DESIGNING THE IOT
UNIT - V

SYLLABUS:

IoT Tools, Devices, and Security
Introduction to configuration tools: Chef, Puppet, Ansible. Device management with NETCONF and YANG, Basic building blocks of IoT physical devices, **Popular Devices:** pcDuino, BeagleBone Black, Raspberry Pi, Arduino, Cubieboard, **Domain-Specific IoTs:** Healthcare, Smart Cities, Industrial IoT, **IoT Security:** Threats, authentication, data integrity, secure communication.

IoT Tools, Devices, and Security

Introduction to configuration tools:

Configuration Tools: Chef, Puppet, and Ansible

Overview of Configuration Management

Configuration Management (CM) is the process of **automating the setup, configuration, and maintenance** of systems — servers, network devices, or IoT gateways — to ensure consistency and reduce manual work.

Without CM, admins must manually install software, set parameters, or apply security updates — a time-consuming and error-prone process.
CM tools like **Chef, Puppet, and Ansible** automate these tasks, ensuring systems are **configured identically** across environments.

Chef

Chef is an open-source configuration management and automation tool developed by *Opscode (now Progress Software)*.
It uses **Ruby** as its scripting language and follows an **imperative approach**, meaning you define *how* the configuration should be applied.

Architecture

Chef has a **client-server model** with the following components:

1. **Chef Server:**
   Central hub that stores configuration data (cookbooks, recipes, roles). Clients pull configuration from the server.
2. **Chef Workstation:**
   Where administrators write configuration code using Ruby, test it, and upload it to the Chef Server.
3. **Chef Client (Node):**
   Installed on each managed system. The client periodically checks with the server for updates and applies configurations locally.
4. **Cookbooks and Recipes:**

- **Recipe:** A Ruby script that defines a set of resources (e.g., install Apache, start service).
- **Cookbook:** A collection of related recipes.

Workflow

1. Write recipes on the workstation.
2. Upload them to the Chef server.
3. Nodes (clients) periodically check the server and apply updates.

Example Recipe

```
package 'httpd' do
  action :install
end

service 'httpd' do
action [:enable,
:start] end
```

➞ Installs and starts the Apache HTTP server automatically.

Advantages

- Scalable and flexible.
- Integrates with cloud platforms (AWS, Azure, GCP).
- Supports complex configurations.

Disadvantages

- Steep learning curve (uses Ruby).
- Requires Chef Server setup.

Puppet

Puppet is one of the earliest and most popular configuration management tools.
It uses a **declarative language** and focuses on describing *what* the system's final state should be, rather than *how* to reach it.

Architecture

Puppet follows a **master-agent model:**

Puppet Master (Server):

Holds configuration definitions written in Puppet DSL (Domain-Specific Language).

Puppet Agent (Client):

Runs on target nodes; requests configuration from the master.

Catalog:

A compiled version of configuration data sent from the master to the agent.

Facter:

A tool that gathers system information (e.g., OS, IP address) used in Puppet configurations.

Workflow

1. Admin writes configuration in Puppet DSL and stores it on the master.
2. The agent sends system facts to the master.
3. The master compiles a configuration "catalog."
4. The agent applies the configuration locally.

Example Puppet Code

```
package { 'nginx': ensure =>
  installed,
}


service { 'nginx': ensure =>
  running, enable => true,
}
```

➞ Ensures that Nginx is installed and running.

Advantages

- Declarative (less coding effort).
- Mature and widely used.
- Supports large-scale enterprise systems.

Disadvantages

- Complex initial setup.
- Requires a Puppet master server.

Ansible

Ansible is an open-source **agentless** configuration management and automation tool. It uses **SSH** for communication — no software needs to be installed on target machines.

It is written in **Python**, and configurations are defined using **YAML** in files called **Playbooks.**

Architecture

Ansible has a **simpler architecture**:

- **Control Node:** Where Ansible is installed and run.
- **Managed Nodes:** Target devices that Ansible configures.
- **Inventory File:** Lists managed nodes (e.g., IP addresses).
- **Playbooks:** YAML files describing the automation tasks.

Workflow

1. The administrator writes a playbook.
2. Ansible connects to target systems via SSH.
3. Executes the playbook instructions.
4. Reports the results.

Advantages

- No agent installation required.
- Easy to learn (YAML syntax).
- Highly scalable and flexible.
- Integrates well with cloud and network devices.

Disadvantages

- Less efficient with very large-scale setups.
- Some complex workflows may be slower.

Comparison Table

Feature	**Chef**	**Puppet**	**Ansible**
Language	Ruby	Puppet DSL	YAML (Playbooks)
Architecture	Client-Server	Master-Agent	Agentless
Communication	Pull-based	Pull-based	Push-based
Learning Curve	High	Medium	Low
Ease of Setup	Moderate	Complex	Simple
Ideal Use	Cloud & large systems	Enterprises	Multi-platform automation

Device Management with NETCONF and YANG

Network device management traditionally relied on CLI commands or SNMP, which were **manual and inconsistent**.
Modern programmable networks use **NETCONF** and **YANG** for standardized, automated configuration.

NETCONF (Network Configuration Protocol)

Definition:

NETCONF (RFC 6241) is an IETF standard protocol designed to manage and configure network devices programmatically.

Key Features

- Uses **XML** for data encoding.
- Uses **SSH** as a transport layer for secure communication.
- Supports configuration, retrieval, and validation.
- Transaction-based (supports *commit* and *rollback*).

NETCONF Operations

<get> – Retrieves data from a device.

<edit-config> – Modifies configuration data.

<copy-config> – Copies one configuration to another.

<delete-config> – Deletes configuration data.

<lock> / <unlock> – Prevents concurrent edits.

<commit> – Applies confirmed changes.

Architecture

- **Manager (Client):** Sends NETCONF commands (e.g., a network controller).
- **Agent (Server):** Network device running NETCONF server.
- Communication occurs via XML RPC over SSH.

Example NETCONF XML Request

```
<rpc message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>

      <running/>

    </target>
    <config>
      <interface>
      <name>GigabitEthernet0/0</name>

      <description>Uplink Interface</description>
     </interface>
    </config>
   </edit-config>
  </rpc>
```

YANG (Yet Another Next Generation)

Definition:
YANG is a **data modeling language** used to describe the configuration and operational data of network devices managed by NETCONF.

Purpose

- Defines *structure and constraints* of the data that can be configured.
- Used to create **standardized data models** for devices and protocols.

Features

- Hierarchical, similar to XML or JSON.
- Defines data types, default values, constraints.
- Promotes interoperability among vendors.

Example YANG Model

```
    module interfaces { container interfaces {

      list interface { key "name"; leaf name {

     type string;

        }
        leaf description { type string;

        }

      }

     }

    }
```

➞ Describes a model for configuring interfaces.

Relationship Between NETCONF and YANG

- **YANG** defines the data structure.
- **NETCONF** transports and manipulates the data according to YANG definitions.

Advantages of NETCONF + YANG

- Vendor-neutral and standardized.

- Supports full configuration lifecycle (create, update, delete).
- Enables network automation and SDN integration.
- Provides validation and rollback mechanisms.

Basic Building Blocks of IoT Physical Devices

An IoT device connects the physical world to the digital world — sensing data, processing it, and acting on it or sending it to the cloud.

Core Components of an IoT Device

a) Sensors

- Detect physical parameters like temperature, motion, pressure, light, humidity, etc.
- Convert real-world data into electrical signals.
- **Example:** DHT11 (temperature & humidity sensor), PIR sensor (motion).

b) Actuators

- Perform actions based on received signals or data processing.
- **Example:** Motors, LEDs, valves.

c) Microcontroller / Processor

- Central control unit that processes sensor data and makes decisions.
  - **Examples:**
    - **Microcontrollers:** Arduino, ESP8266, ESP32.
    - **Microprocessors:** Raspberry Pi, NVIDIA Jetson Nano.

d) Communication Module

- Enables connectivity to networks.
- **Wired:** Ethernet, Serial.
- **Wireless:** Wi-Fi, Bluetooth, ZigBee, LoRa, Cellular (4G/5G), NFC.

e) Power Supply

- Provides energy to the device.
- Battery-powered, solar-powered, or direct AC/DC supply.
- Power efficiency is critical for IoT.

f) Cloud / Edge Interface

- IoT devices send data to cloud platforms (e.g., AWS IoT, Azure IoT, Google Cloud IoT).
- **Edge Computing:** Processes data locally to reduce latency and bandwidth use.

Popular Devices

pcDuino

The **pcDuino** is a mini PC platform developed by **LinkSprite**. It combines the features of a personal computer with those of an Arduino.
It runs **Linux** or **Android OS** and supports **Arduino-style programming**, making it ideal for both software and hardware projects.
Key Features:

- **Processor:** ARM Cortex-A8 (1 GHz)
- **Memory:** 1 GB DDR3 RAM
- **Storage:** MicroSD card slot + onboard flash
- **Operating System:** Ubuntu Linux, Android
- **GPIO Pins:** Compatible with Arduino shields
- **Connectivity:** HDMI, Ethernet, USB, and UART

Advantages:

- Can run full desktop environments (like Ubuntu).
- Arduino-compatible I/O pins for hardware interfacing.
- Supports C/C++, Python, and Arduino IDE.

Applications:

- Media streaming devices
- Smart home controllers
- Robotics and automation systems

BeagleBone Black

The **BeagleBone Black (BBB)** is a low-cost, community-supported development board developed by **Texas Instruments**.
It is a powerful platform for developers and IoT engineers, known for **real-time processing capabilities** and **expandable hardware**.
Key Features:

- **Processor:** ARM Cortex-A8 @ 1 GHz (AM335x processor)
- **Memory:** 512 MB DDR3 RAM
- **Storage:** 4 GB onboard eMMC + microSD slot
- **Operating System:** Debian, Ubuntu, Android, Cloud9 IDE
- **Connectivity:** Ethernet, HDMI, USB, Serial, GPIO, I^{2}C, SPI, PWM

Advantages:

- Powerful for real-time applications using PRUs (Programmable Real-time Units).
- Onboard flash memory — boots without SD card.
- Open-source hardware and software.

Applications:

- Industrial automation
- Robotics (motor control, sensors)
- IoT data acquisition and monitoring systems

Raspberry Pi

The **Raspberry Pi** is one of the most popular and affordable single-board computers developed by the **Raspberry Pi Foundation (UK)**.
It is widely used for IoT, robotics, AI, and education due to its low cost and high flexibility.
Key Features:

- **Processor:** ARM Cortex-A72 (1.5 GHz, Quad-core – varies by model)
- **Memory:** 1 GB to 8 GB RAM (depending on model)
- **Storage:** microSD card slot
- **Operating System:** Raspberry Pi OS (Linux-based), Ubuntu, Windows IoT Core
- **Connectivity:** Wi-Fi, Bluetooth, Ethernet, HDMI, USB, GPIO

Advantages:

- Low cost and energy-efficient.
- Strong community support and extensive tutorials.
- Supports many programming languages: Python, C/C++, Java, Node.js.
- Easily interfaces with sensors and cameras.

Applications:

- Home automation (smart lights, security cameras)
- Weather monitoring systems
- AI and computer vision projects
- Educational and research projects

Arduino

The **Arduino** is a **microcontroller-based development board** designed for simple electronic and IoT projects.
It's easy to program and ideal for beginners, hobbyists, and embedded engineers.
Unlike SBCs (like Raspberry Pi), Arduino doesn't run an operating system — it runs one program at a time.
Key Features:

- **Microcontroller:** Varies by model (e.g., ATmega328P for Arduino Uno)
- **Clock Speed:** 16 MHz
- **Memory:** 2 KB SRAM, 32 KB Flash (for Uno)
- **Operating System:** None (firmware only)
- **Programming Language:** C/C++ (Arduino IDE)
- **Connectivity:** GPIO, UART, SPI, I^2C

Advantages:

- Very easy to learn and program.
- Large number of sensors and modules available.
- Open-source hardware and software.
- Extremely low power consumption.

Applications:

- Smart agriculture (soil moisture, temperature sensing)
- IoT prototypes
- Smart lighting or fan control

- Wearable health devices

Cubieboard

The **Cubieboard** is another ARM-based single-board computer similar to the Raspberry Pi but with more hardware power.
Developed by **Cubietech**, it supports multiple Linux distributions and Android.
Key Features:

- **Processor:** Allwinner A20 (Dual-core Cortex-A7 @ 1 GHz)
- **Memory:** 1 GB DDR3 RAM
- **Storage:** NAND flash + microSD slot + SATA interface (for HDD/SSD)
- **Operating System:** Android, Ubuntu, Fedora, Arch Linux
- **Connectivity:** Ethernet, HDMI, USB, GPIO, SATA

Advantages:

- High performance with SATA interface for external drives.
- Better multimedia and storage capabilities than Raspberry Pi.
- Supports both desktop and embedded Linux environments.

Applications:

- Media servers and network storage (NAS)
- Smart surveillance systems
- IoT gateways
- Data logging and industrial control

Comparison Table

Feature	**pcDuino**	**BeagleBone Black**	**Raspberry Pi**	**Arduino**	**Cubieboard**
Processor	ARM Cortex-A8 (1 GHz)	ARM Cortex-A8 (1 GHz)	ARM Cortex-A72 (1.5 GHz, Quad)	ATmega328P (16 MHz)	Allwinner A20 (Dual 1 GHz)
Memory	1 GB	512 MB	1 GB–8 GB	2 KB SRAM	1 GB
Storage	microSD + Flash	eMMC + microSD	microSD	Flash memory (32 KB)	NAND + SATA
OS Support	Linux, Android	Debian, Ubuntu	Raspberry Pi OS, Ubuntu	None (firmware)	Linux, Android
Connectivity	HDMI, Ethernet, USB	Ethernet, HDMI, GPIO	Wi-Fi, BT, HDMI, USB	GPIO, Serial	Ethernet, HDMI, SATA
Cost (Approx.)	Moderate	Moderate	Low	Very Low	Moderate
Best Use	IoT + Media	Industrial IoT	Education, IoT	Simple control systems	Storage & gateway IoT

IoT Security: Threats, Authentication, Data Integrity, and Secure Communication

The **Internet of Things (IoT)** refers to the interconnection of physical devices through the Internet, enabling them to collect and exchange data. Examples include smart homes, wearable health trackers, industrial sensors, and autonomous vehicles.
While IoT enhances convenience and efficiency, it also introduces significant **security and privacy challenges** because:

- Devices are resource-constrained (limited power, memory, and processing).
- They often operate unattended.
- Millions of devices are connected globally.

Thus, **IoT security** focuses on protecting data and devices from unauthorized access, misuse, or damage.

IoT Security Threats

IoT systems are vulnerable at **multiple levels** — device, network, and cloud.
Understanding these threats helps design strong defenses.

Device-Level Threats

These threats target the **hardware or firmware** of IoT devices.

1. **Physical Tampering:**
   Attackers gain physical access to devices and modify, damage, or extract data from them.
   Example: An attacker opens a smart meter to change electricity readings.
2. **Firmware Replacement:**
   Replacing the original firmware with a malicious version that gives attackers control.
3. **Side-Channel Attacks:**
   Exploiting signals (like power consumption or timing information) to extract encryption keys.
4. **Malicious Code Injection:**
   Injecting unauthorized code into the system to control or damage the device.

Network-Level Threats

These threats occur when data is transmitted between devices, gateways, and cloud servers.

1. **Eavesdropping (Sniffing):**
   Attackers intercept network traffic to steal sensitive information.
2. **Man-in-the-Middle (MitM) Attack:**
   The attacker secretly intercepts and alters communication between two entities.
   Example: Intercepting communication between a smart lock and a user's smartphone.
3. **Denial of Service (DoS) / Distributed DoS (DDoS):**
   Overloading IoT servers or gateways with traffic, making them unavailable.
   Example: The **Mirai Botnet (2016)** infected IoT cameras and routers to launch massive DDoS attacks.

4. **Routing Attacks:**
   In sensor networks, attackers modify routing information to disrupt data transmission.

Application-Level Threats

These affect the software or application interfaces that interact with IoT devices.

1. **Malware and Ransomware:**
   Malicious software that encrypts or destroys data until a ransom is paid.
2. **Unauthorized Access:**
   Weak passwords or insecure APIs allow attackers to control devices remotely.
3. **Data Breaches:**
   Sensitive data (like health or location) may be leaked if not encrypted properly.
4. **Insecure Web Interfaces:**
   Poorly designed dashboards or mobile apps can expose IoT systems to attack.

Cloud-Level Threats

Cloud platforms that store and process IoT data can also be attacked.

1. **Data Leakage:**
   Insecure storage or misconfigured cloud databases can expose confidential data.
2. **Account Hijacking:**
   Attackers exploit weak authentication to access cloud services.
3. **Service Disruption:**
   Cloud servers may be attacked to disrupt IoT operations.

Authentication in IoT

Authentication ensures that devices and users are who they claim to be. Without proper authentication, unauthorized devices can join the network and steal or alter data.

Types of Authentication

1. **Password-Based Authentication:**
   - Most common and simple method.
   - Vulnerable to brute-force and dictionary attacks.
   - Should use strong passwords and regular updates.
2. **Two-Factor Authentication (2FA):**
   - Combines something the user *knows* (password) with something they *have* (OTP, smartcard) or *are* (biometric).
   - Provides stronger protection for IoT applications.
3. **Digital Certificates (Public Key Infrastructure - PKI):**
   - Devices use X.509 certificates to prove their identity.
   - Example: Smart city sensors communicating with the central server.
4. **Biometric Authentication:**
   - Uses fingerprints, voice, or facial recognition.
   - Common in smart locks and personal IoT devices.
5. **Device Authentication:**
   - Uses unique identifiers (UIDs) or cryptographic keys.
   - Example: A secure element or TPM (Trusted Platform Module) in IoT

hardware.

Authentication Protocols

Protocol	Purpose	Use Case
OAuth 2.0	Delegated access control	IoT applications sharing cloud data
OpenID Connect	Adds identity verification to OAuth	User authentication
TLS/SSL	Secure communication channel	HTTPS for IoT web interfaces
DTLS	TLS over UDP	CoAP protocol for constrained devices
EAP (Extensible Authentication Protocol)	Wireless network authentication	Wi-Fi based IoT devices

Data Integrity in IoT

Data integrity ensures that information remains **accurate, consistent, and unaltered** throughout its lifecycle.
If an attacker modifies data, the system may make wrong decisions (e.g., turning off an alarm when there is danger).

Methods to Ensure Data Integrity

1. **Hash Functions:**
   - Generate a fixed-length hash value from data.
   - Any change in data alters the hash.
   - Common algorithms: **SHA-256, SHA-512, MD5 (deprecated)**.
2. **Digital Signatures:**
   - Use private keys to sign messages; recipients verify them with public keys.
   - Ensures authenticity and integrity.
3. **Message Authentication Code (MAC):**
   - Combines a secret key with data to detect tampering.
4. **Blockchain:**
   - A decentralized ledger ensuring immutable and verifiable transactions.
   - Used in supply chain and healthcare IoT systems.
5. **Error Detection Codes:**
   - CRC (Cyclic Redundancy Check) detects transmission errors in IoT communication.

Secure Communication in IoT

Secure communication protects data during transmission between IoT devices, gateways, and cloud servers from interception and tampering.

Encryption Techniques

1. **Symmetric Encryption:**
   - Same key used for encryption and decryption.

   - Examples: **AES (Advanced Encryption Standard), DES (Data Encryption Standard)**.
   - Fast but requires secure key sharing.
2. **Asymmetric Encryption:**
   - Uses a pair of public and private keys.
   - Examples: **RSA, ECC (Elliptic Curve Cryptography)**.
   - ECC is widely used in IoT due to lower computational requirements.
3. **End-to-End Encryption (E2EE):**
   - Only sender and receiver can decrypt messages.
   - Prevents third-party access.

Secure Communication Protocols

Layer	**Protocol**	**Purpose**
Link Layer	WPA3 (Wi-Fi), Bluetooth LE Security, ZigBee Security	Protects wireless connections
Network Layer	IPsec, VPN, 6LoWPAN Security	Protects data at the IP level
Transport Layer	SSL/TLS, DTLS	Provides secure transport sessions
Application Layer	HTTPS, MQTT-S, CoAP (DTLS)	Secures communication between IoT applications

Example: Secure Communication Flow

1. IoT device encrypts sensor data using **AES**.
2. Data is transmitted via **MQTT** over **TLS** connection.
3. Cloud verifies the device identity using **digital certificates**.
4. Data is decrypted and stored securely in the cloud.