Module-5

**1. Explain the Web Application Messaging Protocol (WAMP) – Auto Bahn for Internet of Things.**

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.

 Mainly used in cloud storage model for IoT & other messaging services

 WAMP is a routed protocol, with all components connecting to a *WAMP Router*, where the WAMP Router performs message routing between the component

 It is protocol for Web Socket (PUBSUB based protocol) : uses RPC Messaging Pattern

**Some Important Key Terminologies**

• Transport

• Session

• Clients (Publisher & Subscriber)

• Router

• Broker

• Dealer

• Application Code



**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. – 2. **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: – 3. **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).

**2. Summarize the Xively Cloud for IoT and explain in detail about the services provided along with creation of API’s**

Use of Cloud IoT cloud-based service

 Xively is the latest domain name Commercial Platform as a Service for the Internet of Things/M2M.

 Supports hundreds of platforms, millions of gateways and billions of smart devices.

 Xively provides Comprehensive and secure infrastructure, services

 Online development tools and dev center

 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 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



**3. Explain the Amazon Web Services(AWS) available for IoT, Discuss in brief about Amazon Auto Scaling services**

**i. Amazon EC2** In this example, a connection to EC2 service is first 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.

**i) 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%.

**ii) 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.

**iii) 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.

**iv) Amazon Dynamo DB**

In this example, a connection to DynamoDB service is first established by callingboto.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.

**SkyNet IoT 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.

**4. Explain in detail about the Amazon Elastic Compute Cloud(EC2) and explain with code snippet.**

**\*\*\*Amazon Elastic Compute Cloud (EC2) and Code snippet - Boto Python package that provides interfaces to Amazon Web Services (AWS).**

Amazon EC2 is a domain of computing in AWS, it provides secure and resizable compute capacity in cloud.

**Features of AWS EC2**

* Scaling (Instances can be scaled up and down)
* Integrated with other services (like S3, RDS)
* Pay for what you use
* Instances can be launched in one or more regions and AZ’s(Availability Zones)
* Support for different operating systems.
* Works with Amazon VPC(Virtual Private Cloud) to provide secure network and resources.

**BOTO**

Boto is a Python package that provides interfaces to Amazon Web Services (AWS)

**#Python program for launching an EC2 instance**

import boto.ec2

from time import sleep ACCESS\_KEY="<enter access key>" SECRET\_KEY="<enter secret key>"

REGION="us-east-1" AMI\_ID = "ami-d0f89fb9"

EC2\_KEY\_HANDLE = "<enter key handle>" INSTANCE\_TYPE="t1.micro"

SECGROUP\_HANDLE="default"

conn = boto.ec2.connect\_to\_region(REGION, aws\_access\_key\_id=ACCESS\_KEY, aws\_secret\_access\_key=SECRET\_KEY)

reservation = conn.run\_instances(image\_id=AMI\_ID, key\_name=EC2\_KEY\_HANDLE, instance\_type=INSTANCE\_TYPE, security\_groups = [ SECGROUP\_HANDLE, ] )

* In this example, a connection to EC2 service is first established by calling the *boto.ec2.connect\_to\_region* function.
* 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.

**5. Explain the below Amazon web services**

**i)Amazon EC2**

**ii)Amazon Auto scaling services**

**iii)Cloud watch Alarms**

**i. Amazon EC2** In this example, a connection to EC2 service is first 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.

***iii)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%.