- Setelah berkas layanan dibuat, sistem perlu dimuat ulang menggunakan perintah *sudo systemctl daemon-reload*, kemudian layanan diaktifkan agar berjalan otomatis setiap kali server dinyalakan dengan *perintah sudo systemctl enable chirpstack-monitor.service*, dan dijalankan seketika itu juga dengan perintah *sudo systemctl start chirpstack-monitor.service*.
- Sebelum layanan dijalankan sebagai systemd service, script terlebih dahulu diberikan izin eksekusi dengan menjalankan perintah: *sudo chmod +x /usr/local/bin/chirpstack-monitor-realtime.sh*
- Untuk memverifikasi bahwa alert benar-benar berfungsi saat terjadi kegagalan adalah dengan melakukan simulasi sederhana dengan mematikan salah satu container ChirpStack sebagai bentuk percobaan. Gunakan script berikut: docker stop chirpstack-docker-mosquitto-1.
- pada pendekatan real-time ini notifikasi peringatan pada aplikasi Telegram akan muncul dalam hitungan detik setelah kontainer dihentikan, karena sistem langsung menangkap dan merespons peristiwa (event) perubahan status

kontainer tanpa perlu menunggu jadwal pemeriksaan berikutnya Bentuk pesan peringatan dapat dilihat di Gambar 3.108.

Gambar 3.108 Tampilan Pesan Peringatan Telegram

3.2.13. Realisasi Sistem Failover pada Virtual Machine Cadangan

- Tahap awal dalam merealisasikan sistem failover adalah dengan menyiapkan autentikasi SSH tanpa kata sandi antara VM utama dan VM cadangan, agar proses pengiriman data antar kedua VM dapat dilakukan secara otomatis tanpa memerlukan interaksi manual. Autentikasi ini dibuat dengan menjalankan perintah *ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_backup -N ""* pada VM utama, yang menghasilkan sepasang kunci (key pair) berupa kunci privat dan kunci publik.
- Kunci publik yang telah dihasilkan kemudian disalin ke VM cadangan menggunakan perintah *ssh-copy-id -i ~/.ssh/id_ed25519_backup.pub chirpstack@<IP_VM_CADANGAN>*, sehingga VM utama dapat melakukan autentikasi ke VM cadangan menggunakan kunci tersebut, tanpa perlu memasukkan kata sandi secara berulang setiap kali proses pengiriman data dilakukan.
- Keberhasilan konfigurasi autentikasi tersebut diuji dengan menjalankan perintah *ssh -i ~/.ssh/id_ed25519_backup chirpstack@<IP_VM_CADANGAN > "echo berhasil"*, di mana keberhasilan proses login tanpa diminta kata sandi menjadi indikator bahwa autentikasi telah dikonfigurasi dengan benar.
- Setelah autentikasi berhasil, script backup basis data (*chirpstack-backup.sh*) pada VM utama dimodifikasi dengan menambahkan perintah pengiriman salinan berkas backup secara otomatis ke VM cadangan menggunakan scp, sehingga setiap kali proses backup basis data selesai dijalankan, berkas hasilnya turut tersalin secara otomatis ke direktori penyimpanan pada VM cadangan.

- Mekanisme sinkronisasi tersebut diuji dengan menjalankan script backup secara manual pada VM utama, kemudian memverifikasi keberadaan berkas backup pada direktori tujuan di VM cadangan, guna memastikan proses pengiriman berjalan dengan baik.
- Pada VM cadangan, dilakukan penduplikasian struktur direktori proyek ChirpStack (*chirpstack-docker*) ke dalam sebuah direktori terpisah bernama *chirpstack-docker-failover*, guna menghindari benturan konfigurasi dengan layanan ChirpStack lain yang telah berjalan pada VM cadangan tersebut.
- Pada direktori proyek cadangan tersebut, dilakukan penyesuaian terhadap berkas *docker-compose.yml*, dengan mengubah seluruh port yang digunakan oleh layanan (ChirpStack web, REST API, MQTT broker, serta *Gateway Bridge*), agar tidak bertabrakan dengan port layanan lain yang sudah aktif berjalan pada VM cadangan.
- Selanjutnya dibuat sebuah script aktivasi bernama activate-failover.sh, yang berfungsi untuk menjalankan kontainer basis data (PostgreSQL, Redis, dan Mosquitto), memulihkan (restore) berkas backup basis data terbaru yang tersedia, kemudian menjalankan seluruh layanan ChirpStack menggunakan data hasil pemulihan tersebut.
- Script aktivasi tersebut dijalankan sebagai bentuk simulasi kondisi darurat, guna menguji apakah layanan ChirpStack cadangan dapat diaktifkan menggunakan data terbaru yang telah tersinkronisasi dari VM utama.
- Keberhasilan proses aktivasi diverifikasi dengan memeriksa status seluruh kontainer melalui perintah *docker compose -p chirpstack-failover ps*, memastikan seluruh layanan berjalan tanpa kendala.

Bab IV
PEMBAHASAN

Pada bab ini, proses pengujian untuk Sistem Layanan *Server* LoRaWAN On Premise akan dibagi menjadi 3 skenario pengujian yang menyinggung komponen penting pembangun Chirpstack. Dengan dilakukannya pengujian maka keseluruhan infrastruktur yang telah dibangun dapat diketahui apakah keseluruhan rangkaian perangkat, pengiriman data, pengenalan paket data lalu diteruskan ke *server*, dan penyimpanan data berhasil atau tidak.

Pengujian sistem akan berfokus pada pengujian lama waktu join perangkat ketika diawal inisiasi, pengujian memastikan bahwa data *uplink* dari *application server* berhasil dipublikasi ke *MQTT Broker*, dan pengujian terhadap *gateway* untuk mengetahui *gateway bridge* berhasil menerima paket dan diteruskan. Ketika proses pengujian sudah berhasil dilaksanakan, data-data yang diterima akan diolah kemudian akan dianalisa untuk mengetahui bagaimana kinerja sistem. Pelaksanaan pengujian diatas dilaksanakan berikut:

a. Lokasi : Gedung Laboratorium Telekomunikasi (Lingkungan Gedung G Lab Telkom PNJ)
b. Waktu : Juni 2026
c. Pelaksana : Rakesh Sharma Pramujio
d. Instruktur : Agus Wagyana, S.T., M.T.

4.1. Pengujian Join OTAA

4.1.1. Deskripsi Pengujian

Pengujian *Over The Air Activation* (OTAA) merupakan salah satu skenario pengujian yang dilakukan dengan menggunakan *end device Heltec WiFi LoRa 32 V3* (chipset SX1262) untuk bergabung ke dalam *network server* pada platform Chirpstack. Kondisi pengujian dilakukan dengan memastikan bahwa setiap percobaan join dimulai dari kondisi *cold start*, yaitu kondisi awal inisiasi perangkat (“esp_restart()”) di mana identitas perangkat berupa DevEUI, AppEUI (JoinEUI), dan AppKey sudah terdaftar pada Chirpstack namun perangkat belum pernah bergabung didalam lingkungan ChirpStack. Nilai

DevNonce dipertahankan melalui penyimpanan *Non-Volatile Storage* (NVS) pada setiap restart agar tidak ditolak oleh mekanisme *anti-replay protection* milik Chirpstack.

Parameter utama yang diamati pada pengujian ini adalah waktu join (*Join Duration*), yaitu selisih waktu antara pengiriman paket *Join Request* oleh *end device* dan penerimaan paket *Join Accept* dari *network server*, serta status keberhasilan join (*success rate*). Pengamatan paket LoRaWAN dilakukan melalui dash*board* Chirpstack dan serial monitor pada *Arduino IDE*. Pengujian utama dilakukan dengan memvariasikan nilai *Spreading factor* (SF7 hingga SF12, setara dengan DR5 hingga DR0) pada *Bandwidth* 125 kHz sesuai regional parameter AS923-2.

4.1.2. Prosedur Pengujian

Pengujian Join OTAA dilakukan untuk mengetahui kemampuan *network server* dalam melakukan proses autentikasi dan aktivasi perangkat LoRaWAN pada berbagai konfigurasi *Spreading factor* dan *Bandwidth*. Tahapan pengujian yang dilakukan adalah sebagai berikut:

1. Pengujian Join OTAA dilakukan untuk mengetahui kemampuan *network server* dalam melakukan proses autentikasi dan aktivasi perangkat LoRaWAN pada berbagai konfigurasi *Spreading factor* dan *Bandwidth*. Tahapan pengujian yang dilakukan adalah sebagai berikut:
2. Memastikan seluruh layanan pendukung pada *server* LoRaWAN, seperti *MQTT Broker*, *PostgreSQL*, *Redis*, dan Chirpstack berjalan dengan normal pada lingkungan *Docker*.
3. Memastikan *gateway SenseCAP M2* telah terhubung dengan *server* Chirpstack dan berada dalam status online pada dash*board* monitoring.
4. Membuat device profile dan application pada Chirpstack sesuai dengan spesifikasi *Heltec WiFi LoRa 32 V3*.
5. Menambahkan end-device ke dalam aplikasi Chirpstack dengan memasukkan parameter identitas perangkat berupa DevEUI, JoinEUI, dan AppKey yang sesuai dengan konfigurasi pada firmware.

6. Mengunggah program LoRaWAN berbasis OTAA (menggunakan pustaka *RadioLib*) ke perangkat *Heltec WiFi LoRa 32 V3* dan memastikan parameter DevEUI, JoinEUI, serta AppKey telah sesuai dengan data yang terdaftar pada Chirpstack.
7. Menonaktifkan fitur *Adaptive data rate* (ADR) pada konfigurasi end-device dan device profile di Chirpstack, kemudian menetapkan nilai *Spreading factor* secara manual sesuai dengan skenario pengujian yang akan dijalankan (SF7–SF12 pada BW 125 kHz).
8. Merestart perangkat end-device (esp_restart()) sehingga perangkat secara otomatis mengirimkan paket *Join Request* menuju *gateway* LoRaWAN menggunakan konfigurasi SF yang telah ditetapkan, dengan DevNonce yang dipertahankan melalui NVS.
9. Mengamati catatan komunikasi pada dash*board* Chirpstack untuk memverifikasi penerimaan paket *Join Request* oleh *network server*.
10. Mengamati respons *Join Accept* yang dikirimkan oleh *network server* kepada end-device melalui *gateway*.
11. Mencatat waktu pengiriman *Join Request* dan waktu penerimaan *Join Accept* melalui serial monitor *Arduino IDE*, kemudian menghitung *Join Duration* sebagai selisih kedua waktu tersebut.
12. Memverifikasi keberhasilan proses join dengan mengamati status perangkat pada Chirpstack. Apabila perangkat tidak menerima *Join Accept* dalam batas waktu yang ditentukan, percobaan dicatat dengan status “Gagal”.
13. Melakukan flush sesi join sebelumnya pada Chirpstack dan memberikan jeda waktu ±2 menit sebelum percobaan berikutnya dimulai, untuk memastikan jendela penerimaan (RX1/RX2) telah tertutup sepenuhnya dan menghindari konflik DevNonce pada percobaan selanjutnya.
14. Mengulangi langkah 7 hingga 12 sebanyak 15 kali untuk setiap konfigurasi (SF7, SF8, SF9, SF10, SF11, SF12 pada BW 125 kHz), sehingga diperoleh total 90 kali percobaan, guna memperoleh nilai rata-rata *Join Duration* dan *success rate* pada setiap konfigurasi SF.

4.1.3 Data Hasil Pengujian Join OTAA

Hasil pengujian Join OTAA terhadap 6 (enam) konfigurasi SF pada BW 125 kHz dengan masing-masing 15 kali pengulangan (total 90 percobaan) disajikan dalam bentuk rekapitulasi statistik pada Tabel 4.1. Data mentah hasil seluruh percobaan disajikan secara lengkap pada bagian Lampiran.

Tabel 4.1 Hasil Pengujian Join OTAA

SF	DR	BW (kHz)	n	Berhasil	Gagal	Success Rate (%)	Rata-Rata *Join Duration* (ms)
7	5	125	15	15	0	100,0	5127
8	4	125	15	15	0	100,0	5221
9	3	125	15	15	0	100,0	5387
10	2	125	15	15	0	100,0	5720
11	1	125	15	15	0	100,0	7173
12	0	125	15	15	0	100,0	7832

4.1.4 Analisis Data Pengujian Join OTAA

Berdasarkan Tabel 4.1, terlihat bahwa *Join Duration* meningkat secara konsisten seiring kenaikan nilai SF, dari 5.127 ms pada SF7 hingga 7.832 ms pada SF12 (kondisi BW 125 kHz).

Untuk mengetahui tingkat kestabilan dan konsistensi data pada setiap konfigurasi SF, dilakukan analisis statistik tambahan berupa nilai minimum, maksimum, standar deviasi, dan variansi dari 15 kali pengulangan pada masing-masing titik SF, sebagaimana disajikan pada Tabel 4.2.

Tabel 4.2 Statistik Variasi Data *Join Duration* per Konfigurasi SF

SF	DR	BW (kHz)	n	Standar Deviasi (ms)	Minimum (ms)	Maksimum (ms)	Variansi
7	5	125	15	0	5127	5127	0
8	4	125	15	0	5221	5221	0
9	3	125	15	0	5387	5387	0
10	2	125	15	0	5720	5720	0
11	1	125	15	0	7173	7173	0
12	0	125	15	0	7832	7832	0

Berdasarkan Tabel 4.2, nilai standar deviasi dan variansi yang tercatat sebesar 0 pada seluruh konfigurasi SF mengindikasikan bahwa seluruh 15 percobaan pada masing-masing konfigurasi menghasilkan nilai *Join Duration* yang identik, sehingga nilai minimum dan maksimum pada setiap konfigurasi juga bernilai sama dengan rata-ratanya.

Untuk membuktikan secara kuantitatif bahwa kenaikan tersebut sejalan dengan teori Time on Air (ToA) LoRa, dilakukan perhitungan ToA teoritis untuk paket *Join Request* (23 byte) dan *Join Accept* (17 byte) pada setiap konfigurasi SF/BW yang diuji.

Waktu simbol (symbol time) pada modulasi LoRa dirumuskan sebagai:

$$Ts = 2SF / BW$$

dengan Ts adalah waktu simbol (detik), SF adalah *Spreading factor*, dan BW adalah *Bandwidth* (Hz). Total waktu transmisi paket (ToA) dihitung dari jumlah simbol preamble ditambah jumlah simbol *payload*, dikalikan dengan Ts, menggunakan formula standar Semtech AN1200.13 yang turut

memperhitungkan aktivasi Low *Data rate* Optimization (DE) ketika Ts melebihi 16 ms (yaitu pada SF11 dan SF12 dengan BW 125 kHz).

Hasil perhitungan Ts menunjukkan bahwa setiap kenaikan SF sebesar 1 menyebabkan Ts menjadi tepat 2 (dua) kali lipat (1,024 ms → 2,048 ms → 4,096 ms → 8,192 ms → 16,384 ms → 32,768 ms), sesuai dengan rumus di atas dan sejalan dengan teori yang disampaikan bahwa setiap kenaikan SF akan menurunkan kecepatan data sebesar 50% dan melipatgandakan waktu transmisi. ToA *Join Request* dan *Join Accept* turut meningkat sekitar 1,8 kali lipat pada setiap kenaikan SF, dengan lompatan yang lebih besar (2,2 kali) pada transisi SF10 ke SF11 akibat aktifnya Low *Data rate* Optimization.

Untuk mengevaluasi kontribusi ToA terhadap total *Join Duration* yang terukur, dilakukan prediksi *Join Duration* dengan formula:

Join Duration (prediksi) = Receive Delay 1 + ToA *Join Request* + ToA *Join Accept*

dengan Receive Delay 1 bernilai 5.000 ms sesuai spesifikasi regional parameter LoRaWAN. Perbandingan antara prediksi dan hasil observasi disajikan pada Tabel 4.3.

Tabel 4.3 Perbandingan ToA dengan Hasil Observasi *Join Duration*

SF	BW (kHz)	DR	Ts (ms)	ToA Join Request (ms)	ToA Join Accept (ms)	Join Duration (ms)	Observasi Join Duration (ms)	Selisih (ms)
7	125	5	1.024	61.7	51.46	5113.2	5127	+13,8
8	125	4	2.048	113.15	92.67	5205.8	5221	+15,2
9	125	3	4.096	205.82	164.86	5370.7	5387	+16,3
10	125	2	8.192	370.69	329.73	5700.4	5720	+19,6
11	125	1	16.384	823.3	659.46	6482.8	7173	+690,2
12	125	0	32.768	1482.75	1318.91	7801.7	7832	+30,3

Tabel 4.3 menunjukkan bahwa pada 5 dari 6 konfigurasi (SF7, SF8, SF9, SF10, dan SF12), nilai prediksi mendekati hasil observasi dengan selisih yang relatif konsisten, yaitu sekitar 14–20 ms. Selisih tersebut diduga merupakan overhead pemrosesan pada sisi *gateway* dan *network server* yang tidak tercakup dalam model ToA murni. Kesesuaian ini membuktikan bahwa komponen dominan dari *Join Duration* bukan berasal dari ToA itu sendiri, melainkan dari jendela penerimaan (Receive Delay 1) yang bersifat tetap (fixed) sesuai spesifikasi protokol LoRaWAN, sementara ToA hanya berkontribusi sebagai komponen tambahan yang nilainya meningkat sejalan dengan kenaikan SF.

Pada konfigurasi SF11, hasil observasi (7.173 ms) memiliki selisih yang jauh lebih besar (690 ms) dibandingkan prediksi (6.482,8 ms). Penyimpangan ini kemungkinan disebabkan oleh penggunaan jendela penerimaan kedua (RX2) pada beberapa atau seluruh percobaan SF11, mengingat RX2 memiliki Receive Delay 2 yang lebih besar dibandingkan RX1, atau adanya overhead pemrosesan tambahan yang spesifik pada konfigurasi tersebut.

Untuk mengevaluasi tingkat akurasi model prediksi berbasis ToA secara kuantitatif, dihitung nilai persentase error pada setiap konfigurasi SF serta menghitung nilai *Mean Absolute Percentage Error* (MAPE) sebagai ukuran rata-rata akurasi model prediksi. Berikut perhitungan persentase error pada setiap konfigurasi yang dapat dilihat pada tabel 4.4.

Tabel 4.4 Analisis Error Prediksi *Join Duration*

SF	Prediksi (ms)	Observasi (ms)	Selisih (ms)	Error (%)
7	5113.2	5127	13.8	0.27
8	5205.8	5221	15.2	0.29
9	5370.7	5387	16.3	0.30
10	5700.4	5720	19.6	0.34
11	6482.8	7173	690.2	10.65
12	7801.7	7832	30.3	0.39

Berdasarkan Tabel 4.4, persentase error pada konfigurasi SF7 hingga SF10 dan SF12 berada dalam rentang 0,27% hingga 0,39%, yang menunjukkan kesesuaian yang sangat tinggi antara model prediksi berbasis ToA dan hasil observasi aktual. Sementara itu, konfigurasi SF11 mencatatkan persentase error sebesar 10,65% jauh di atas pola error normal pada konfigurasi lainnya.

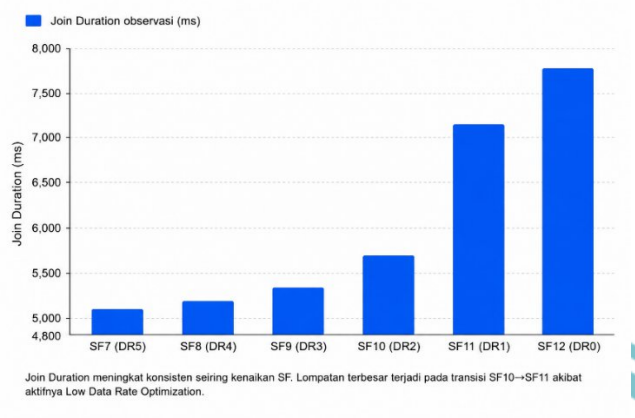
Penyimpangan ini konsisten dengan temuan pada Tabel 4.2 sebelumnya, di mana selisih antara prediksi dan observasi pada SF11 mencapai 690,2 ms dan diduga disebabkan oleh penggunaan jendela penerimaan RX2 yang memiliki Receive Delay 2 sebesar 6.000 ms, bukan RX1 sebesar 5.000 ms yang menjadi asumsi dasar model prediksi.

Nilai MAPE atau *Mean Absolute Percentage Error* keseluruhan dihitung sebagai berikut:

$$MAPE = \frac{0.27 + 0.29 + 0.30 + 0.34 + 10.65 + 0.39}{6} = 2.04\%$$

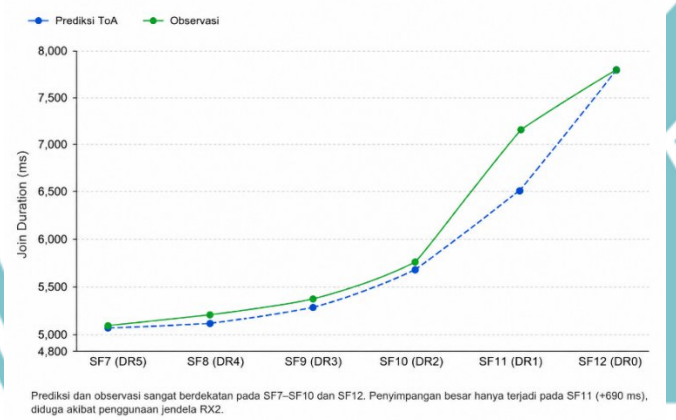
Nilai MAPE sebesar 2,04% secara keseluruhan mencerminkan pengaruh dominan anomali SF11 terhadap akurasi rata-rata model prediksi. Hal ini menegaskan bahwa anomali pada SF11 bukan sekadar penyimpangan statistik biasa, melainkan indikasi adanya perbedaan mekanisme penerimaan yang mendasar dibandingkan konfigurasi SF lainnya.

Setelah mendapat jawaban dari pola kenaikan *Join Duration*, prediksi ToA, hingga persentase error dan MAPE pada setiap SF, dalam memperjelas pola kenaikan *Join Duration* dan kesesuaiannya dengan prediksi teoritis, hasil pengujian divisualisasikan dalam bentuk grafik pada Gambar 4.1 hingga Gambar 4.3.



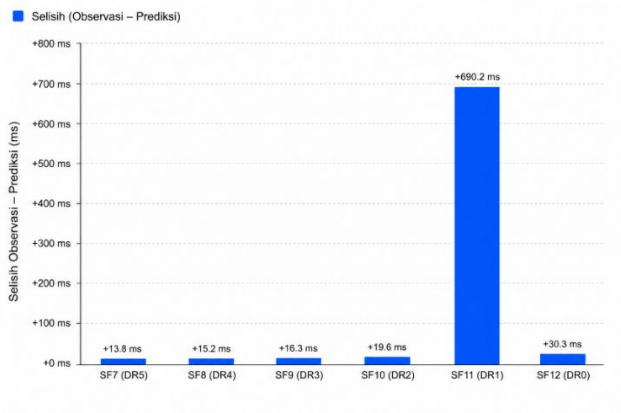
Gambar 4.1 Grafik Perbandingan Nilai SF dengan Waktu Join

Gambar 4.1 menampilkan hubungan antara nilai *Spreading factor* dengan rata-rata *Join Duration* yang terukur. Grafik tersebut memperlihatkan bahwa kenaikan *Join Duration* tidak bersifat linear, melainkan semakin besar seiring meningkatnya SF, dengan lompatan yang paling signifikan terjadi pada transisi SF10 ke SF11 akibat aktifnya mekanisme Low *Data rate* Optimization.



Gambar 4.2 Grafik Perbandingan Prediksi dan Observasi

Gambar 4.2 menyajikan perbandingan antara nilai *Join Duration* hasil prediksi berbasis ToA dan nilai *Join Duration* hasil observasi pada setiap konfigurasi SF. Kedua garis tampak berhimpitan pada konfigurasi SF7 hingga SF10 serta SF12, yang mengonfirmasi kesesuaian antara model teoritis dan hasil pengukuran aktual. Penyimpangan yang terlihat jelas hanya terjadi pada SF11, di mana garis observasi berada jauh di atas garis prediksi.



Gambar 4.3 Grafik Nilai Observasi

Gambar 4.3 menampilkan selisih antara nilai observasi dan prediksi (error) pada setiap konfigurasi SF. Pada SF7 hingga SF10 dan SF12, selisih berkisar antara 14 hingga 20 ms yang merepresentasikan overhead pemrosesan pada sisi *gateway* dan *network server*. Sementara itu, selisih pada SF11 mencapai 690 ms dan tampak jauh menonjol dibandingkan konfigurasi lainnya, menegaskan bahwa penyimpangan pada SF11 bukan merupakan bagian dari pola overhead normal, melainkan disebabkan oleh faktor lain yang diduga berkaitan dengan penggunaan jendela penerimaan RX2.

Dengan demikian, hasil pengujian ini dapat dinyatakan memiliki tingkat konsistensi dan *repeatability* yang sangat tinggi, dan data yang diperoleh selama seluruh sesi pengujian, perangkat *end device Heltec WiFi LoRa 32 V3* tidak berpindah posisi dari titik pengujian di Gedung Laboratorium Telekomunikasi PNJ (Lingkungan Gedung G Lab Telkom PNJ), sehingga kondisi lingkungan dan jarak antara *end device* dengan *gateway SenseCAP M2* dapat dianggap konstan pada setiap percobaan.

4.1.5. Implementasi Sketch *Arduino IDE* pada Pengujian Join OTAA

Implementasi sketch program pada *Arduino IDE* yang digunakan dalam pengujian Join OTAA dengan variasi konfigurasi *Spreading factor* (SF) dan *Bandwidth* (BW) menggunakan pustaka *RadioLib* yang mendukung modul radio SX1262 pada *Heltec WiFi LoRa 32 V3*, dengan objek LoRaWANNode yang diinisiasi mengacu pada regional parameter AS923.

Pustaka *Preferences* digunakan untuk menyimpan penghitung repetisi (currentRep) pada Non-Volatile Storage (NVS) ESP32-S3 sesuai kunci unik per DR, sehingga perangkat dapat melanjutkan urutan repetisi secara otomatis setiap kali terjadi restart hingga mencapai batas maksimum 15 kali percobaan per titik DR.

Variabel TEST_DR menjadi parameter kunci yang menentukan konfigurasi SF dan BW pada setiap sesi pengunggahan sketch, dengan nilai 0 hingga 6 yang masing-masing merepresentasikan DR0 (SF12) hingga DR6 (SF7/BW250) sesuai regional parameter AS923-2, di mana DR6 difungsikan secara khusus sebagai satu-satunya titik perbandingan BW murni terhadap DR5 karena keduanya menggunakan SF yang sama namun BW berbeda.

const LoRaWANBand_t Region = AS923;, Region diset ke AS923 sebagai regional parameter yang menentukan rentang frekuensi, daftar kanal, serta aturan *duty cycle* dan dwell time yang berlaku — ini sesuai regulasi Indonesia (Permenkominfo No. 2 Tahun 2023).

Proses autentikasi OTAA diinisialisasi melalui fungsi node.beginOTAA() dengan parameter identitas devEUI, joinEUI, dan appKey yang telah disesuaikan dengan data perangkat terdaftar pada Chirpstack, kemudian diikuti dengan penonaktifan *Adaptive data rate* melalui node.setADR(false) agar nilai SF yang ditetapkan tidak berubah secara otomatis selama pengujian berlangsung, serta penonaktifan batas dwell time melalui node.setDwellTime(false, 0) agar paket pada konfigurasi SF tinggi seperti DR0 tidak ditolak radio akibat melebihi batas dwell time 400 milidetik yang diterapkan pada regulasi AS923.

Kondisi *cold start* pada setiap repetisi dicapai melalui mekanisme restart fisik perangkat, di mana seluruh logika pengujian dieksekusi penuh di dalam fungsi setup() sementara fungsi loop() dikosongkan, sehingga setiap repetisi merupakan satu siklus boot utuh tanpa sesi aktif sebelumnya. Status keberhasilan ditentukan berdasarkan nilai return node.activateOTAA(), di mana nilai *RADIOLIB*_LORAWAN_NEW_ SESSION menandakan join berhasil, sedangkan nilai return lainnya dicatat sebagai kode kegagalan.

4.2. Pengujian *Gateway bridge*

4.2.1. Deskripsi Pengujian

Pengujian integrasi *Gateway bridge* merupakan skenario pengujian yang dilakukan untuk memastikan kemampuan *Gateway bridge* Chirpstack dalam menerima, mengkonversi, dan meneruskan paket komunikasi antara *gateway SenseCAP M2* dengan *Network server*. Pengujian ini bertujuan untuk memastikan bahwa *Gateway bridge* berfungsi dengan benar sebagai jembatan protokol yang mengubah format paket *packet forwarder* dari *gateway* menjadi pesan *MQTT* yang dapat diproses oleh komponen Chirpstack lainnya, serta sebaliknya meneruskan perintah *downlink* dari *Network server* kembali ke *gateway* dalam format yang sesuai.

Pengamatan dilakukan terhadap empat aspek utama fungsi *Gateway bridge*, yaitu deteksi koneksi *gateway* ke *Gateway bridge*, penerimaan statistik *gateway* (stats), penerusan paket *uplink* dari *gateway* ke *broker MQTT*, serta pengiriman paket *downlink* dari *Network server* menuju *gateway*. Parameter utama yang diamati pada pengujian ini yaitu status koneksi *gateway* pada *Gateway bridge* serta kemunculan dan kelengkapan isi pesan JSON pada topic *MQTT* yang relevan, tanpa mengukur parameter waktu atau *throughput* secara numerik. Pengamatan dilakukan melalui command *Mosquitto*_sub pada topic *gateway/{gateway_id}/event/stats* dan *gateway/{gateway_id}/event/up*, serta pembacaan log layanan *Gateway bridge* yang berjalan di dalam lingkungan kontainer *Docker*.

4.2.2. Prosedur Pengujian

Pengujian *Gateway bridge* dilakukan untuk mengetahui kemampuan komponen tersebut dalam mendeteksi koneksi *gateway*, menerima statistik *gateway*, meneruskan paket *uplink*, serta mengirimkan paket *downlink* melalui protokol *MQTT*. Tahapan pengujian yang dilakukan adalah sebagai berikut:

1. Memastikan seluruh kontainer layanan pendukung pada *server* LoRaWAN, yaitu *Mosquitto* (*MQTT Broker*), *PostgreSQL*, *Redis*, Chirpstack, dan *Gateway bridge*, berjalan dengan normal melalui command *docker* ps, dengan memastikan seluruh kontainer menunjukkan status Up.

2. Memeriksa log layanan *Gateway bridge* melalui command *docker* logs Chirpstack-*docker*-Chirpstack-*gateway*-bridge-1 untuk memastikan komponen tersebut telah berhasil terhubung ke *broker Mosquitto*, ditandai dengan kemunculan baris log koneksi berhasil tanpa adanya pesan error.
3. Memastikan *gateway SenseCAP M2* dalam kondisi menyala dan telah dikonfigurasi untuk meneruskan paket menggunakan protokol UDP *packet forwarder* menuju alamat IP dan port 1700 milik *Gateway bridge* pada *server*.
4. Membuka sesi terminal SSH pada VM Ubuntu untuk menjalankan command *Mosquitto*_sub -h localhost -t "*gateway*/+/event/stats" -v guna memantau topic statistik *gateway* yang dipublikasikan secara periodik oleh *Gateway bridge*.
5. Mengamati terminal untuk memverifikasi kemunculan pesan JSON pada topic event/stats, yang menandakan *Gateway bridge* berhasil mendeteksi koneksi *gateway SenseCAP M2* dan menerima data statistik dari *gateway* tersebut secara berkala.
6. Membuka sesi terminal terpisah untuk menjalankan command *Mosquitto*_sub -h localhost -t "*gateway*/+/event/up" -v guna memantau topic yang memuat paket *uplink* yang diteruskan oleh *Gateway bridge*.
7. Menyalakan perangkat *end device Heltec WiFi LoRa 32 V3* untuk mengirimkan paket *Join Request* maupun data *uplink* aplikasi melalui sinyal radio LoRa ke *gateway SenseCAP M2*.
8. Mengamati terminal pada langkah 6 untuk memverifikasi kemunculan pesan JSON pada topic event/up, yang menandakan *Gateway bridge* berhasil menerima paket radio dari *gateway* dalam format *packet forwarder* dan mengkonversinya menjadi pesan *MQTT* yang dipublikasikan ke *broker*.
9. Memeriksa kelengkapan isi pesan JSON yang muncul pada kedua topic, meliputi identitas *gateway* (*gateway*Id), timestamp (time), serta untuk topic event/up ditambah dengan informasi frekuensi, *Spreading factor*, RSSI, dan SNR paket yang diterima.
10. Memverifikasi kemampuan pengiriman paket *downlink* dengan mengamati log *Gateway bridge* pada saat *Network server* mengirimkan respons *Join*

Accept kepada *end device*, untuk memastikan *Gateway bridge* berhasil menerima perintah *downlink* dari topic *gateway*/{*gateway*_id}/ command/down dan meneruskannya ke *gateway* dalam format UDP *packet forwarder*.

11. Mencatat status keberhasilan pengujian berdasarkan indikator berikut: status seluruh kontainer layanan (Up), koneksi *gateway* ke *Gateway bridge* (terdeteksi melalui topic stats), penerusan paket *uplink* (pesan muncul pada topic event/up), dan pengiriman paket *downlink* (log konfirmasi *Join Accept* diteruskan ke *gateway*).

4.2.3. Data Hasil Pengujian *Gateway bridge*

Hasil pengujian integrasi *Gateway bridge* dalam mendeteksi koneksi *gateway*, menerima statistik *gateway*, meneruskan paket *uplink*, serta mengirimkan paket *downlink* melalui protokol *MQTT* disajikan dalam bentuk rekapitulasi status pada Tabel 4.5. Bukti catatan log pengujian disajikan pada bagian Lampiran.

Tabel 4.5 Hasil Data Pengujian *Gateway bridge*

No.	Komponen yang Diuji	Indikator	Status	Keterangan
1	Status UDP listener *Gateway bridge*	Log *Chirpstack-docker-Chirpstack-gateway-bridge-1*	Berhasil	Log menampilkan starting *gateway* udp listener addr="0.0.0.0:1700" menandakan *Gateway bridge* siap menerima paket dari *gateway* pada port 1700
2	Koneksi *Gateway bridge* ke *MQTT Broker*	Log *integration/MQTT: connected to MQTT broker*	Berhasil	Log menampilkan *baris connected to MQTT broker* tanpa didahului pesan connection refused, menunjukkan koneksi berhasil
3	Penerimaan statistik *gateway* (event/stats)	*gateway*/+/event/stats	Berhasil	Pesan JSON statistik *gateway* muncul secara periodik pada terminal *Mosquitto*_sub, memuat *gateway*Id dan jumlah paket

No.	Komponen yang Diuji	Indikator	Status	Keterangan
4	Penerusan paket *uplink* (event/up)	*gateway*/+/event/up	Berhasil	Pesan JSON muncul setiap kali *end device* mengirimkan sinyal, memuat informasi frekuensi, SF, RSSI, dan SNR paket yang diterima *gateway*
5	Pengiriman paket *downlink* (*command/down*)	Log saat *Join Accept* dikirim	Berhasil	Log *Gateway bridge* menampilkan penerimaan perintah *downlink* dari *Network server* dan meneruskannya ke *gateway* dalam format UDP *packet forwarder*

4.2.4. Analisis Data Pengujian *Gateway bridge*

Berdasarkan hasil pengujian pada Tabel 4.3, seluruh komponen yang diuji pada *Gateway bridge* menunjukkan status berhasil tanpa ditemukan adanya kegagalan pada setiap indikator yang diamati. Keberhasilan *Gateway bridge* dalam membuka UDP listener pada port 1700 menjadi prasyarat dasar yang membuktikan komponen tersebut siap menerima paket *packet forwarder* dari *gateway SenseCAP M2* sebelum proses komunikasi lebih lanjut dapat berlangsung. Selanjutnya, keberhasilan koneksi *Gateway bridge* ke *broker Mosquitto* yang ditandai dengan kemunculan log connected to *MQTT broker* tanpa didahului pesan connection refused menunjukkan bahwa proses handshake antara *Gateway bridge* dan *broker MQTT* berjalan lancar pada percobaan pertama, sehingga jalur distribusi pesan dapat segera terbentuk.

Keberhasilan penerimaan statistik *gateway* pada topic event/stats membuktikan bahwa *Gateway bridge* mampu mendeteksi keberadaan dan status koneksi *gateway SenseCAP M2* secara periodik, yang menjadi indikator bahwa *gateway* fisik telah terhubung dengan baik ke infrastruktur backend Chirpstack. Lebih lanjut, keberhasilan penerusan paket *uplink* pada topic event/up menunjukkan bahwa *Gateway bridge* berfungsi sebagaimana mestinya dalam mengkonversi paket radio LoRa yang diterima *gateway* dari format *packet forwarder* menjadi pesan *MQTT*, lengkap dengan informasi frekuensi, *Spreading factor*, RSSI, dan SNR yang dibutuhkan untuk proses autentikasi maupun analisis kualitas sinyal pada tahap selanjutnya.

4.3. Pengujian Integrasi *MQTT* pada *Application server*

4.3.1. Deskripsi Pengujian

Pengujian integrasi *MQTT* pada *Application server* merupakan salah satu skenario pengujian yang dilakukan untuk memverifikasi keberhasilan distribusi data pada platform Chirpstack. Pengujian ini berfokus pada mekanisme publish-subscribe yang digunakan Chirpstack untuk meneruskan dua jenis event utama, yaitu event keberhasilan proses join OTAA dan event penerimaan data *uplink* aplikasi dari *end device Heltec WiFi LoRa 32 V3*.

Kondisi pengujian dilakukan dengan memantau dua topic *MQTT* secara bersamaan melalui command *Mosquitto_sub*, yaitu topic *application/{id}/device/{dev_eui}/event/join* dan topic *application/{id}/device/{dev_eui}/event/up*, pada *broker Mosquitto* yang berjalan di dalam lingkungan kontainer *Docker*. Parameter utama yang diamati pada pengujian ini yaitu status keberhasilan koneksi *application server* terhadap *MQTT broker* serta kemunculan pesan JSON pada kedua topic tersebut setiap kali proses join OTAA dan pengiriman data *uplink* berhasil dilakukan. Pengamatan dilakukan melalui pembacaan log pada terminal subscriber *MQTT* serta log layanan Chirpstack.

4.3.2. Prosedur Pengujian

Pengujian integrasi *MQTT* pada *application server* dilakukan untuk mengetahui kemampuan Chirpstack dalam mendistribusikan data event join dan

event *uplink* melalui protokol *MQTT* secara konsisten. Tahapan pengujian yang dilakukan adalah sebagai berikut:

1. Memastikan seluruh kontainer layanan pendukung pada *server* LoRaWAN, yaitu *Mosquitto* (*MQTT Broker*), *PostgreSQL*, *Redis*, dan Chirpstack, berjalan dengan normal melalui command *docker* ps, dengan memastikan seluruh kontainer menunjukkan status Up.
2. Memeriksa log layanan Chirpstack melalui command *docker logs Chirpstack-docker-Chirpstack-1 | grep -i MQTT* untuk memastikan *Application server* telah berhasil terhubung ke *broker Mosquitto*, ditandai dengan kemunculan baris log yang mengindikasikan koneksi berhasil tanpa adanya pesan error atau connection refused.
3. Membuka dua sesi terminal SSH terpisah pada VM Ubuntu yang sama untuk menjalankan command *Mosquitto_sub* secara bersamaan, yaitu sesi pertama untuk memantau topic *application/+/device/+/event/join* dan sesi kedua untuk memantau topic *application/+/device/+/event/up*.
4. Menyalakan perangkat *end device Heltec WiFi LoRa 32 V3* dalam kondisi *cold start* untuk memulai proses join OTAA ke *Network server* Chirpstack.
5. Mengamati terminal pada sesi pertama untuk memverifikasi kemunculan pesan JSON pada topic event/join, yang menandakan *Application server* berhasil menerima notifikasi keberhasilan autentikasi OTAA dari *Network server* dan meneruskannya sebagai pesan *MQTT*.
6. Mengamati terminal pada sesi kedua setelah perangkat berhasil join, untuk memverifikasi kemunculan pesan JSON pada topic event/up, yang menandakan *Application server* berhasil menerima dan mempublikasikan data *uplink* aplikasi yang dikirimkan oleh *end device*.
7. Mencatat status keberhasilan publikasi pesan pada kedua topic berdasarkan indikator berikut: status layanan *MQTT broker* (active (running)), koneksi *application server* ke *broker* (tidak ditemukan error pada log), kemunculan pesan pada topic event/join, serta kemunculan pesan pada topic event/up.
8. Mengulangi langkah 4 hingga 7 pada beberapa sesi pengujian join OTAA yang telah dilakukan sebelumnya untuk memastikan integrasi *MQTT*.

4.3.3. Data Hasil Pengujian Integrasi *MQTT*

Hasil pengujian integrasi *MQTT* pada *Application server* terhadap publikasi event join dan event *uplink* melalui *broker Mosquitto* disajikan dalam bentuk rekapitulasi status pada Tabel 4.4. Data hasil observasi pada setiap komponen yang diuji disajikan secara lengkap pada bagian Lampiran.

Tabel 4.6 Hasil Pengujian Integrasi *MQTT*

No.	Komponen yang Diuji	Indikator	Status	Keterangan
1	Status Layanan *MQTT Broker* (*Mosquitto*)	*docker-compose ps*	Berhasil	Semua layanan menunjukkan status up
2	Koneksi *Application server* ke *MQTT Broker*	Log layanan Chirpstack	Berhasil	Tidak ditemukan pesan error atau connection refused
3	Publikasi pesan pada topic event/join	application/+/device/+/event/join	Berhasil	Pesan JSON muncul setiap kali proses *join* OTAA berhasil
4	Publikasi pesan pada *topic* event/up	application/+/device/+/event/up	Berhasil	Pesan JSON muncul setiap kali *end device* mengirimkan data *uplink*

No.	Komponen yang Diuji	Indikator	Status	Keterangan
5	Konsistensi pada sesi pengujian SF7–SF12	Observasi log selama seluruh sesi pengujian	Berhasil	Tidak ditemukan kegagalan publikasi pesan pada seluruh percobaan yang diamati

4.3.4. Analisis Data Pengujian Integrasi *MQTT*

Berdasarkan hasil pengujian pada Tabel 4.3, seluruh komponen yang diuji menunjukkan status berhasil tanpa ditemukan adanya kegagalan pada setiap indikator yang diamati. Status layanan *MQTT Broker* yang menunjukkan kondisi Up pada seluruh kontainer *Docker* menjadi prasyarat dasar yang membuktikan kesiapan *Mosquitto* sebelum proses komunikasi antar komponen Chirpstack berjalan, sementara keberhasilan koneksi *Application server* ke *broker* tanpa adanya pesan error atau connection refused pada log layanan Chirpstack mengindikasikan bahwa proses pembangunan komunikasi antara kedua komponen tersebut berjalan lancar sejak sistem pertama kali dijalankan.

Keberhasilan publikasi pesan pada topic *event/join* dan *event/up* menunjukkan bahwa *Application server* mampu menerima notifikasi internal dari *Network server*, baik berupa konfirmasi autentikasi OTAA maupun data *uplink* aplikasi, dan meneruskannya sebagai pesan *MQTT* secara konsisten setiap kali kedua event tersebut terjadi. Hal ini membuktikan bahwa jalur komunikasi publish-subscribe yang terbentuk antara *Network server*, *Application server*, dan *broker Mosquitto* telah terintegrasi dengan baik dalam mendistribusikan data secara real-time, mulai dari proses autentikasi perangkat hingga pengiriman data sensor dari *end device*.