UNIT V IoT Physical Servers & Cloud Offerings

5.1 Introduction to Cloud Storage Models & Communication APIs –

5.2 WAMP - AutoBahn for IoT

5.3 Xively Cloud for IoT

5.4 Django

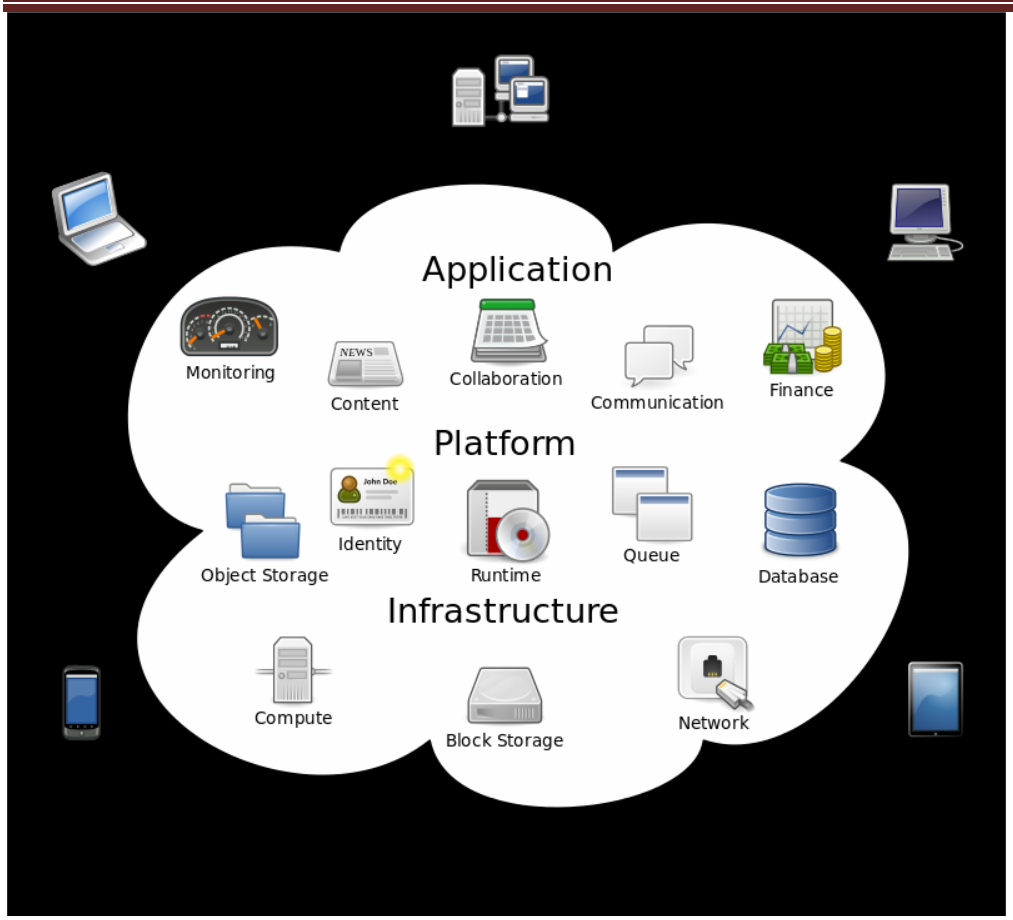
5.5 Designing a RESTful Web API

5.6Amazon Web Services for IoT

5.7 SkyNetIoT Messaging Platform.

5.1 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 technologiescreate a widespread connection of ―things.‖This has lead 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 qualityof 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 fromanywhere.



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 beingcalled

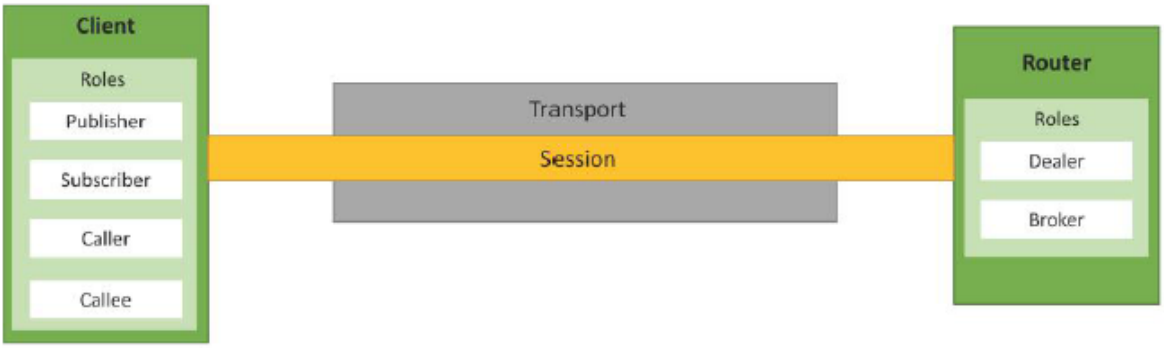
Three basic types of APIs
APIs take three basic forms: local, web-like and program-like.
1. **Local APIs** are the original form, from which the name came. They offer OS or middleware services to application programs. Microsoft's .NET APIs, the TAPI (Telephony API) for voice applications, and database access APIs are examples of the local APIform.

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 theinternet.

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 programAPIs.

5.2 **WAMP - AutoBahn for IoT**

Web Application Messaging Protocol (WAMP) is a sub-protocol of Websocket which provides publish-subscribe and remote procedure call (RPC) messaging patterns.



Transport: Transport is channel that connects two peers.

• Session: Session is a conversation between two peers that runs over a transport.

• Client: Clients are peers that can have one or more roles. In publish-subscribe model client can have following roles:

– Publisher: Publisher publishes events (including payload) to the topic maintained by the Broker.

– Subscriber: Subscriber subscribes to the topics and receives the events including the payload.

In RPC model client can have following roles:

– Caller: Caller issues calls to the remote procedures along with call arguments.

– Callee: Callee executes the procedures to which the calls are issued by the caller and returns the results back to the caller.

• Router: Routers are peers that perform generic call and event routing. In publish-subscribe model Router has the role of a Broker:

– Broker: Broker acts as a router and routes messages published to a topic to all subscribers subscribed to the topic.

In RPC model Router has the role of a Broker:

– Dealer: Dealer acts a router and routes RPC calls from the Caller to the Callee and routes results from Callee to Caller.

• Application Code: Application code runs on the Clients (Publisher, Subscriber, Callee or Caller).

In RPC model client can have following roles: –

1. Caller: Caller issues calls to the remote procedures along with call arguments. – Callee: Callee executes the procedures to which the calls are issued by the caller and returns the results back to the caller. • Router: Routers are peers that perform generic call and event routing. In publish-subscribe model Router has the role of a Broker: – Broker: Broker acts as a router and routes messages published to a topic to all subscribers subscribed to thetopic.

In RPC model Router has the role of a Broker: –

1. Dealer: Dealer acts a router and routes RPC calls from the Caller to the Callee and routes results from Callee toCaller.

2. Application Code: Application code runs on the Clients (Publisher, Subscriber, Callee or Caller).

5.3 Xively Cloud for IoT

Use of Cloud IoT cloud-based service •The service provides for the data collection, data points, messages and calculation objects. • The service also provisions for the generation and communication of alerts, triggers and feeds to the user. • A user is an application or service. The user obtains responses or feeds from the cloud service.

Pachube platform: for data capture in real-time over the Internet • Cosm: a changed domain name, where using a concept of console, one can monitor the feeds • Xively is the latest domain name.

A commercial PaaS for the IoT/M2M • A data aggregator and data mining website often integrated into the Web of Things • An IoT PaaS for services and business services.

Xively PaaS services:

• Data visualisation for data of connected sensors to IoT devices.

• Graphical plots of collected data.

• Generates alerts.

• Access to historical data

• Generates feeds which can be real-world objects of own or others

Xively HTTP based APIs

• Easy to implement on device hardware acting as clients to Xively web services

• APIs connect to the web service and send data.

• APIs provides services for logging, sharing and displaying sensor data of all

Xively Support

•The platform supports the REST, WebSockets and MQTT protocols and connects the devices to Xively Cloud Services

• Native SDKs for Android, Arduino, ARM mbed, Java, PHP, Ruby, and Python languages

• Developers can use the workflow of prototyping, deployment and management through the tools provided at Xively

Xively APIs

• Enable interface with Python, HTML5, HTML5 server, tornado

• Interface with WebSocket Server and WebSockets

• Interface with an RPC (Remote Procedure Call).

Xively PaaS services

• Enables services

• Business services platform which connects the products, including collaboration products

• Rescue, Boldchat, join.me, and operations to Internet

• Data collection in real-time over Internet

Xively Methods for IoT Devices Data

•Concept of users, feeds, data streams, data points and triggers

• Data feed typically a single location (e.g. a device or devices network),

• Data streams are of individual sensors associated with that location (for example, ambient lights, temperatures, power consumption).

• Pull or Push (Automatic or Manual Feed)

Xively Data formats and Structures

• Number of data formats and structures enable the interaction, data collection and services

• Support exists for JSON , XML and CSV

• Structures: Tabular, spreadsheet, Excel, Data numbers and Text with a comma-separated values in file

Xively Uses in IoT/M2M

• Private and Public Data Access

• Data streams, Data points and Triggers

• Creating and Managing Feeds

• Visualising Data

5.4 Django

Django is an open source web application framework for developing web applications in Python.

• A web application framework in general is a collection of solutions, packages and best practices that allows development of web applications and dynamic websites.

• Django is based on the Model-Template-View architecture and provides a separation of the data model from the business rules and the user interface.

• Django provides a unified API to a database backend.

• Thus web applications built with Django can work with different databases without requiring any code changes.

• With this fiexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem, Django is best suited for cloud applications.

• Django consists of an object-relational mapper, a web templating system and a regular-expression based URL dispatcher**.** Django is Model-Template-View (MTV) framework.

Model

• The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc. A Django model is a Python class that outlines the variables and methods for a particular type of data.

Template

• In a typical Django web application, the template is simply an HTML page with a few extra placeholders. Django's template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV, etc.)

View

• The view ties the model to the template. The view is where you write the code that actually generates the web pages.

View determines what data is to be displayed, retrieves the data from the database and passes the data to the template.

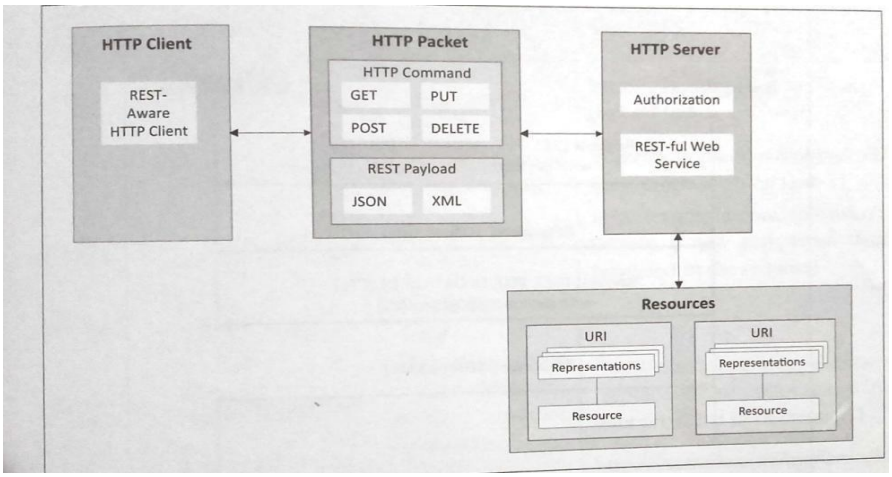
5.5 **Designing a RESTful Web API**

i) REST based communication APIs(Request-Response Based Model)

ii) WebSocket based Communication APIs(Exclusive Pair Based Model)

i) REST based communication APIs: Representational State Transfer(REST) is a set of architectural principles by which we can design web services and web APIs that focus on a system's resources and have resource states are addressed and transferred.

The REST architectural constraints are as follows: The below figure shows the communication between client server with REST APIs



Client-Server: The principle behind client-server constraint is the separation of concerns. Separation allows client and server to be independently developed and updated.

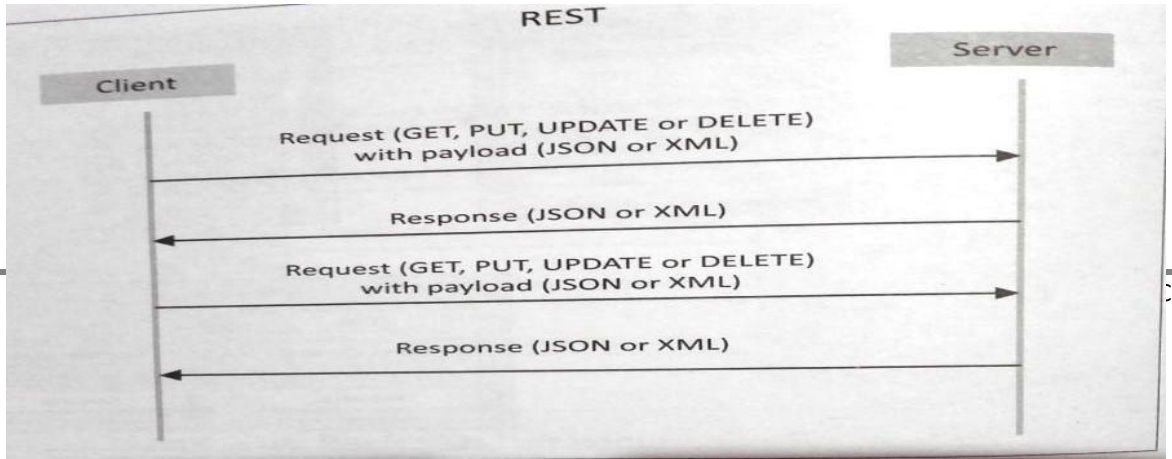
Stateless: Each request from client to server must contain all the info. Necessary to understand the request, and cannot take advantage of any stored context on the server.

Cache-able: Cache constraint requires that the data within a response to a request be implicitly or explicitly labeled as cache-able or non-cacheable. If a response is cache-able, then a client cache is given the right to reuse that response data for later, equivalent requests.

Layered System: constraints the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.

User Interface: constraint requires that the method of communication between a client and a server must be uniform.

Code on Demand: Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

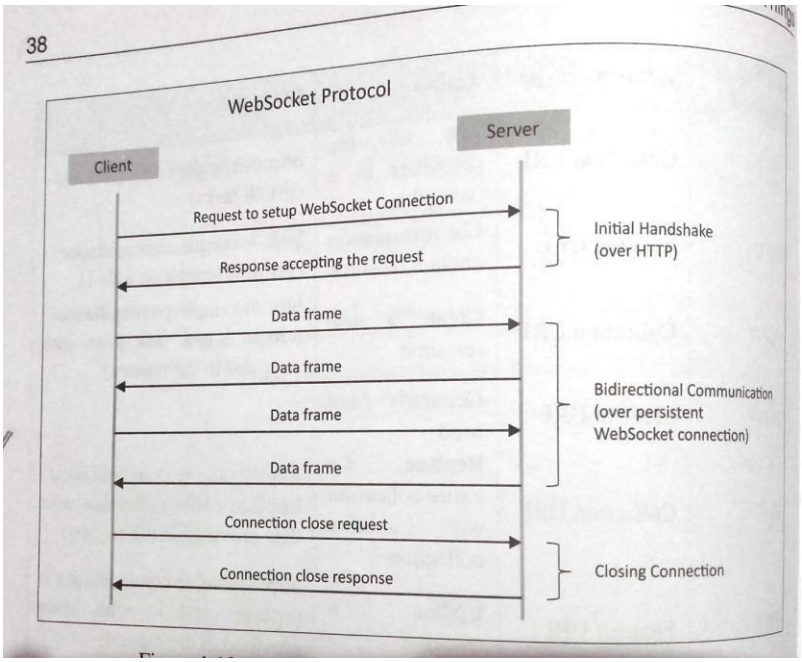


The Request-Response model used by REST:

RESTful web service is a collection of resources which are represented by URIs. RESTful web API has a base URI(e.g: http://example.com/api/tasks/). The clients and requests to these URIs using the methods defined by the HTTP protocol(e.g: GET, PUT, POST or DELETE). A RESTful web service can support various internet media types.

ii) WebSocket Based Communication APIs

WebSocket APIs allow bi-directional, full duplex communication between clients and servers. WebSocket APIs follow the exclusive pair communication model.



5.6Amazon Web Services for IoT

i) Amazon EC2

In this example, a connection to EC2 service is fi rst established by calling boto.ec2.connect_to_region.

• The EC2 region, AWS access key and AWS secret key are passed to this function. After connecting to EC2 , a new instance is launched using the conn.run_instances function.

• The AMI-ID, instance type, EC2 key handle and security group are passed to this function.

ii) Amazon AutoScaling

AutoScaling Service

• A connection to AutoScaling service is first established by calling boto.ec2.autoscale.connect_to_region function.

• Launch Configuration

• After connecting to AutoScaling service, a new launch configuration is created by calling conn.create_launch_con f iguration. Launch configuration contains instructions on how to launch new instances including the AMI-ID, instance type, security groups, etc.

AutoScaling Group

• After creating a launch configuration, it is then associated with a new AutoScaling group. AutoScaling group is created by calling conn.create_auto_scaling_group. The settings for AutoScaling group such as the maximum and minimum number of instances in the group, the launch configuration, availability zones, optional load balancer to use with the group, etc.

AutoScaling Policies

• After creating an AutoScaling group, the policies for scaling up and scaling down are defined.

• In this example, a scale up policy with adjustment type Change In Capacity and scaling_ad justment = 1 is defined.

• Similarly a scale down policy with adjustment type ChangeInCapacity and scaling_ad justment = -1 is defined.

CloudWatch Alarms

• With the scaling policies defined, the next step is to create Amazon CloudWatch alarms that trigger these policies.

• The scale up alarm is defined using the CPUUtilization metric with the Average statistic and threshold greater 70% for a period of 60 sec. The scale up policy created previously is associated with this alarm. This alarm is triggered when the average CPU utilization of the instances in the group becomes greater than 70% for more than 60 seconds.

• The scale down alarm is defined in a similar manner with a threshold less than 50%.

iii)Amazon S3:

• In this example, a connection to S3 service is first established by calling boto.connect_s3 function.

• The upload_to_s3_bucket_path function uploads the file to the S3 bucket specified at the specified path.

iv)Amazon RDS

In this example, a connection to RDS service is first established by calling boto.rds.connect_to_region function.

• The RDS region, AWS access key and AWS secret key are passed to this function.

• After connecting to RDS service, the conn.create_dbinstance function is called to launch a new RDS instance.

• The input parameters to this function include the instance ID, database size, instance type, database username, database password, database port, database engine (e.g. MySQL5.1), database name, security groups, etc.

v) Amazon Dynamo DB

In this example, a connection to DynamoDB service is first established by calling boto.dynamodb.connect_to_region.

• After connecting to DynamoDB service, a schema for the new table is created by calling conn.create_schema.

• The schema includes the hash key and range key names and types.

• A DynamoDB table is then created by calling conn.create_table function with the table schema, read units and write units as input parameters.

5.7 SkyNetIoT Messaging Platform.

SkyNet is running on a dozen Amazon EC2 servers and has nearly 50,000 registered smart devices including: Arduinos, Sparks, Raspberry Pis, Intel Galileos, and BeagleBoards, Matthieu said. SkyNet runs as an IoT platform-as-a-service (PaaS) as well as a private cloud through Docker, the new lightweight container technology. The platform is written in Node.js and released under an MIT open source license on GitHub.

The single SkyNet API supports the following IoT protocols: HTTP, REST, WebSockets, MQTT (Message Queue Telemetry Transport), and CoAP (Constrained Application Protocol) for guaranteed message delivery and low-bandwidth satellite communications, Matthieu said. Every connected device is assigned a 36 character UUID and secret token that act as the device's strong credentials. Security permissions can be assigned to allow device discoverability, configuration, and messaging.

Part A-Multiple Choice Questions

[Separately discussed]

Part B- 7 Marks

1. Elaborate the WAMP – AutoBahn for IoT.
2. Explain about Xively Cloud for IoT.
3. Draw the Django Architecture with explanation
4. How to Design a RESTful Web API? Explain

Part C- 16 Marks

1. List out the Amazon Web Services for IoT