

## **UNIT IV**

### **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.

#### **Explanation:**

##### **1. Device discovery capabilities:**

The Internet of Things (IoT) ecosystem has given consumers access to a world of possibilities, but it also comes with certain security risks.

With IoT devices constantly monitoring and collecting data concerning the user and device behaviour, the probability of malicious or ransomware attacks by hackers and other ill-intended entities has increased.

In fact, according to Kaspersky, some 1.51 billion IoT breaches occurred in the first six months of 2021, with most using the telnet remote access protocol.

Device discovery tools and IoT security solutions help address the said security challenges for consumer-level and industrial applications.

'Your IoT device discovery and security solution must go beyond simple device-finding and be capable of threat detection or endpoint profiling.' - Intuz

IoT security systems are indeed needed to protect end-users from cyber threats.

The former allows the latter to leverage the power of IoT devices and networks while having complete control over their sensitive personal data.

IT and security professionals need systems that help them discover devices and effectively mitigate potential threats.

##### **The Role of IoT Device Discovery Discovery& Security:**

With the increased use of connected gadgets, it has become more obvious that most of them still have little or no security features in place.

They are vulnerable and can easily be compromised by hackers, giving them remote access to our sensitive personal details.

In fact, cyber threat is bound to grow as the usage of IoT devices reaches 75 billion by 2025. Naturally, specific measures have to be taken to keep cyber attacks to a minimum.

Most system admins have limited knowledge about the devices trying to connect to the web through their local network.

Even visitors or passers-by carry devices programmed to detect networks in their vicinity. These devices are continuously trying to connect with the available corporate network.

It is nearly impossible to vet and register every device manually. In addition, you may even accidentally approve unknown malicious devices.

Dynamic IoT device discovery and profiling automate identifying processes that allow specific devices on the network.

These IoT security solutions offer much-needed protection to your network, while IoT device discovery forms the critical foundation in establishing security.

Both can be included as modules within routers, gateways, UTM's, and other similar devices that allow inbound and outbound network traffic because these tools are backed by a knowledge base, which helps detect new devices even without any foreknowledge of them.

## **2. Registering a device, Deregister a device:**

A device is a "Thing" in "Internet of Things": a processing unit that is capable of connecting to the internet (directly or indirectly) and exchanging data with the cloud. A device registry is a container of devices with shared properties. For more details about devices and registries, see [Devices](#).

## **Creating a device registry**

To use Cloud IoT Core, you must create at least one device registry. You can create a registry using Google Cloud console, the API, or gcloud.

### **Console**[gcloud](#)[API](#)

1. Go to the **Registries** page in Google Cloud console.

Go to the **Registries** page

2. At the top of the page, click **Create a registry**.
3. Enter a **Registry ID** and select a [cloud region](#). For information on registry naming and size requirements, see [Permitted characters and size requirements](#).
4. Select the **Protocols** that devices in this registry will use to connect to Cloud IoT Core: MQTT, HTTP, or both.
5. Select a **Default telemetry topic** or create a new one.

The default topic is used for published telemetry events that don't have a subfolder or if the subfolder used doesn't have a matching [Cloud Pub/Sub](#) topic.

6. (Optional) If you want to publish separate streams of data from a device, add more telemetry topics.

Each topic maps to a subfolder specified in either the MQTT topic path or the HTTP request when the data is published.

To create additional telemetry topics:

- a. Click **Add more telemetry topics**, then click **Add topic and subfolder**.
- b. Select a Pub/Sub topic or create a new one.
- c. Enter a descriptive subfolder name.

For information on subfolder naming and size requirements, see [Permitted characters and size requirements](#).

7. (Optional) Select a **Device state topic** or create a new one. This topic can be the same as a telemetry topic, or can be used only for state data.

State data is published to Cloud Pub/Sub on a best-effort basis: if publication to the topic fails, it will not be retried. If no topic is defined, device state updates are still persisted internally by Cloud IoT Core, but only the last 10 states are retained.

For more information, see [Getting device state](#).

8. Click **Create** to continue.

## Getting device details

You can get details about one or more devices using Google Cloud console, the API, or gcloud.

### ConsolegcloudAPI

1. Go to the **Registries** page in Google Cloud console.  
Go to the **Registries** page
2. Click the ID of the registry for the device.
3. In the menu on the left, click **Devices**.
4. Click the ID of the device to go to the **Device details** page. This page summarizes recent device activity, including the last time a message was published and the time of the most recent error. This page also shows the [device numeric ID](#).
5. Click the **Configuration & state history** tab to see recent configuration versions and update times for the device.

The fields for the last heartbeat time and the last time a configuration was ACKed are for the MQTT bridge only. The HTTP bridge does not support heartbeat or explicit ACKs.

## Deleting devices and registries

You can delete devices and registries using Google Cloud console, the API, or gcloud. To delete a registry, first delete all the devices within it.

### ConsolegcloudAPI

You can delete one or more devices from the registry's list of devices.

To delete devices:

1. Go to the **Registries** page in Google Cloud console.

Go to the **Registries** page

2. Click the ID of the registry for the device.
3. In the menu on the left, click **Devices**.
4. Select each device you want to delete, then click **Delete**.
5. Confirm you want to delete the selected devices, then click **Delete**.

### **3. Introduction to Cloud Storage Models & Communication APIs:**

In truth, cloud computing and IoT are tightly coupled.

The growth of IoT and the rapid development of associated technologies create a widespread connection of —things. This has led to the production of large amounts of data, which needs to be stored, processed and accessed.

Cloud computing as a paradigm for big data storage and analytics.

While IoT is exciting on its own, the real innovation will come from combining it with cloud computing.

The combination of cloud computing and IoT will enable new monitoring services and powerful processing of sensory data streams.

For example, sensory data can be uploaded and stored with cloud computing, later to be used intelligently for smart monitoring and actuation with other smart devices.

Ultimately, the goal is to be able to transform data to insight and drive productive, cost-effective action from those insights.

The cloud effectively serves as the brain to improved decision-making and optimized internet-based interactions. However, when IoT meets cloud, new challenges arise.

There is an urgent need for novel network architectures that seamlessly integrate them.

The critical concerns during integration are quality of service (QoS) and quality of experience (QoE), as well as data security, privacy and reliability.

The virtual infrastructure for practical mobile computing and interfacing includes integrating applications, storage devices, monitoring devices, visualization platforms, analytics tools and client delivery.

Cloud computing offers a practical utility-based model that will enable businesses and users to access applications on demand anytime and from anywhere.

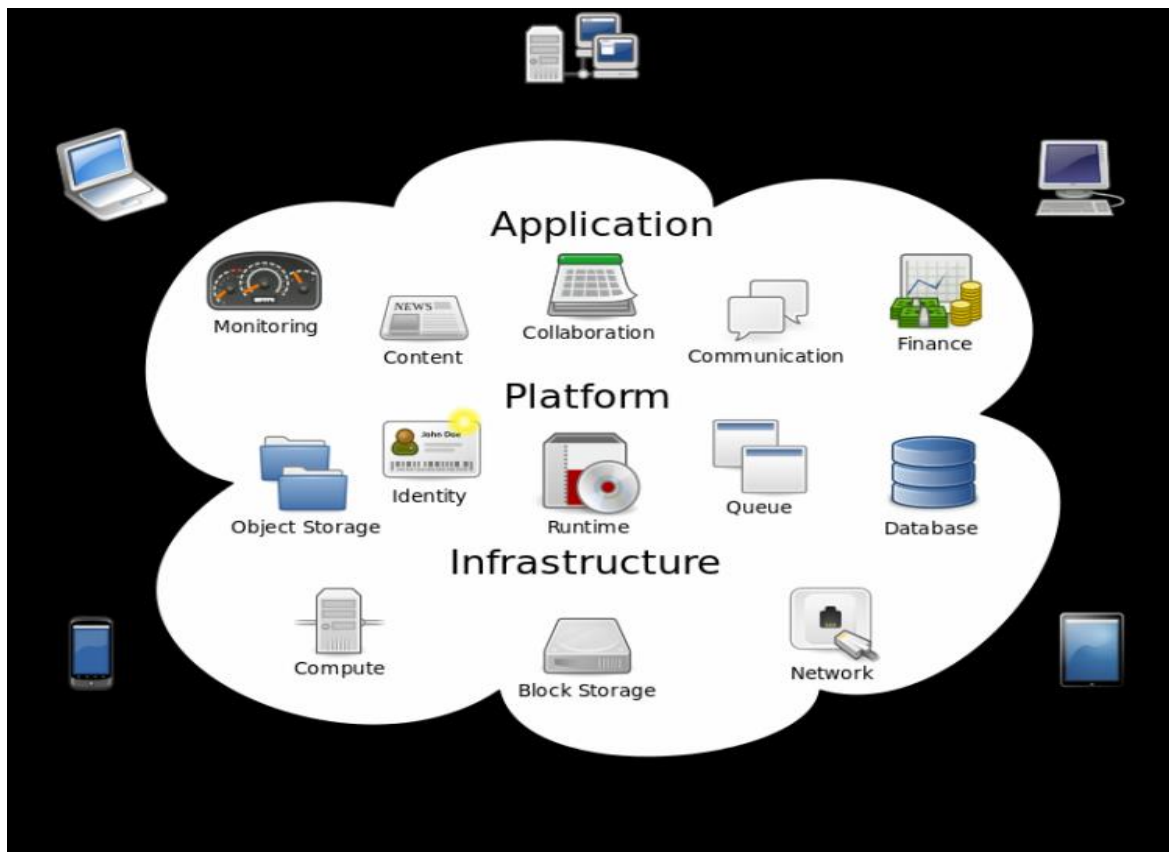
## Deployment models

Deployment in cloud computing comprises four deployment models: private cloud, public cloud, community cloud and hybrid cloud.

A cloud storage API is an application program interface that connects a locally-based application to a cloud-based storage system, so that a user can send data to it and access and work with data stored in it.

To the application, the cloud storage system is just another target device, like tape or disk-based storage.

An application program interface (API) is code that allows two software programs to communicate with each other.



The API defines the correct way for a developer to write a program that requests services from an operating system (OS) or other application.

APIs are implemented by function calls composed of verbs and nouns. The required syntax is described in the documentation of the application being called.

## Three basic types of APIs:

**APIs take three basic forms: local, web-like and program-like.**

**1. 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 API form.

**2. 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.

**3. 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 program APIs.

## **4. Web server for IoT:**

Any computer that can implement http or https is able to play the role of a web server.

Http is a protocol, a way of communication which supplies web pages. It is pretty widely used and easy to implement.

Through http you can transfer html and create simple user interfaces, it can implement Java Script and make more complicated web pages and it is available in most of the browsers.

One of the great qualities of this protocol is that it replaced complicated and heavy displays with user friendly web pages.

How does it work? The browser sends a request to the server who searches the demanded page and returns it to the browser for the user.

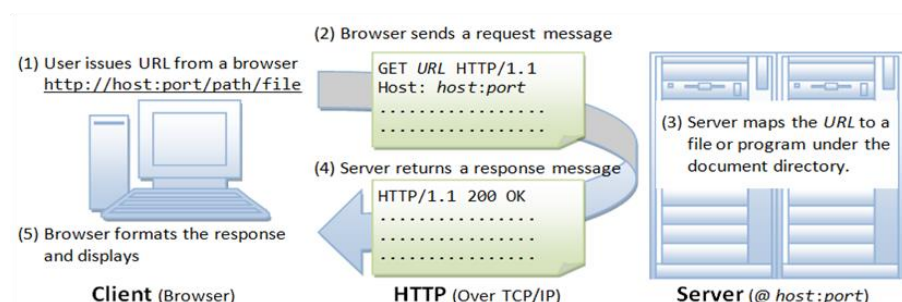
The request will consist of information about the kind of browser that is used, about the computer or about the document requested.

It will have a method, a URL, a query string and the upload body in case you want data to be sent to the server.

The response will include the status, which tells the browser if the page was found or not (the errors among the 400s are about a not found page, 300 are redirections and 200s are confirmations of the page being found). Https has two important security roles.

- It encrypts the data. The request and the response will be both encrypted on sending and decrypted when read.
- The server is always asked for a certificate of authenticity before it is asked for a page. This prevents against stolen data through false web pages.

What does a query consist of? It will always look like



this: [http://address:\[port\]URL?querystring](http://address:[port]URL?querystring). The port can be absent, in which case it will be 80 for http and 443 for https. It has to be specified if it is not one of the two. Concerning the URL, when it is not written, the default will be /. The available methods in http are : get, post, put and delete. The main ones being the first two.

- Get method needs no upload body. It will only ask for data from the server and send only the headers, the address, the URL.
- Post sends important data to the server, which will be uploaded. Post has the role of modifying data on the server. The response of both these methods is the page and any additional information that was requested.
- Put is similar to post, only that in the semantic way, this method only creates an object on the server.
- Delete also plays a semantic part. It needs no upload body and it deletes objects on the server. The same action can be performed however using get.

On one server there can be more than one websites, which means that, if the host is not specified in the request, the response may not be the one the browser expects.

Also, the response may have more than text. Any additional feature: images, JavaScript objects and so on will need a new request, so the process will be slowed down.

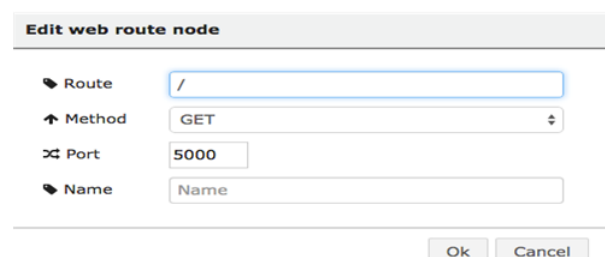
### Webserver on gadgets using Wyliodrin

---

The boards are non powerful computers. With wyliodrin there is no need to install any software or make any configuration on the boards to run a webserver on them.

To create a webserver in wyliodrin you will need a *web* node. The simplest way to use a web page in this particular way is to send static files. In the project files, create a new folder *static*. Everything inside it will be sent back to the browser by the server, regardless of the fact that they are html, Java Script or CSS files. Images can be added as well, but they will definitely make the process slower. There are other ways of adding an image. For example by using a storage system and including the images from there. This method will solve speed and memory issues.

The *web node*: The route option is actually the URL. The webserver will be active when it stumbles upon the specified route. Afterwards you choose the method, and write the port to setup the server. This port will only be used once, in the beginning.



Edit web route node	
Route	/
Method	GET
Port	5000
Name	Name
<div>Ok Cancel</div>	

The payload goes either in the query string for the get method or in the upload data for post. The message is built on this payload, on two mandatory variables: *res* which stands for the response and *req* which is the request. Without the last two, the server won't be able to

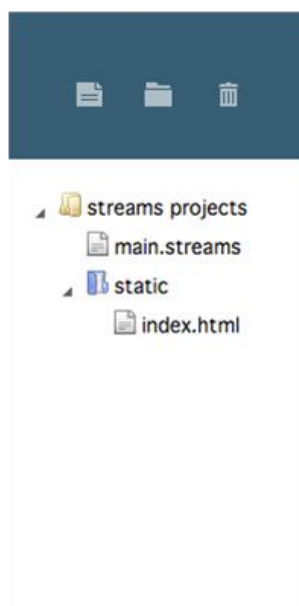
provide a response.



The *web response* node: The message received by this node comes from one web node. For a web response the simplest way is to make a redirection. Which means, in the redirect field, you can write the path to one of the static files and the browser will be sent to this page. On top of these, you will need the board's IP address which might not be public unless it is in the same network with the web server.

**Edit web response node**

Status/Error	<input type="text" value="200"/>
Redirect	<input type="text" value="/static/index.html"/>
Name	<input type="text" value="Name"/>



As a solution, IOT servers have a public address. The port for these servers can be either 80 for http or 443 for https. The user accesses the public page, through the IOT server which is connected to Wylidrin as well as the board. Now the problem with the board and the web being in the same network is solved as both can communicate with Wylidrin.

*Web templates*: Just as for the static files, you will need a *templates* folder. This time, when you use the node, you don't need the whole path. You can only write the name of the file in the templates folder. What does the node do? It processes the response, meaning it loads the values plus the payload in it and sends it back to the browser. The values need to be in between two sets of curly brackets `{{}}`. Note that the values won't update unless the page is reloaded.



Edit web template node

Template

index.html

Name

Name

Ok

Cancel

streams projects

main.streams

static

templates

index.html

## Web services

A long time ago, the web services were more complicated. Now the application only requests the web server for the data, and it is the browser's job to rearrange it so that it is in the right format for the application.

How to implement it into a Wylidrin application? Using a simple web response and web server node, you send a static page to the user and each time you make a query, instead of a template, you use a web response node and send the payload to the browser, which can be a number, an object or anything else.

## JQuery

There is a library called JQuery, based on JavaScript, thus available in any browser, which can make function calls to the server.

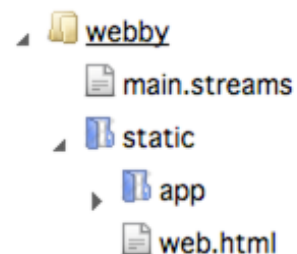
*Case study* : You have the following situation: you change the payload into a variable which stores values from a sensor. You want this variable to be shown in your web page. Practically, when an API gets called, what you will do, will be to make a get request to the server using the web address that you want with the URL */sensor* . The web page will send values that you will store in a variable in your html file.

## Web sockets

A web socket is based on the http or https protocol. It builds a connection between the browser and the server, so that either one can send data. When the browser makes a request, the server recognises the socket and doesn't close the connection. The two parties send the packages they need to send. If the server does not know how to work with sockets, the socket.io will go back to querying.

## AngularJS

AngularJS is a library through which you can build browser applications. In the next example, every web node will create a new socket and serve a static web page. If you include in the response a variable, and this variable changes, the Wylodrin controller will be notified every time this kind of novelty appears and AngularJS will replace the old value of the



variable with the new one, creating a dynamic web page.



```
1 <!DOCTYPE html>
2 <html ng-app="myApp">
3   <head>
4     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
5     <script src="/socket.io/socket.io.js"></script>
6     <script src="/static/app/controllers/wylodrincontroller.js"></script>
7   </head>
8   <body ng-controller="WylodrinCtrl">
9     {{random}}
10  </body>
11 </html>
```