Unit I — Introduction to IoT: Smart Healthcare

Problem overview

Design a scalable, secure IoT ecosystem for remote patient monitoring, asset tracking, and clinical decision support.

Key challenges

- **Data confidentiality & privacy:** HIPAA-like requirements, encryption at-rest and in-transit, anonymization for analytics.
- **Latency and reliability:** Critical alerts (fall detection, arrhythmia) require low-latency paths and fault-tolerant delivery.
- **Heterogeneity:** Multiple device types (wearables, bedside monitors, imaging systems) and protocols (BLE, Wi-Fi, LoRaWAN).
- **Scalability:** Tens of thousands of endpoints and high-frequency telemetry (e.g., ECG streams).
- **Interoperability & standards:** FHIR for clinical data, HL7 messaging for legacy systems.
- **Regulatory & auditability:** Secure logging, device attestations, firmware provenance.

Architectural requirements

1. **Device layer:** Constrained devices and gateways. Support secure boot, TPM/secure elements.
2. **Edge/fog layer:** Local preprocessing, real-time analytics, anomaly detection, and transient data caching to reduce cloud cost and latency.
3. **Communication layer:** Transport security (TLS/DTLS), MQTT for telemetry, CoAP for constrained devices, HTTP/REST for management.
4. **Cloud platform layer:** Device management, identity & access management (IAM), long-term storage, analytics, and ML model hosting.
5. **Application layer:** Clinical dashboards, alerting, EHR integration (FHIR), role-based access for clinicians.

Enabling technologies & APIs

- **MQTT/AMQP** for pub/sub telemetry; MQTT TLS + client certs for secure mutual auth.
- **CoAP/DTLS** for constrained devices.
- **REST** APIs for backend services and EHR integration (use FHIR RESTful endpoints).
- **OAuth2 / OpenID Connect** for user & app authentication.
- **Edge toolkits:** TensorFlow Lite for on-device inference; Apache NiFi/StreamSets for data flow.

Domain-specific concepts

- **Patient identity mapping** (link device IDs to patient IDs securely, tokenization).

- **Clinical-grade SLAs** (guarantees for alert delivery times).
- **Quality of Service (QoS)** levels in MQTT to prioritize critical messages.

Unit II — Infrastructure & Service Discovery: LPWAN in Smart Waste Management

Scenario

City-wide network to monitor fill-level sensors on waste bins, route optimization, and scheduled collection.

Why LPWAN?

- **Long range & low power:** Devices operate years on battery and cover wide urban/rural areas.
- **Cost-effective** for low-bandwidth telemetry (fill-level, battery status, tamper alerts).

Comparison: LoRaWAN vs 6LoWPAN

- **LoRaWAN**: Star-of-stars topology with gateways; adaptive data rate; good for sparse uplink telemetry; network servers handle deduplication, ADR; not IP-native.
- **6LoWPAN**: IPv6 over low-power radios (e.g., IEEE 802.15.4); supports mesh; IP-native allowing REST/CoAP directly to endpoints; better where mesh is needed.

Layered protocol architecture (example with LoRaWAN)

- **Application layer:** JSON/CBOR payloads, application server processes telemetry.
- **Transport/session:** LoRaWAN MAC and network server (handles join, ADR, frame counters).
- **Network:** Gateways forward packets over TCP/TLS to network server.
- **Physical/MAC:** LoRa PHY parameters (SF, bandwidth).

Device & service discovery

- **Device provisioning:** OTAA (Over The Air Activation) or ABP with secure keys managed in network server.
- **Service discovery:** For 6LoWPAN, use CoAP resource discovery (/.well-known/core) and mDNS in local domains; for LoRaWAN, discovery occurs via device registry mapping DevEUI→Application.

Real-time requirements & strategies

- Use **edge gateways** to perform local aggregation and burst-tolerant uplinks.

- **Duty cycle & downlink limits** of LoRaWAN mean avoid frequent downlinks — schedule bulk configuration windows.
- **QoS for critical events** (fire, overflow) routed via redundant paths (cellular fallback on gateway).

Diagram — network layout

Unit III — Python & Raspberry Pi: Home Automation Prototype

System design

Raspberry Pi acts as central controller (edge hub). Sensors (temperature, motion, light) and actuators (relay-controlled lights, motorized curtains) are interfaced via GPIO, I²C, SPI, and UART.

Hardware interfacing

- **GPIO:** Digital inputs (PIR motion), outputs to drive relays (via transistor/optocoupler).
- **I²C:** Environmental sensors (BME280) and displays (OLED) share I²C bus (pull-ups needed).
- **SPI:** High-speed sensors (e.g., ADC like MCP3008) or external radio modules.
- **UART/Serial:** Communication with microcontrollers (Arduino) or Zigbee coordinators.

Sample Python components

- Use RPi.GPIO or gpiozero for GPIO control.
- smbus2 for I²C, spidev for SPI, pyserial for UART.
- Lightweight local broker: mosquitto for MQTT on Pi; publish sensor telemetry to ThingSpeak or cloud via HTTPS.

ThingSpeak integration

- Send periodic HTTP POST or MQTT messages to ThingSpeak channels (API keys used as write keys).
- Use field formatting and timestamps. For alerts, push to IFTTT or SMS gateway.

Data flow & control loop

- Local automation rules run on Pi (e.g., motion + night -> turn on light) to avoid cloud dependence for critical controls.
- Cloud used for long-term logging, visualization, remote overrides.

Diagram — prototype

Unit IV — Cloud for IoT: Smart Traffic with Hybrid Cloud & Fog

Problem

Real-time traffic signal optimization, incident detection, and historical analytics for planning.

Architectural components

1. **Edge/Fog nodes:** Roadside units (RSUs) and signal controllers run low-latency inference (vehicle count, emergency vehicle preemption).
2. **Message bus:** Kafka clusters for durable ingestion of high-throughput telemetry (video metadata, sensor counts).
3. **Stream processing:** Apache Spark Structured Streaming for windowed aggregations, feature extraction, and feeding ML models.
4. **Long-term storage:** Time-series DB (InfluxDB / Prometheus for metrics), object storage for video (S3), and data warehouse for planning (Snowflake/Redshift).
5. **Control plane:** Microservices handle route suggestions, signal timing pushes, and public APIs.

Hybrid cloud + fog rationale

- **Fog** handles low-latency control loops (sub-100ms) and de-identification of video at the edge to preserve privacy.
- **Cloud** handles heavy ML training, global coordination, long term analytics.

IoT Data Analytics Pipeline (Kafka + Spark)

1. Devices → Edge gateway → Kafka producers → Kafka topics (ingestion)
2. Spark Structured Streaming consumes Kafka topics, performs sliding-window counts and anomaly detection
3. Results → Control topic (Kafka) → Edge consumer to adjust signals; also stored in time-series DB and fed to dashboards

Diagram — pipeline & placement

Unit V — AWS IoT: Industrial Deployment

Use case

Industrial plant with thousands of sensors for predictive maintenance, process control, and telemetry visualizations.

AWS IoT features used

- **AWS IoT Core:** secure device connectivity (MQTT over TLS) and rules engine for routing messages to AWS services.
- **Device Management:** fleet provisioning, jobs, OTA firmware updates.
- **Device Shadow:** virtual device state to allow apps to read/set desired states even when device offline.
- **Resource tagging:** organize resources (by plant, line, criticality) for cost allocation and access control.

Data storage & retrieval

- Route telemetry via IoT Rules to **Kinesis** or **Kafka** (MSK) and then to **S3** for raw data; use **Timestream** or **InfluxDB** for time-series analytics.
- Use **Lambda** for lightweight transforms and **AWS Glue** for ETL into data warehouse (Redshift).

Scalability & best practices

- **Thing registry** with hierarchical naming and provisioning templates.
- Use **X.509 certs** and AWS IoT jobs for secure OTA updates.
- **Sharding topics** and using partition keys for downstream consumers to scale ingestion.
- Implement backpressure handling (IoT Core → Kinesis limits) and auto-scaling consumer groups.

Example workflow

1. Device publishes telemetry (MQTT TLS) to plant/line1/machineA/telemetry.
2. IoT Rule matches and sends to Kinesis → Lambda for enrichment → Timestream & S3.
3. Dashboard queries Timestream for near-real-time KPIs and S3 for batch ML training.
4. To update setpoint, operator writes desired state to Device Shadow; device syncs on next connect.

Diagram — AWS IoT deployment

Common design recommendations (cross-unit)

- **Security-first:** hardware root-of-trust, mutual TLS, principle of least privilege, encrypted storage.
- **Edge processing:** move time-sensitive logic to edge to reduce latency and cloud costs.
- **Standards & interoperability:** favor IP-native stacks where possible, and adopt healthcare/industrial standards as appropriate.
- **Observability:** centralized logging, distributed tracing, device health telemetry, and SLA monitoring.
- **Testing & simulation:** digital twins and test harnesses to validate scale, failover, and update workflows before production deployment.