UNIT-V (6 Lectures)

IoT PHYSICAL SERVERS & CLOUD OFFERINGS: Python Packages for IoT, WAMP - AutoBahn for IoT, Python Web Application Framework – Django,Amazon Web Services for IoT,SkyNet IoT messaging platform

Python Packages of Interest

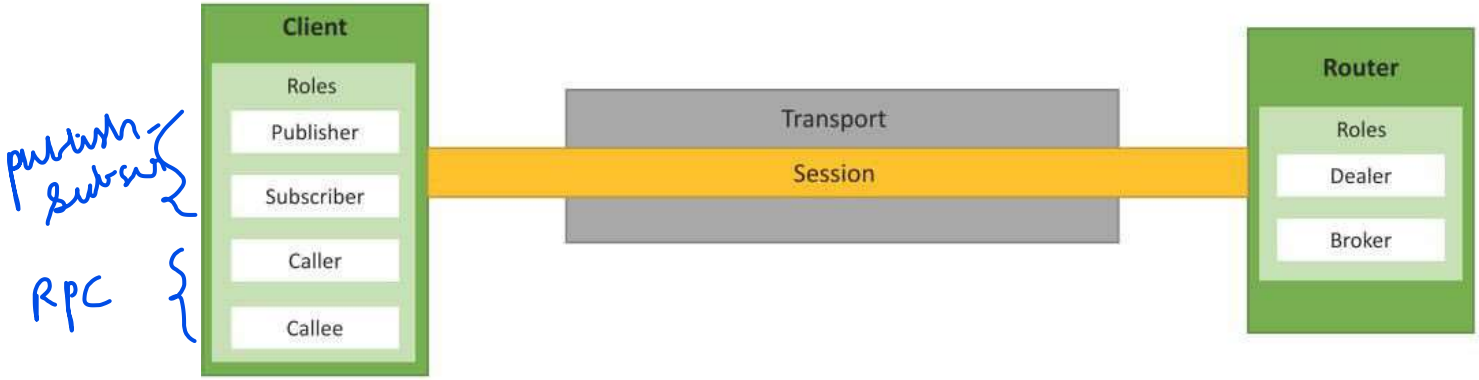
for iot

- JSON
  - JavaScript Object Notation (JSON) is an easy to read and write data-interchange format. JSON is used as an alternative to XML and is is easy for machines to parse and generate. JSONis built on two structures - a collection of name-value pairs (e.g. a Python dictionary) and ordered lists of values (e.g.. a Pythonlist).
- XML
  - XML (Extensible Markup Language) is a data format for structured document interchange. The Python minidom library provides a minimal implementation of the Document Object Model interface and has an API similar to that in other languages.
- HTTPLib & URLLib
  - HTTPLib2 and URLLib2 are Python libraries used in network/internet programming
- SMTPLib
  - Simple Mail Transfer Protocol (SMTP) is a protocol which handles sending email and routing e-mail between mail servers. The Python smtplib module provides an SMTPclient session object that can be used to send email.
- NumPy
  - NumPy is a package for scientific computing in Python. NumPy provides support for large multi-dimensional arrays and matrices
- Scikit-learn
  - Scikit-learn is an open source machine learning library for Python that provides implementations of various machine learning algorithms for classification, clustering, regression and dimension reductionproblems.

WAMP for IoT

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.



WAMP - Concepts

payload - Meta data

- Transport: Transport is channel that connects two peers.
- Session: Session is a conversation between two peers that runs over atransport.
- 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 RPCmodel 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.

Dealer

In RPCmodel Router has the role of a ~~Broker~~:

  - Dealer: Dealer acts a router and routes RPCcalls 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).~~

Book website: http://www.internet-of-things-book.com

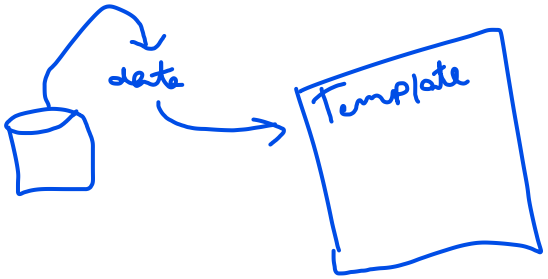
Python Web Application Framework - Django

- 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 flexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem, Django is best suited for IoT applications.
- Django consists of an object-relational mapper, a web templating system and a regular-expression- based URL dispatcher.

Django Architecture

- 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.



- When a new Django project is created a number of files are created as described below:
  - _init_.py: this file tells python that this folder is a python package
  - Manage.py: this file contains an array of functions for managing the site.
  - Settings.py: this file contains the website's settings.
  - Urls.py: this file contains the URL patterns that map URLs to pages.
- A Django project can have multiple applications. Apps are where code written that makes website functioned. Each project can have multiple apps and each app can be part of multiple projects
- When a new application is created a new directory for the application is also created which has a number of files including:
  - Model.py: this file contains the description of the models for the applications
  - Views.py: this file contains the application views

application's model ka description

Amazon web services for IoT:

Amazon EC2:

- Amazon EC2 is an Infrastructure –as- a – Service (IaaS) provided by Amazon.
- EC2 delivers scalable, pay as you go compute capacity in the cloud.
- EC2 is a web service that provides computing capacity in the form of virtual machines that are launched in amazon's cloud computing environment
- **Boto** is a Python package that provides interfaces to Amazon Web Services(AWS)
- **boto.ec2.connect_to_region :** connection to EC2service is first established by calling this 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 and it returns a reservation.
- The instances associated with a reservation is obtained by using **reservation.instances**
- The status of an instance associated with a reservation is obtained by using **instance.update** function.
- **conn.get_all_instances** function – get information on all running instances(reservations and IDs of instances associated with each reservations.
- **conn.stop_instances**: instances are stopped.

Amazon AutoScaling

- Amazon AutoScaling allows automatically scaling amazon EC2 capacity up or down according to user defined conditions.
- Therefore with autoscaling users can increase the number of EC2 instances running their applications seamlessly during spikes in the application workloads to meet the application performance requirements and scale down capacity when the workload is low to save costs.
- AutoScaling can be used for auto scaling IoT applications and IoT platforms deployed on Amazon EC2.
- ***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_configuration**. 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 ChangeInCapacity and scaling_adjustment = 1 is defined.
  - Similarly a scale down policy with adjustment type ChangeInCapacity and scaling_adjustment = -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%.

Amazon S3

- Amazon S3 is an online cloud based data storage infrastructure for storing and retrieving a very large amount of data.
- S3 provides highly reliable, scalable, fast, fully redundant and affordable storage infrastructure.
- S3 can serve as a raw data store(or "thing tank") for IoT systems for storing raw data, such as sensor data, log data, image, audio and video data.
- a connection to S3 service is first established by calling **boto.connect_s3 function**. this AWS access key and AWS secret key are passed to this function.
- The **upload_to_s3_bucket_path** function uploads the file to the S3bucket specified at the specified path.
- The **upload_to_s3_bucket_root** function uploads the file to the S3 bucket root.

Amazon RDS

- Amazon RDS is a web service that allows you to create instances of MySQL, Oracle or Microsoft SQL server in the cloud. With RDS developers can easily set up, operate, and scale a relational data base in the cloud.
- RDS can serve as a scalable data store for IoT systems. i.e. can store any amount of data
- 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.

- To create MySQL table uses the **MySQLdb** python package.
- **MySQLdb.connect** function is used to connect to the MySQL RDS instance, the end point of the RDS instance, database user name, password and port are passed to this function.
- After the connection to the RDS instance is established, a cursor to the data base is obtained by calling **conn.cursor**
- To execute the SQL commands for data base manipulation, the commands are passed to the **cursor.execute** function

Amazon DynamoDB

- Amazon DynamoDB is a fully- managed, scalable, high performance no-SQL database service.
- DynamoDb can serve as a scalable datastore for IoT systems.
- With DynamoDB, IoT system developers can store any amount of data and serve any level of frequent for the data.
- 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.

Writing and reading from a DynamoDB table:

- After establishing a connection with DyanamoDB service, the **conn.get_table** is called to retrieve an existing table.
- A new DynamoDB Table item is created by calling **table.new_item** and hash key and range key is specified.
- The data item is finally committed to DynamoDB by calling **item.put**
- To read data from DynamoDB, the **table.get_item** function is used with the hash key and range key as input parameters.

Amazon Kinesis

- Amazon Kinesis is a fully managed commercial service that allows real time processing of streaming data.
- Kinesis scales automatically to handle high volume streaming data coming from large number of sources.
- The streaming data collected by kinesis can be processed by applications running on Amazon EC2 instances or any other compute instance that can connect to kinesis.
- Kinesis is well suited for IoT systems that generate massive scale data and have strict real time requirements for processing the data.
- Kinesis allows rapid and continuous data intake and support data blobs of size upto 50Kb.
- The data producers write data records to kinesis streams.
- A data record comprises of a sequence number, a partition key and the data blob.

- Data records in a kinesis steam are distributed in shards. Each shard provides a fixed unit of capacity and a steam can have multiple shards.
- Single shard of throughput allows capturing 1MBps of data at upto 1,000 PUT transactions per second and allows applications to read data at up to 2MB per second.
- **Kinesis.put_record**: to write data to kinesis stream
- **Kinesis.get_shard_iterator**: to obtain shard iterator and it specifies the postion in the shard from which you want to start reading data records sequentially.
- **Kinesis.get_records**: to read data, which returns one or more data records from a shard.

Amazon SQS:

- Amazon SQS offers a highly scalable and reliable hosted queue for storing messages as they travel between distinct components of applications.
- SQS guarantees only that messages arrive, not that they arrive in the same order in which they were put in the queue.
- SQS and Kinesis seems to be almost similar, while kinesis is meant for real time applications that involve high data ingress and egress rates, SQS is simply a queue system that stores and releases messages in a scalable manner.
- Used In distributed IoT applications in which various application components need to exchange messages.

- A connection to SQS service is first established by calling **boto.sqs.connect_to_region**. The AWS region, access key and secret key are passed to this function.
- After connecting to SQS service, **conn.create_queue** is called to create a new queue with queue name as input parameter.
- **Conn.get_all_queues** : used to retrieve all SQS queues.
- **Queue.write**: used to write the message to the queue
- **Queue.read**: to read a message from a queue

Amazon EMR:

- Amazon EMR is a web service that utilizes Hadoop frame work running on Amazon EC2 and amazon S3.
- EMR allows processing of massive scale data, hence suitable for IoT applications that generate large volumes of data that needs to be analyzed.
- Data processing jobs are formulated with the MapReduce parallel data processing model.
- MapReduce is a parallel data processing model for processing and analysis of massive scale data.
- MapReduce has 2 phases: Map and Reduce
- MapReduce programs are written in functional programming style to create Map and Reduce functions. The input data to the map and reduce phases is in the form of key-value pairs.
- Consider an IoT system that collects data from a machine(or sensor data) which is logged in a cloud storage (such as Amazon S3) and analyzed on hourly basis to generate alerts if a certain sequence occurred more than a predefined number of times. Since the scale of data involved in such applications can be massive, MapReduce is an ideal choice for processing such data.

- Boto.emr.connect_to_region: used to establish connection to EMR service. The AWS region, access key and security key are passed to this function.
- After connecting to EMR service, a jobflow step is created. There are 2 types of steps- streaming and custom jar.
- To create a streaming job an object of the StreamingStep class is created by specifying the job name, locations of the mapper, reducer, input and output.
- The job flow is then started using the conn.run_jobflow function with streaming step object as input.
- When MapReduce job completes, the output can be obtained from the output location on the S3 bucket specified while creating the streaming step.

SkyNet IoT messaging system

- SkyNet is an open source instant messaging platform for IoT. The SkyNet API supports both HTTP REST and real-time websockets.
- SkyNet allows to register devices(or nodes) on the network.
- A device can be anything including sensors, smart home devices, cloud resources, drones et.
- Each device is assigned a UUID and a secret token. Devices or client applications can subscribe to other devices and receive/send messages.

■ Box 8.39: Commands for Setting up SkyNet

```
#Install DB Redis:
sudo apt-get install redis-server

#Install MongoDB:
sudo apt-key adv -keyserver keyserver.ubuntu.com -recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart
dist 10gen' |
sudo tee /etc/apt/sources.list.d/10gen.list
sudo apt-get update
sudo apt-get install mongodb-10gen

#Install dependencies
sudo apt-get install git
sudo apt-get install software-properties-common
sudo apt-get install npm

#Install Node.JS
sudo apt-get update
sudo apt-get install -y python-software-properties python g++ make
sudo add-apt-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs

#Install Skynet:
git clone https://github.com/skynetim/skynet.git
npm config set registry http://registry.npmjs.org/
npm install
```

Box 8.40: Sample SkyNet configuration file

```
module.exports = {
  databaseUrl:  "mongodb://localhost:27017/skynet",
  port:  3000,
  log:  true,
  rateLimit:  10, // 10 transactions per user per secend
  redisHost:  "127.0.0.1",
  redisPort:  "6379"
};
```

Box 8.41: Using the SkyNet REST API

```
#Creating a device
$curl -X POST -d "name=mydevicename&token=mytoken&color=green"
http://localhost:3000/devices
```

```
{"name":"mydevicename","token":"mytoken","ipAddress":"127.0.0.1",
"uuid":"myuuid","timestamp":1394181626324,"channel":"main",
"online":false,"_id":"531985fa16ac510d4c000006","eventCode":400}

---------------------
#Listing devices
$curl http://localhost:3000/devices/myuuid

{"name":"mydevicename","ipAddress":"127.0.0.1","uuid":"myuuid",
"timestamp":1394181626324, "channel":"main","online":false,
"_id":"531985fa16ac510d4c000006", "eventCode":500}

-----------------------
#Update a device
$curl -X PUT -d "token=mytoken&color=red"
http://localhost:3000/devices/myuuid

#Get last 10 events for a device
$curl -X GET http://localhost:3000/events/myuuid?token=mytoken

{"events":[{"color":"red","fromUuid":"myuuid",
"timestamp":1394181722052, "eventCode":300,"id":"mytoken"},{
"name":"mydevicename","ipAddress":"127.0.0.1" ,
"uuid":"myuuid",
"timestamp":1394181626324, "channel":"main","online":false,
"eventCode":500, "id":"531985fa16ac510d4c000006"}]}

-----------------------
#Subscribing to a device
$curl -X GET http://localhost:3000/subscribe/myuuid?token=mytoken

---------------------
#Sending a message
$curl -X POST -d '{"devices": "myuuid", "message":
{"color":"red"}}' http://localhost:3000/messages
```