IOT-UNIT-4 NOTES

Syllabus :

Device Discovery and Cloud Services for IoT

Device discovery capabilities- Registering a device, Deregister a device, Introduction to Cloud Storage models and communication APIs Web-Server, Web server for IoT.

4. 1. Device discovery capabilities :

1) Network Scanning:

- IoT platforms can perform network scanning to identify devices that are connected to the network. This involves scanning IP addresses or network segments to detect devices and determine their availability for integration.

2) Auto-Registration:

- Some IoT platforms support automatic registration of devices.
  When a new device is connected to the network or communicates with the platform, it can trigger an automated registration process.
- The platform recognizes the device and initiates the necessary steps to register it, such as generating device identifiers, assigning security credentials, and adding it to the device inventory.

3) Device Metadata and Advertisement:

- IoT devices can provide metadata or advertise their presence on the network.
- This metadata may include information such as device type, capabilities, unique identifiers, and supported communication protocols. The IoT platform can listen for these advertisements and use the provided information to discover and onboard devices.

4) Discovery Protocols:

There are specific discovery protocols designed for IoT, such as mDNS (Multicast DNS) and SSDP (Simple Service Discovery Protocol). These protocols allow devices to announce their presence on the network, and IoT platforms can use them to discover and identify devices automatically.

5) API-Based Discovery:

- Some IoT platforms provide APIs (Application Programming Interfaces) that allow developers t query and discover devices based on specific criteria. The API endpoints provide methods to search for devices based on device type, attributes, or other parameters, enabling developers to programmatically discover and interact with devices.

6) Gateway or Hub Integration:

- In scenarios where IoT devices are connected through gateways or hubs, the discovery capabilities may involve communication between the gateway/hub and the IoT platform.
- The gateway or hub can report the presence of connected devices to the platform, allowing for centralized device management and control.

It's important to note that the availability and specific implementation of device discovery capabilities may vary depending on the IoT platform or system being used.

It's recommended to consult the documentation or support resources of the specific platform to understand the device discovery features and functionality it provides.

4.2. Registering a device for IoT :

(i) Choose an IoT platform or service:

Research and select an IoT platform or service that aligns with your requirements. There are numerous options available, such as AWS IoT, Google Cloud IoT, Microsoft Azure IoT, and many more.

(ii) Create an account:

Sign up for an account on the chosen IoT platform or service. This usually involves providing your email address, creating a username/password, and agreeing to the terms of service.

(iii) Set up your device:

Prepare your device for registration by connecting it to the necessary sensors, actuators, or other components. Make sure the device is powered on and ready for communication.

(iv) Configure device connectivity:

Depending on the IoT platform, you may need to configure network connectivity settings for your device. This could involve assigning an IP address, setting up Wi-Fi or cellular connectivity, or configuring MQTT (Message Queuing Telemetry Transport) settings.

(v) Generate device credentials:

In most cases, you'll need to generate security credentials for your device to establish a secure connection with the IoT platform. This typically involves generating a unique device ID, certificates, or access tokens.

(vi) Register your device:

Access the IoT platform's management console or API and locate the device registration section. Enter the necessary information, such as the device ID, credentials, and any additional metadata required.

(vii) Provision the device:

After registration, you may need to provision the device with additional configuration settings or metadata. This could include specifying its capabilities, defining data ingestion or telemetry settings, or assigning it to specific groups or deployments.

(viii) Test device connectivity:

Verify that your device can establish a connection with the IoT platform. This may involve sending a test message or telemetry data to ensure it is received correctly.

(ix) Integrate with IoT platform services:

Explore the various services provided by the IoT platform, such as data storage, analytics, rules engine, or device management. Integrate your device with these services to leverage the full capabilities of the IoT platform.

(x) Monitor and manage your device:

Once your device is registered and integrated with the IoT platform, you can monitor its status, receive telemetry data, and manage it remotely. This includes tasks like sending commands, firmware updates, or adjusting device settings.

Remember, the specific steps may vary depending on the IoT platform or service you choose. Consult the documentation and resources provided by the platform for detailed instructions on device registration and management.

4.3. Deregister a device for IoT :

➢ To deregister a device for IoT (Internet of Things), you typically need to follow these genera steps:

❖ **Access the IoT Platform:**

Log in to the IoT platform or service provider where the device is registered. Use your account credentials to gain access to the platform's dashboard or management console.

❖ **Locate the Device:**

Navigate to the device management section or the list of registered devices within the IoT platform. Look for the specific device you want to deregister.

❖ **Select the Device:**

Identify the device you wish to deregister from the list and select it. There might be checkboxes or buttons next to each device entry that allow you to perform actions on the device.

❖ **Deregister or Delete the Device:**

Once you have selected the device, look for an option to deregister, remove, or delete the device. The exact wording and location of this option may vary depending on the IoT platform you are using. It is commonly found in the device details page or in a context menu associated with the device entry.

❖ **Confirm the Deregistration:**

The platform may prompt you to confirm the deregistration or deletion of the device. Read any warnings or notifications displayed to ensure you understand the consequences of removing the device from the IoT platform. If you are certain about deregistering the device, confirm the action.

❖ **Review Deregistration Status:**

After confirming the deregistration, the IoT platform will process the request. The platform may provide a confirmation message indicating that the device has been successfully deregistered or removed from the system. Verify that the device is no longer listed or associated with your account.

❖ **Device Cleanup (Optional):**

Depending on the nature of the device and its connections, you may need to perform additional cleanup steps. For example, if the device was using certificates for secure communication, you might want to revoke or remove those certificates. If the device was

connected to any cloud resources or integrations, you may want to disable or remove those connections as well.

4.4. Introduction to Cloud Storage models and communication APIs Web-Server :

(i) Introduction to Cloud Storage models :

- **Object Storage:**

  Object storage is a popular cloud storage model that organizes data as objects, each with a unique identifier. Objects consist of data, metadata, and a key for retrieval. Object storage is highly scalable, allowing for the storage of massive amounts of data.

  It is commonly used for storing files, backups, images, videos, and other unstructured data. Examples of object storage services include Amazon S3, Google Cloud Storage, and Microsoft Azure Blob Storage.

- **File Storage:**

  File storage provides a hierarchical file system in the cloud, similar to traditional on-premises file servers. It enables the storage and retrieval of files using familiar file system protocols, such as Network File System (NFS) or Server Message Block (SMB).

  File storage is suitable for applications that require shared file access and compatibility with existing file-based workflows. Examples of file storage services include Amazon EFS, Azure Files, and Google Cloud File store.

- **Block Storage:**

  Block storage allows users to create and manage virtual block devices that can be attached to virtual machines or instances in the cloud. It offers low-level access to storage blocks and is commonly used by applications that require direct disk access or need to manage their own file systems.

  Block storage provides high performance and is often used for databases, transactional workloads, and virtual machine disks. Examples of block storage services include Amazon EBS, Azure Managed Disks, and Google Cloud Persistent Disks.

- Cloud storage models are models of cloud computing that stores data on the internet via cloud computing providers. These providers manage and operate data storage as a service.
- Cloud storage is basically an online storage of data. Data that is stored can be accessed from multiple connected devices, which constitute a cloud.

- Cloud storage can provide various benefits like greater accessibility and reliability, str protection of data backup, rapid deployment, and disaster recovery purposes.
- Moving to the cloud also decreases overall storage costs due to cutting costs incurred on the purchase of storage devices and their maintenance.
- As companies have started embracing the virtual disk model, the landscape of the data center is shifting.
- These models are pioneered in virtualization also providing new models that enable fully virtualized storage stacks.
- The cloud environment tries to provide a self-service with a precise separation between application and infrastructure.

- There are 3 more cloud storage models :
  - → **Instance storage:** Virtual disks in the cloud
  - → **Volume storage:** SAN sans the physical
  - → **Object storage:** Web-scale NAS

➢ Instance storage: Virtual disks in the cloud :

In a traditional virtualized environment, the virtual disk storage model is the eminent one. The nomenclature of this model is based upon this very reason, instance storage, meaning storage that is used like conventional virtual disks. It is crucial to note that instance storage is a storage model, not a storage protocol.

This storage can be implemented in numerous ways. For example, DAS (Direct-attached storage)is generally used to implement instance storage. It is often stated as ephemeral storage as the storage isn't highly reliable.

➢ Volume storage: SAN(Storage Area Network) sans the physical

Volume storage is also known as block storage. It supports operations like read/write and keeping the system files of running virtual machines. As suggested by its name, data is stored in structured blocks and volumes where files are split into equal-sized blocks. Each block has its own address.

However, unlike objects, they don't possess any metadata. Files are bifurcated into simpler blocks of fixed size, storing large amounts of data, which are dispensed amongst the storage nodes.

➢ Object storage: Web-scale NAS

Network-attached storage (NAS) is dedicated file storage that enables multiple users and heterogeneous client devices to retrieve data from centralized disk capacity.

Cloud-native applications need space, for storing data that is shared between different VMs. However, often there's a need for spaces that can extend to various data centers across multiple geographies which is catered by Object storage.

For example, Amazon Simple Storage Service (S3) caters to a single space across an entire region, probably, across the entire world.

Object storage stores data as objects, unlike others which go for a file hierarchy system. But it provides for eventual consistency.

Each object/block consists of data, metadata, and a unique identifier.

What object storage does differently is that it tries to explore address capabilities that are overlooked by other storages viz a namespace, directly programmable interface, data distribution, etc.

Object storage also saves a substantial amount of unstructured data. This kind of storage is used for storing songs on audio applications, photos on social media, or online services like Dropbox.

➢ **Conclusion :**

Now that we have a clear understanding of the system of cloud storage models. Each and every one of these has some benefits over one another with a few downsides but it depends on the kind of organization and their preference. While all three types of storage cannot be used by organizations, no single kind can cater to all the requirements.

(ii) Introduction to Communication APIs and Web Servers:

- Communication APIs and web servers play essential roles in enabling web-based applications and services. Here's an introduction to these concepts:

➢ **Communication APIs:**

Communication APIs (Application Programming Interfaces) are software interfaces that allow applications to communicate and interact with external systems or services.

They provide a set of functions, protocols, and tools that developers can use to integrate communication capabilities into their applications.

Communication APIs can facilitate various types of communication, such as voice calls, video calls, instant messaging, SMS, email, and more.

Examples of popular communication APIs include Twilio, Nexmo, and SendGrid.

- **Web Servers:**

Web servers are software applications that deliver web content to client devices, such as web browsers, upon receiving HTTP requests. They handle the processing of incoming requests, fetch the requested resources, and send the response back to the client.

Web servers can host static web pages, dynamic web applications, APIs, and other web-related services. Popular web servers include Apache HTTP Server, Nginx, Microsoft IIS, and Node.js (which includes a web server module).

Web servers can interact with communication APIs to enable real-time communication features within web applications. For example, a web server can integrate with a communication API to handle incoming voice or video calls, manage user authentication and authorization, send notifications, and process messaging services.

In summary, cloud storage models provide scalable and flexible options for storing data in the cloud, including object storage, file storage, and block storage.

Communication APIs enable developers to integrate communication capabilities, such as voice, video, and messaging, into their applications. Web servers handle HTTP requests and responses, serving web content and facilitating interactions with communication APIs for real-time communication within web applications.

- **Communication APIs for Web Servers :**
  - APIs take three basic forms:

→ Local APIs
→ Web APIs
→ Program APIs

(i) **Local APIs :**

Local APIs are the original form, from which the name came. They offer OS or middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local APIform.

(ii) Web APIs :

- Web APIs are designed to represent widely used resources like HTML pages and are accessed using a simple HTTP protocol. Any web URL activates a web API.

  Web APIs are often called REST (representational state transfer) or RESTful because the publisher of REST interfaces doesn't save any data internally between requests. As such, requests from many users can be intermingled as they would be on the internet.

(iii) Program APIs :

Program APIs are based on remote procedure call (RPC) technology that makes a remote program component appear to be local to the rest of the software. Service oriented architecture (SOA) APIs, such as Microsoft's WS-series of APIs, are programAPIs.

➢ Communication APIs for web servers enable interaction and data exchange between the server and external systems or clients. Some common communication APIs used with web servers include:

(i) HTTP(S) APIs:

Hypertext Transfer Protocol (HTTP) and its secure counterpart (HTTPS) are fundamental communication protocols for web servers. They define how clients (such as web browsers or IoT devices) request resources from the server and how the server responds with the requested data. HTTP(S) APIs are widely used for web applications, RESTful services, and IoT communication.

(ii) Web Socket:

Web Socket is a communication protocol that provides full-duplex communication channels over a single TCP connection. Unlike traditional HTTP, WebSocket enables real-time, bidirectional communication between the server and clients. It is commonly used for applications that require continuous data streaming, such as real-time chat applications or IoT dashboards.

(iii) MQTT:

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol designed for resource-constrained devices and networks. MQTT follows a publish-subscribe model, where clients (publishers) send messages to a broker, and other clients (subscribers) receive those messages. MQTT is widely used in IoT applications for efficient and reliable messaging.

(iv) Web RTC:

Web Real-Time Communication (WebRTC) is a collection of protocols and APIs that enable real-time communication directly between web browsers or devices. It provides peer-to-peer audio, video, and data communication capabilities, making it suitable for applications such as video streaming, and IoT device communication.

- These communication APIs provide different ways to establish and manage communication between web servers and clients, depending on the requirements of your application or IoT system.
- They offer various features and protocols to ensure efficient, secure, and real-time data exchange.

4.5. Web Server for IoT :

- When it comes to choosing a web server for IoT (Internet of Things) applications, several options are available, depending on your specific requirements and constraints.
- Here are a few popular web servers commonly used in IoT:

(i) Apache HTTP Server (Apache):

Apache is a widely used open-source web server that offers robust and scalable performance. It supports various programming languages, such as Python, PHP, and Java, making it flexible for IoT applications.

(ii) Node.js:

Node.js is a JavaScript runtime that allows you to build scalable and event-driven applications. It is commonly used in IoT due to its lightweight and efficient nature. With the help of frameworks like Express.js, you can quickly build RESTful APIs and handle HTTP requests.

(iii) NGINX:

NGINX is a high-performance, lightweight web server known for its excellent scalability and reverse proxy capabilities. It can handle a large number of concurrent connections, making it suitable for IoT deployments with high traffic or resource constraints.

(iv) Eclipse Mosquitto:

Mosquitto is an open-source MQTT (Message Queuing Telemetry Transport) broker that can be used as a lightweight web server for IoT messaging protocols. MQTT is commonly used in IoT for its low overhead and efficient pub/sub messaging pattern.

(v) Python-based frameworks:

Python is a popular language for IoT development due to its simplicity and extensive library support. Frameworks like Flask and Django allow you to build web servers with ease. Flask is lightweight and suitable for smaller IoT projects, while Django provides a more comprehensive solution for larger-scale applications.

(vi) Microsoft Azure IoT Hub:

If you prefer a cloud-based IoT solution, Microsoft Azure IoT Hub offers a robust and scalable platform for managing IoT devices and processing data. It provides various communication protocols, including HTTPS and MQTT, to interact with IoT devices via a web server hosted in the cloud.

These are just a few examples, and the choice of web server depends on factors such as the specific requirements of your IoT application, scalability needs, programming language preferences, and resource constraints .