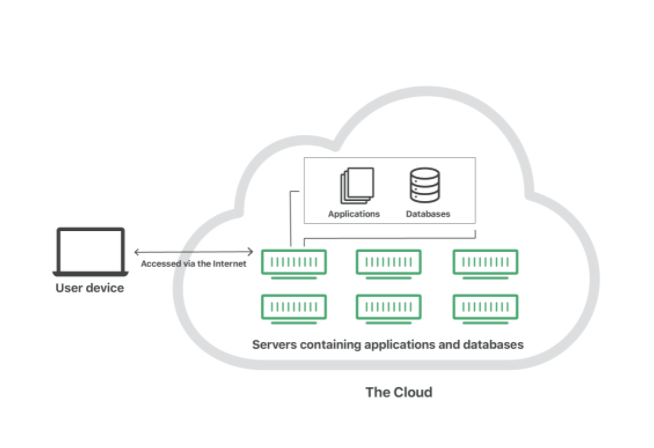
**Cloud setup and Communication**

**What is Cloud?**

"The cloud" refers to servers that are accessed over the Internet, and the software and databases that run on those servers. Cloud servers are located in data centres all over the world. By using cloud computing, users and companies don't have to manage physical servers themselves or run software applications on their own machines.



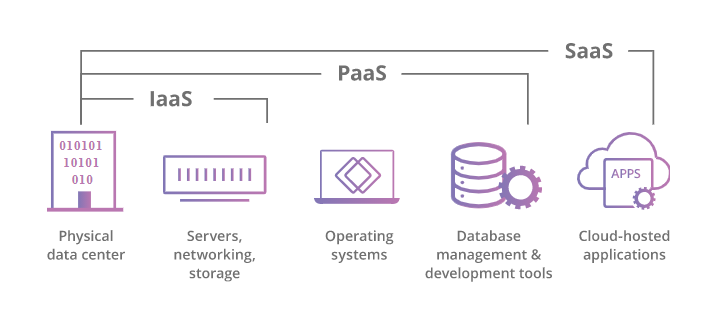
The cloud enables users to access the same files and applications from almost any device, because the computing and storage takes place on servers in a data centre, instead of locally on the user device. This is why a user can log into their Instagram account on a new phone after their old phone breaks and still find their old account in place, with all their photos, videos, and conversation history. It works the same way with cloud email providers like Gmail or Microsoft Office 365, and with cloud storage providers like Dropbox or Google Drive.

**How does it work?**

Cloud computing is possible because of a technology called virtualization. Virtualization allows for the creation of a simulated, digital-only "virtual" computer that behaves as if it were a physical computer with its own hardware. The technical term for such a computer is virtual machine. When properly implemented, virtual machines on the same host machine are sandboxed from one another, so they don't interact with each other at all, and the files and applications from one virtual machine aren't visible to the other virtual machines even though they're on the same physical machine.

Virtual machines also make more efficient use of the hardware hosting them. By running many virtual machines at once, one server becomes many servers, and a data centre becomes a whole host of data centres, able to serve many organizations. Thus, cloud providers can offer the use of their servers to far more customers at once than they would be able to otherwise, and they can do so at a low cost.

**Service Models**



**Software-as-a-Service (SaaS)**: Instead of users installing an application on their device, [SaaS](https://www.cloudflare.com/learning/cloud/what-is-saas/) applications are hosted on cloud servers, and users access them over the Internet. SaaS is like renting a house: the landlord maintains the house, but the tenant mostly gets to use it as if they owned it. Examples of SaaS applications include Salesforce, MailChimp, and Slack.

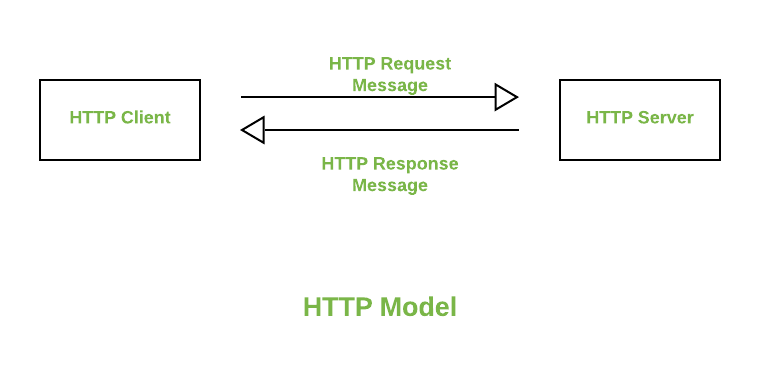
**Platform-as-a-Service (PaaS)**: In this model, companies don't pay for hosted applications; instead they pay for the things they need to build their own applications. [PaaS](https://www.cloudflare.com/learning/serverless/glossary/platform-as-a-service-paas/) vendors offer everything necessary for building an application, including development tools, infrastructure, and operating systems, over the Internet. PaaS can be compared to renting all the tools and equipment necessary for building a house, instead of renting the house itself. PaaS examples include Heroku and Microsoft Azure.

**Infrastructure-as-a-Service (IaaS)**: In this model, a company rents the servers and storage they need from a cloud provider. They then use that cloud infrastructure to build their applications. IaaS is like a company leasing a plot of land on which they can build whatever they want – but they need to provide their own building equipment and materials. IaaS providers include Digital Ocean, Google Compute Engine, and OpenStack.

|  |  |
| --- | --- |
| **Platform Type** | **Common Examples** |
| **SaaS** | Google Workspace, Dropbox, Salesforce, Cisco WebEx, Concur, GoToMeeting |
| **PaaS** | AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos, OpenShift |
| **IaaS** | DigitalOcean, Linode, Rackspace, Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE) |
|  |  |

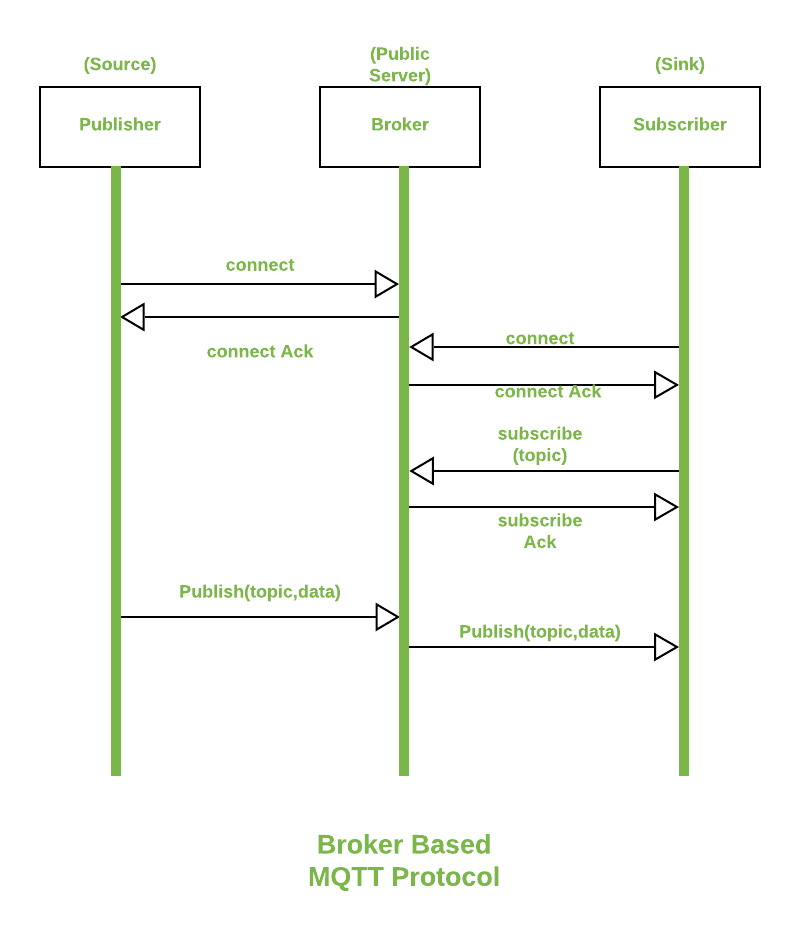
**Communication Protocols**

**HTTP**



It is used for defining how its messages are going to be transmitted and formatted. This protocol is responsible for the action that a server has to take while sending information over the network. When a URL is being entered into the browser, this protocol sends an HTTP request to the server and then an HTTP response is sent back to the browser. This protocol is also responsible for the controlling of webpages on the World Wide Web for their formatting and representation.

**MQTT**



It was created by Andy Standford-Clark and Arlen Nipper. It is an [IoT](https://www.geeksforgeeks.org/introduction-to-internet-of-things-iot-set-1/) interaction protocol based on the Publish/Subscribe model. This model is a simple model that provides support for QoS (Quality of Service). Due to its abilities, it can be found in every second IoT based device. This protocol has many features as it is over [TCP](https://www.geeksforgeeks.org/tcp-ip-model/) and uses SSL/TLS for security. For messaging between server it uses CONNECT, PUBLISH, SUBSCRIBE, DISCONNECT, etc.

**MQTT BASICS**

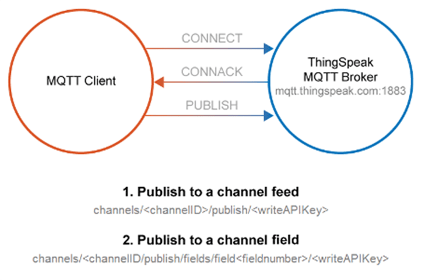
MQTT is a publish/subscribe architecture that is developed primarily to connect bandwidth and power-constrained devices over wireless networks. It is a simple and lightweight protocol that runs over TCP/IP sockets or WebSockets. MQTT over WebSockets can be secured with SSL. The publish/subscribe architecture enables messages to be pushed to the client devices without the device needing to continuously poll the server.

The MQTT broker is the central point of communication, and it is in charge of dispatching all messages between the senders and the rightful receivers. A client is any device that connects to the broker and can publish or subscribe to topics to access the information. A topic contains the routing information for the broker. Each client that wants to send messages publishes them to a certain topic, and each client that wants to receive messages subscribes to a certain topic. The broker delivers all messages with the matching topic to the appropriate clients.

ThingSpeak™ has an MQTT broker at the URL mqtt.thingspeak.com and port 1883. The ThingSpeak broker supports both MQTT publish and MQTT subscribe as shown in the diagrams.

**MQTT Publish**

The figure describes the topic structure. The Write API Key is required to publish. The broker acknowledges a correct CONNECT request with CONNACK.



**MQTT Subscribe**

The figure describes the topic structure. To subscribe to a public or private channel, you must also supply a user name and your MQTT API key as password when you connect. The broker acknowledges a correct SUBSCRIBE request with SUBACK.

