IoE – ADVANCE LEARNER ASSIGNMENT 2

Name: Kazi Jawwad A Rahim *Roll No: 28*

Q.1. Using PHP create a RESTful API (refer- https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html)

ANS. Following are steps to create a RESTful API to create and delete a product using PHP.

1. Create product

Create create.php file

- Open product folder.
- Create a new create.php file.
- Open that file and put the following code inside it.

```
<?php
// required headers
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers,
Authorization, X-Requested-With");
// get database connection
include_once '../config/database.php';
// instantiate product object
include_once '../objects/product.php';
$database = new Database();
$db = $database->getConnection();
$product = new Product($db);
// get posted data
$data = json_decode(file_get_contents("php://input"));
// make sure data is not empty
if(
   !empty($data->name) &&
   !empty($data->price) &&
   !empty($data->description) &&
   !empty($data->category_id)
){
   // set product property values
   $product->name = $data->name;
   $product->price = $data->price;
   $product->description = $data->description;
   $product->category_id = $data->category_id;
   $product->created = date('Y-m-d H:i:s');
   // create the product
   if($product->create()){
      // set response code - 201 created
      http_response_code(201);
      // tell the user
```

```
        echo json_encode(array("message" => "Product was created."));
    }
    // if unable to create the product, tell the user
    else{
        // set response code - 503 service unavailable
        http_response_code(503);
        // tell the user
        echo json_encode(array("message" => "Unable to create product."));
    }
}
// tell the user data is incomplete
else{
    // set response code - 400 bad request
    http_response_code(400);
    // tell the user
    echo json_encode(array("message" => "Unable to create product. Data is incomplete."));
}
?>
```

Product create() method

- Open objects folder.
- Open product.php file.
- The previous section will not work without the following code inside the Product (objects/product.php) class.

```
// create product
function create(){
    // query to insert record
    $query = "INSERT INTO" . $this->table_name . " SET name=:name, price=:price,
description=:description, category_id=:category_id, created=:created";
    // prepare query
    $stmt = $this->conn->prepare($query);
    // sanitize
    $this->name=htmlspecialchars(strip_tags($this->name));
    $this->price=htmlspecialchars(strip_tags($this->price));
    $this->description=htmlspecialchars(strip_tags($this->description));
    $this->category_id=htmlspecialchars(strip_tags($this->category_id));
    $this->created=htmlspecialchars(strip_tags($this->created));
    // bind values
    $stmt->bindParam(":name", $this->name);
    $stmt->bindParam(":price", $this->price);
    $stmt->bindParam(":description", $this->description);
    $stmt->bindParam(":category_id", $this->category_id);
    $stmt->bindParam(":created", $this->created);
    // execute query
    if($stmt->execute()){
        return true;
    }
    return false;
}
```

Output

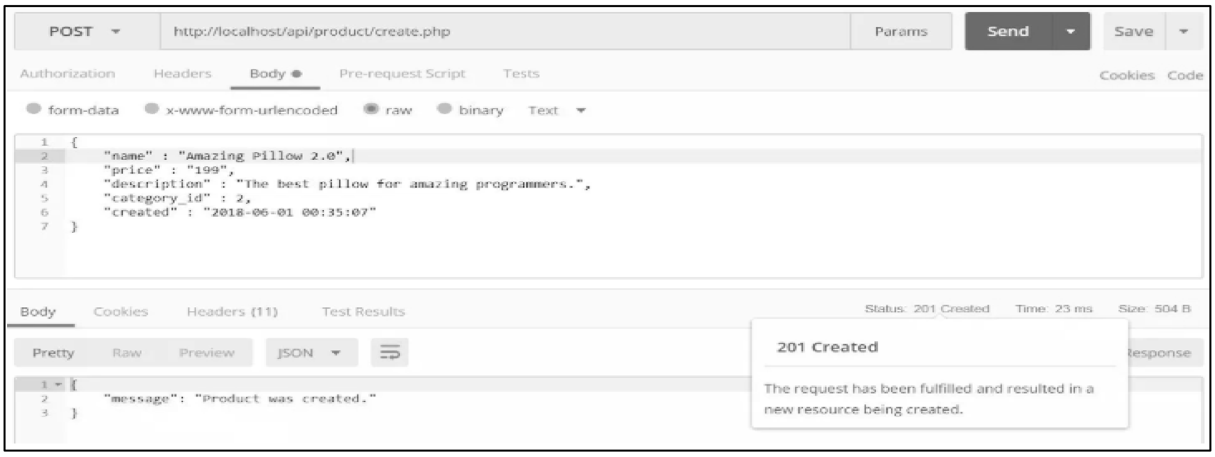
To test for the successful creation of a product, open POSTMAN. Enter the following as the request URL

http://localhost/api/product/create.php

- Click the "Body" tab.
- Click "raw".
- Enter this JSON value:

```
{
    "name" : "Amazing Pillow 2.0",
    "price" : "199",
    "description" : "The best pillow for amazing programmers.",
    "category_id" : 2,
    "created" : "2018-06-01 00:35:07"
}
```

It should look like this:



POST http://localhost/api/product/create.php

Body: raw

```
{
    "name" : "Amazing Pillow 2.0",
    "price" : "199",
    "description" : "The best pillow for amazing programmers.",
    "category_id" : 2,
    "created" : "2018-06-01 00:35:07"
}
```

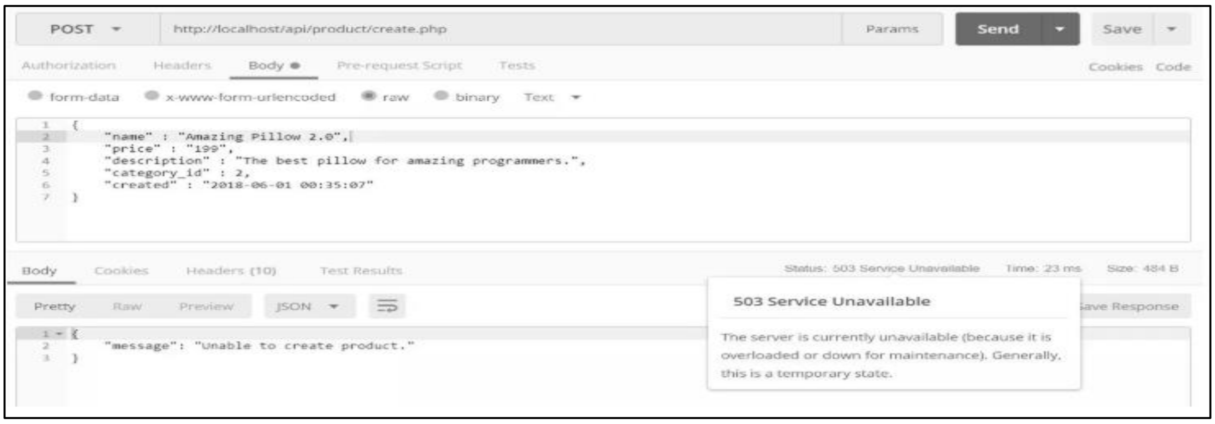
Status: 201 Created Time: 23 ms Size: 504 B

```
{
    "message": "Product was created."
}
```

201 Created

The request has been fulfilled and resulted in a new resource being created.

If the system is unable to create the product, it should look like this:



POST http://localhost/api/product/create.php

Body: raw

```
{
    "name" : "Amazing Pillow 2.0",
    "price" : "199",
    "description" : "The best pillow for amazing programmers.",
    "category_id" : 2,
    "created" : "2018-06-01 00:35:07"
}
```

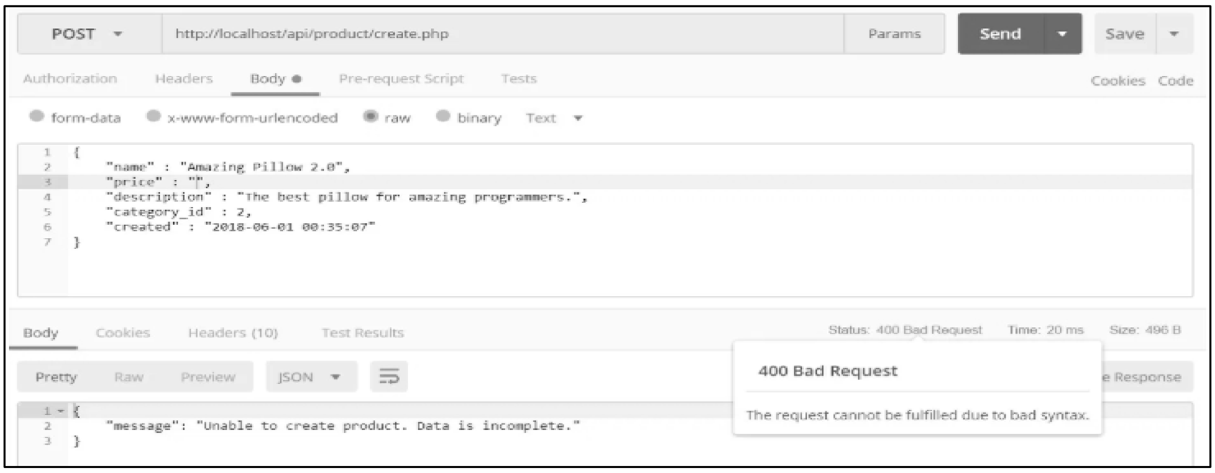
Status: 503 Service Unavailable Time: 23 ms Size: 484 B

```
{
    "message": "Unable to create product."
}
```

503 Service Unavailable

The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state.

If the sent data is incomplete, for example, it is missing the price data, the output should look like this:



http://localhost/api/product/create.php

```
{
    "name" : "Amazing Pillow 2.0",
    "price" : "",
    "description" : "The best pillow for amazing programmers.",
    "category_id" : 2,
    "created" : "2018-06-01 00:35:07"
}
```

Status: 400 Bad Request Time: 20 ms Size: 496 B

400 Bad Request

The request cannot be fulfilled due to bad syntax.

```
{
    "message": "Unable to create product. Data is incomplete."
}
```

2. DELETE PRODUCT

Create "delete.php" file

- Open product folder.
- Create new delete.php file.
- Open that file and put the following code inside it.

```
<?php
// required headers
header("Access-Control-Allow-Origin: *");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Allow-Methods: POST");
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers,
Authorization, X-Requested-With");
// include database and object file
include_once '../config/database.php';
include_once '../objects/product.php';
// get database connection
$database = new Database();
$db = $database->getConnection();
// prepare product object
$product = new Product($db);
// get product id
$data = json_decode(file_get_contents("php://input"));
// set product id to be deleted
$product->id = $data->id;
// delete the product
if($product->delete()){
    // set response code - 200 ok
    http_response_code(200);
    // tell the user
    echo json_encode(array("message" => "Product was deleted."));
}
// if unable to delete the product
else{
    // set response code - 503 service unavailable
    http_response_code(503);
    // tell the user
    echo json_encode(array("message" => "Unable to delete product."));
}
?>
```

Product delete() method

- Open objects folder.
- Open product.php file.
- The previous section will not work without the following code inside the Product class.

```
// delete the product
function delete(){
    // delete query
    $query = "DELETE FROM " . $this->table_name . " WHERE id = ?";
    // prepare query
    $stmt = $this->conn->prepare($query);
    // sanitize
    $this->id=htmlspecialchars(strip_tags($this->id));
    // bind id of record to delete
    $stmt->bindParam(1, $this->id);
    // execute query
    if($stmt->execute()){
        return true;
    }
    return false;
}
```

Output

Open POSTMAN. Enter the following as the request URL.
http://localhost/api/product/delete.php

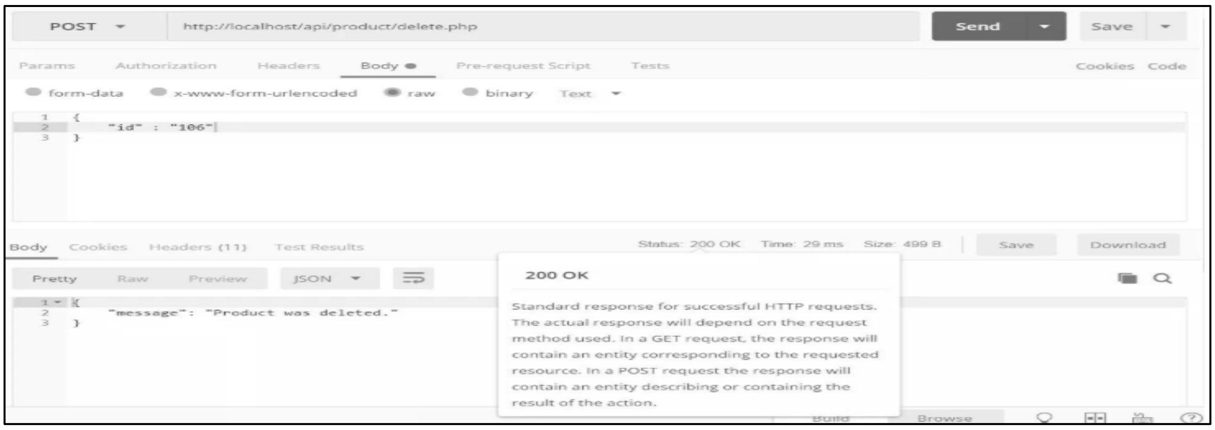
- Click the "Body" tab.
- Click "raw".
- Enter the following JSON value.

Make sure the ID exists in your database.
Click the blue "Send" button.

```
{
    "id" : "106"
```

}

If a product was successfully deleted, it should look like this:


POST http://localhost/api/product/delete.php

Body: raw, Text

```
{
    "id" : "106"
}
```

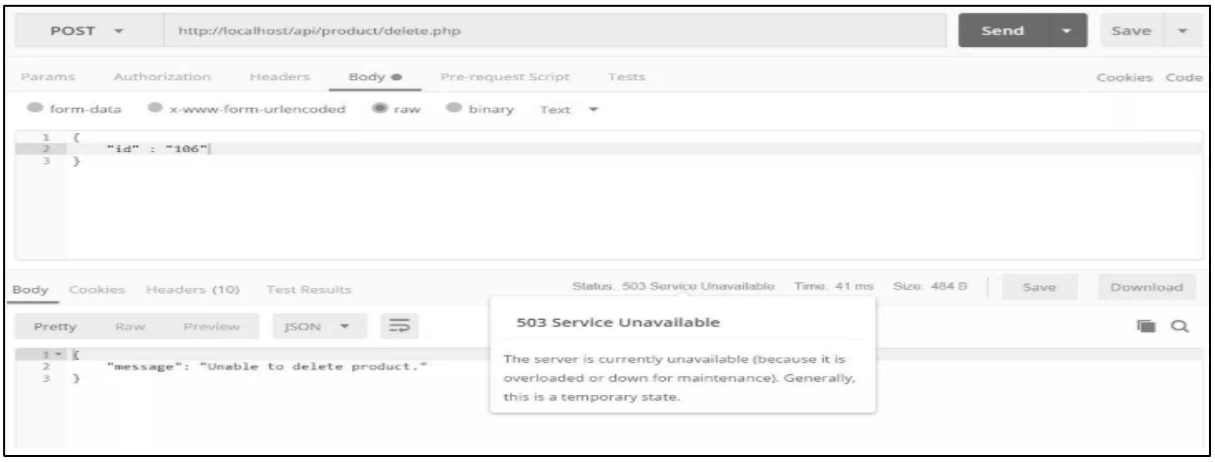
Status: 200 OK Time: 29 ms Size: 499 B

```
{
    "message": "Product was deleted."
}
```

200 OK

Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request the response will contain an entity describing or containing the result of the action.

If the system fails to delete the product, the output will look like this:


POST http://localhost/api/product/delete.php

Body: raw, Text

```
{
    "id" : "106"
}
```

Status: 503 Service Unavailable Time: 41 ms Size: 484 B

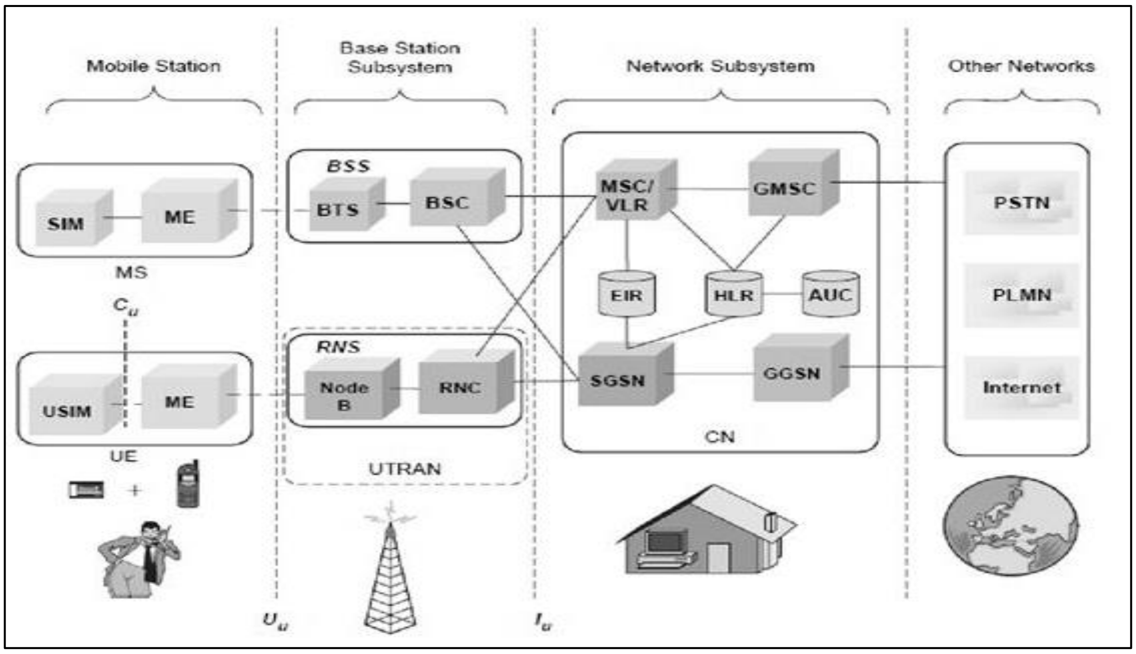
```
{
    "message": "Unable to delete product."
}
```

503 Service Unavailable

The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state.

Q.2. What is UMTS, explain with its architecture? (Content beyond Syllabus)

ANS. UMTS Architecture:



i. A UMTS system can be divided into a set of domains and the reference points that interconnect them.
ii. The UMTS network architecture is partly based on existing 2G network components and some new 3G network components. It inherits the basic functional elements from the GSM architecture on the core network (CN) side.
iii. The MS of GSM is referred as user equipment (UE) in UMTS. The MSC has quite similar functions both in GSM and UMTS. Instead of circuit-switched services for packet data, a new packet node SGSN is introduced. This SGSN is capable of supporting data rates of up to 2 Mbps.
iv. The core-network elements are connected to the radio network via the Iu interface, which is very similar to the A-interface used in GSM.
v. The major changes in the UMTS architecture are in the Radio Access Network (RAN), which is also called UMTS terrestrial RAN (UTRAN). There is a totally new interface called lur, which connects two neighboring Radio Network Controllers (RNCs). BSs are connected to the RNC via the lub interface.

UMTS terrestrial RAN (UTRAN)

i. UTRAN consist of Radio Network Subsystems (RNSs). The RNS has two main elements: Node B and a Radio Network Controllers (RNC). ii. Radio network controller (RNC):

- The RNC is responsible for control of the radio resources in its area. One RNC controls multiple nodes B.
- The RNC in UMTS provides functions equivalent to the Base Station Controller (BSC) functions in GSM/GPRS networks.
- The major difference is that RNCs have more intelligence built-in than their GSM/GPRS counterparts. For example, RNCs can autonomously manage handovers without involving MSCs and SGSNs.

iii. Node B:

- The Node B is responsible for air-interface processing and some radio-resource management functions.
- The Node B in UMTS networks provides functions equivalent to the base transceiver station (BTS) in GSM/GPRS networks. UMTS operates at higher frequencies than GSM/GPRS and therefore the signal coverage range is less.

Features of UMTS interfaces:

i. The UMTS interfaces can be categorized as follows:
a. Uu :

- This is the interface between the user equipment and the network. That is, it is the UMTS air interface.
- The equivalent interface in GSM/GPRS networks is the um interface.

b. The Iuis split functionally into two logical interfaces, Iupsconnecting the packet switched domain to the access network and the Iucsconnecting the circuit switched domain to the access network. The standards do not dictate that these are physically separate, but the user plane for each is different and the control plane may be different.
c. Iu –CS :

- This is the circuit-switched connection for carrying (typically) voice traffic and signaling between the UTRAN and the core voice network.
- The main signaling protocol used is Radio Access Network Application Part (RANAP).
- The equivalent interface in GSM/GPRS networks is the A-interface.

d. Iub :

- This is the interface used by an RNC to control multiple Node Bs.
- The main signaling protocol used is Node B Application Part (NBAP).
- The equivalent interface in GSM/GPRS networks is the A-bis interface.
- The Iubinterface is the main standardized and open, unlike the A-bis interface.

e. Iu –PS :

- This is the packet-switched connection for carrying (typically) data traffic and signaling between the UTRAN and the core data GPRS network.
- The main signaling protocol used is RANAP.
- The equivalent interface in GSM/GPRS networks is the Gb-interface.

f. Iur :

- The primary purpose of the Iur interface is to support inter-MSC mobility. When a mobile subscriber moves between areas served by different RNCs, the mobile subscriber’s data is now transferred to the new RNC via Iur.
- The original RNC is known as the serving RNC and the new RNC is known as the drift RNC.
- The main signaling protocol used is Radio Network Subsystem Application Part (RNSAP).
- There is no equivalent interface in GSM/GPRS networks.

Q.3. Explain the Range based and range free localization? If GPS is available why do we need it?

ANS. Localizations-

1. **Range-based localization** – It needs an extra hardware to accomplish ranging and then utilizes some algorithm to calculate the coordinates. Range-based algorithms calculate location information from the range-based measurement techniques like Received Signal Strength (RSS), Time of Arrival (ToA), Time Difference of Arrival (TDoA) and Angle of Arrival (AoA).
2. **Range-free localization** – It exploits the characteristics of the network connectivity. Range free algorithms calculate the location information from the connectivity information. Some physical measurement-based localization schemes are classified as Coarse-Grained and Fine-Grained.
   The localization is helpful to find coarse and grained coordinates with the help of GPS.

Q.4. Give Protocol Stack description of NETCONF YANG.

ANS. Protocol description of NETCONF YANG-

Network Configuration Protocol (NETCONF) is a network device management protocol that is similar to SNMP. NETCONF provides a framework for users to add, modify, or delete network device configurations, or query configurations, status, and statistics. Similar to SNMP that uses MIB files to model data, NETCONF uses Yet Another Next Generation (YANG) as a data modeling language to describe the interaction models between the NETCONF client and server.

One of the key network requirements of the cloud era is network automation, which includes quick, automatic, and on-demand service provisioning and automatic O&M. The traditional CLI mode and SNMP do not meet the requirements of cloud-based networks due to the following disadvantage:

- The traditional CLI mode is based on man-machine interfaces. The configuration is complex and varies with vendors. Therefore, the manual learning cost is high.
- The SNMP configuration efficiency is low, and the transaction mechanism is not supported. Therefore, SNMP is often used for monitoring.

Q.5. Compare Chef and Puppet. How they are applicable in IoT Management

ANS. **Puppet:** Puppet is a Configuration Management tool that is used for deploying, configuring and managing servers. It performs the following functions:

- Defining distinct configurations for each and every host, and continuously checking and confirming whether the required configuration is in place and is not altered (if altered Puppet will revert back to the required configuration) on the host.
- Dynamic scaling-up and scaling-down of machines.
- Providing control over all your configured machines, so a centralized (master-server or repo-based) change gets propagated to all, automatically.

Puppet uses a Master Slave architecture in which the Master and Slave communicate through a secure encrypted channel with the help of SSL.

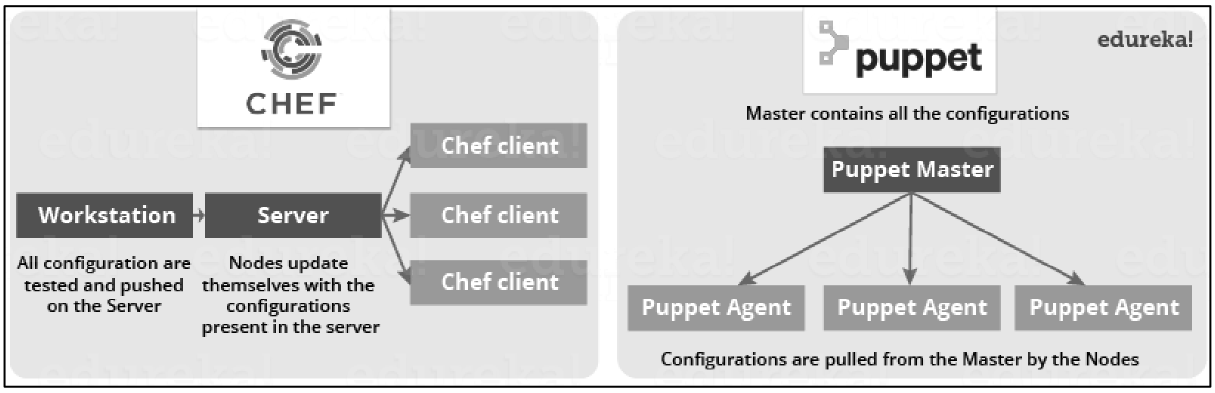
Chef: Chef is a tool used for Configuration Management and is closely competing with Puppet. Chef is an automation tool that provides a way to define infrastructure as code. Infrastructure as Code (IAC) simply means that managing infrastructure by writing code (Automating infrastructure) rather than using manual processes. It can also be termed as programmable infrastructure. Chef uses a pure-Ruby, domain-specific language (DSL) for writing system configurations.

Chef supports multiple platforms like AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows and Ubuntu. Additional client platforms include Arch Linux, Debian and Fedora. It can be integrated with cloud-based platforms such as Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure and Rackspace to automatically provision and configure new machines. Chef has an active, smart and fast-growing community support.

Because of Chef's maturity and flexibility, it is being used by giants like Mozilla, Expedia, Facebook, HP Public Cloud, Prezi, Xero, Ancestry.com, Rackspace, Get Satisfaction, IGN, Marshall University, Socrata, University of Minnesota, Wharton School of the University of Pennsylvania, Bonobos, Splunk, Citi, DueDil, Disney, and Cheezburger. Below are the types of automation done by Chef, irrespective of the size of infrastructure:

- Infrastructure configuration
- Application deployment
- Configurations are managed across your network

Like Puppet which has a Master-Slave architecture even Chef has a Client-Server architecture. But Chef has an extra component called Workstation. Refer the diagram below:


CHEF
Workstation
Server
Chef client
Chef client
Chef client
All configuration are tested and pushed on the Server
Nodes update themselves with the configurations present in the server
puppet
edureka!
Master contains all the configurations
Puppet Master
Puppet Agent
Puppet Agent
Puppet Agent
Configurations are pulled from the Master by the Nodes

Q.6. What is Pub-Sub model? How it is used in IoT deployment?

ANS. Publish/Subscribe is a software design pattern that describes the flow of messages between applications, devices, or services in terms of a publisher-to-subscriber relationship. Pub/Sub, as it is often called, works like this: a publisher (i.e. any source of data) pushes messages out to interested subscribers (i.e. receivers of data) via live-feed data streams known as channels (or topics). All subscribers to a specific publisher channel are immediately notified when new messages have been published on that channel, and the message data (or payload) is received together with the notification. As a general messaging pattern that can be implemented across various programming languages and platforms, Publish/Subscribe has a few core principles that are universally useful. These are:

- Publishers do not need to know anything about their subscribers, and subscribers only have to know the name of the topic channel they are subscribed to. Publishers simply define what data goes in which channel and transmit the channel data once only from one-to-many, so that it can be easily shared out amongst other apps for their own uses.
- Any publisher may also be a subscriber and data streams can be multiplexed, enabling the creation of interlinked systems that mesh together in an elegant, distributed, and internally-consistent manner.
- Since intercommunication is event-driven and based on notifications, there is no need for client-to-server polling techniques.
- Message data types can be anything from strings to complex objects representing text, sensor data, audio, video, or other digital content.

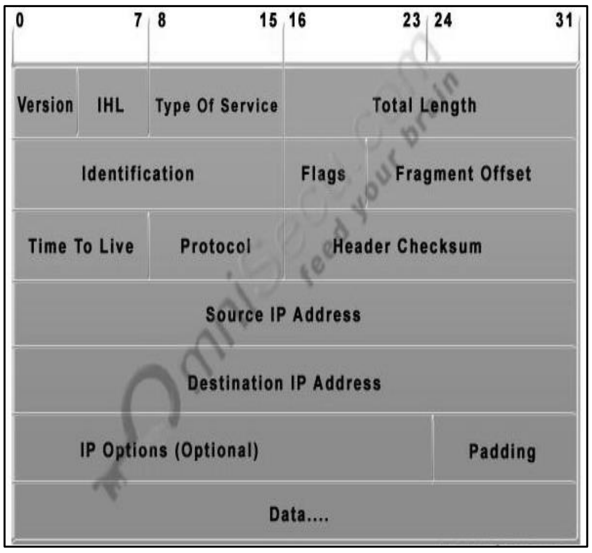
In ordinary usage, particularly in IoT, automation, network operations, or distributed cloud environments, there is often a need for an intermediary layer called a message broker that handles the distribution and filtering of messages, and which also provides low-latency message delivery over dedicated network infrastructure.

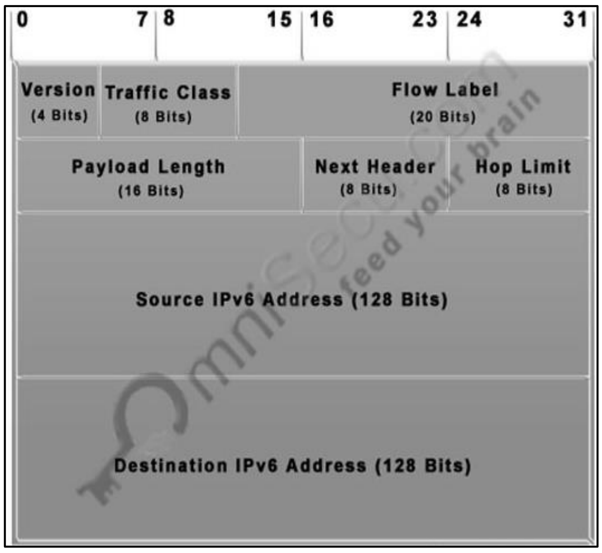
Q.7. Why HTTP is not suitable for IoT?

ANS. HTTP is for the Old-World Internet. The New World Internet (the IoT) is made up of unseen devices that require very little interaction. They consume very little power and frequently have poor network connectivity. HTTP is too heavy to be a good fit for these devices. An HTTP request requires a minimum of nine TCP packets, even more when you

consider packet loss from poor connectivity, and plain text headers can get very verbose. And even with all this overhead HTTP doesn't guarantee message delivery.
The HTTP overhead also adds to IoT operating expenses. Current costs of wireless connectivity are exorbitant. My cellphone provider charges $10/GB, and I've seen as much as $1/MB! This is to say nothing of the wireless spectrum shortage. Bandwidth conservation is especially important with enterprise customers that often have hundreds of thousands or millions of devices deployed.
Fortunately, there are suitable alternatives to HTTP for communicating with IoT devices.

Q.8. Explain the changes in IPv6 header as compared to IPv4
ANS.





IPv4 Datagram Header **IPv6 Datagram Header**

Following are the main differences and comparison between IPv4 header and IPv6 header.
• IPv6 header is much simpler than IPv4 header.
The size of IPv6 header is much bigger than that of IPv4 header, because of IPv6 address size. IPv4 addresses are 32bit binary numbers and IPv6 addresses are 128-bit binary numbers.
• In IPv4 header, the source and destination IPv4 addresses are 32-bit binary numbers. In IPv6 header, source and destination IPv6 addresses are 128-bit binary numbers.
• IPv4 header includes space for IPv4 options. In IPv6 header, we have a similar feature known as extension header. IPv4 datagram headers are normally 20-byte in length. But we can include IPv4 option values also along with an IPv4 header. In IPv6 header we do not have options, but have extension headers.
• The fields in the IPv4 header such as IHL (Internet Header Length), identification, flags are not present in IPv6 header.
• Time-to-Live (TTL), a field in IPv4 header, typically used for preventing routing loops, is renamed to its exact meaning, "Hop Limit".