



**Fakulteti i Shkencave të Natyrës**  
**Dega: Informatikë, Master Shkencor**  
**Lënda: Arkitektura Aplikimi Enterprise**

**Monitorimi i *Server Room* përmes përdorimit të  
Sensorëve dhe RaspberryPI**

## Tabela Përmbajtjes

Abrivime .....	4
Përshkrimi i sistemit.....	4
Aktorët kryesorë dhe përgjegjësitë .....	5
Madhësia e sistemit, metrikat rreth rrjetit dhe paisjet Hardware .....	6
Kërkesat Hardware.....	6
Kërkesat Software .....	6
Rrjeti .....	7
Protokollet .....	7
Lidhja mes RaspberryPI dhe sensorit DHT22 .....	7
Kërkesat.....	8
Shtresa 1 .....	8
Marrjen e të dhënave nga sensori.....	8
Hedhja e të dhënave në Cloud (AWS IoT ) .....	9
Shtresa 2 .....	10
Dërgimi i notifikimeve në formën e E-mail (G-mail) përmes shërbimit SNS.....	10
Hedhja e të dhënave në DynamoDB .....	12
Shtresa 3 .....	12
Vlerësimi i efektivitetit të sistemit .....	13
Rreziqet e mundshme të sistemit.....	14
Përmirësime potenciale të arkitekturës.....	15
Përmirësimi i konfigurimit të load-balancer-it. ....	15
Tomcat Clustering .....	15
Nginx load-balancer .....	16
Konfigurimi i Nginx si load balancer .....	16
Një problem që duhet zgjidhur .....	17
Session Manager në Tomcat .....	18
Analogji me sisteme të tjera .....	19
Referenca .....	23



### Abrivime

DHT11	Sensori i cili mat temperaturën dhe lagështinë relative
RaspberryPI	Single-Board computer përmes të cilit mund të programojmë mbi vlerat e marra nga sensori
AWS	Amazon Web Services
SNS	Simple Notification Service (AWS)
IoT	Internet of Things
TEMP	Temperatura
RH	Lagështia relative
MQTT	Message Queuing Telemetry Transport

### Përshkrimi i sistemit

Sistemi “Server Room Monitoring” investigon në kohë reale parametrat atmosferik të një *Server Room*, përkatësisht temperaturën dhe lagështinë relative të marra nga sensori *DHT22* përmes paisjes *RaspberryPI*.

Qëllimi kryesor i tij është rritja e vlerës për një biznesi që zotëron një dhomë serverash duke përmirësuar sigurinë dhe disponueshmërinë të cilat janë dy tipare thelbësore në një *enterprise system*.

Sistemi ka dy software të implementuar, i pari mbledh të dhënat nga sensori dhe bën dërgimin e tyre në cloud, ndërkohë software i dytë i grumbullon këto të dhëna duke i shfaqur në një aplikacion web në formën e grafikëve. Për të bërë të mundur një sistem të tillë ne investiguam mënyra të ndryshme se si këto aplikacione mund të komunikojnë me njëri-tjetrin në mënyrë që të jenë të sinkronizuar, për këtë arsye sistemi përdor AWS për ruajtjen e të dhënave (në databazë) të marra nga sensori, nga e cila më pas bëjmë query për të marrë vlerat e temperaturës dhe lagështisë. Komunikimi me shërbimet e AWS kryhet përmes përdorimit të protokollit MQTT i cili është më lightweight për të dhëna të tilla, që gjenerohen në intervale të shpeshta kohore. Sistemi ofron dhe mundësinë për të marrë historikun e të dhënave nga data fillestare e hedhjes së tyre në databazë.

Në vazhdim dokumentacioni paraqet teknologjitë e përdorura, lidhjen mes tyre, qëllimet si dhe përmirësime të mëvonshme.

*Sistemi mund të implementohet dhe për monitorimin e aktiviteteve të tjera (generator room, backup room, UPS battery room etj), ku parametrat që do të monitorohen janë të njëjtë si më lartë.*

## Aktorët kryesorë dhe përgjegjësitë

Më poshtë po japim një përshkrim për sa i përket aktorëve që marrin pjesë në sistem:

### 1. Server Room

- Merr vlerat e temperaturës dhe lagështisë nga sensori DHT22 duke përdorur librarinë e Adafruit
- Filtron të dhënat për të kuptuar nëse janë alarmante apo jo. E dhënë ruhet në formë stringu në variablin **status**
- Dërgon të dhënat në AWS IoT për një interval kohor prej 20 sekondash duke përdorur protokollin MQTT dhe Paho-MQTT Client

### 2. AWS

- Filtrohen të dhënat në AWS IoT broker , nëse vlera e variablit **status** është e ndryshme nga **“normal”** do të dërgohet një e-mail drejt personave të autorizuar përmes shërbimit AWS SNS.
- Dërgohet e-mail drejt personave të autorizuar edhe nëse RaspberryPI nuk dërgon më të dhëna (arsyet mund të jenë të ndryshme: shkëputje interneti, raspberry ka probleme dhe kodi nuk ekzekutohet suksesshëm, kodi nuk ekzekutohet sukseshëm për arsye të tjera etj.)
- Cdo e dhënë e marrë nga RaspberryPI do të ruhet në DynamoDB database

### 3. Java Web Application

#### • Screens

- Screen 1 : Login JSP, vetëm përdoruesit e autorizuar mund të logohen.
- Screen 2: Pasi përdoruesi autentikohet do të drejtohet tek dashboard-i. Në dashboard bëhet query DynamoDB, grafiku i merr të dhënat në mënyrë asinkrone (Javascript SDK), maksimalisht merren 10 të dhënat më të fundit nga databaza.
- Screen 3: Përdoruesi tashmë i autentikuar mund të ndryshojë kredencialet e tij.
- Përdoruesi mund të bëjë shkarkojë në format CSV historikun e të dhënave nga DynamoDB (JAVA SDK)

#### • Servers

- Tomcat Servlet Container (Nr 1\_WebServer) bëhet deploy aplikacioni
- Tomcat Servlet Container (Nr 2\_WebServer) bëhet deploy aplikacioni
- NginX Load Balancer drejton trafikun drejt dy serverave Tomcat
- Si Databazë është përdorur MySQL. Kjo databazë lidhet me të dy serverat Tomcat.

## Madhësia e sistemit, metrikat rreth rrjetit dhe paisjet Hardware

Sistemi për monitorimin e “Server Room”, sic u tha dhe më lartë , ka të detyrueshme përdorimin e një sensori dhe RaspberryPI. Më poshtë po japim një specifikim për sa i përket këtyre paisjeve, si funksionojnë ato dhe si është kryer lidhja mes tyre.

### Kërkesat Hardware

- Sensor DHT22 AM2302 , sensori dixhital
- Specifikimet teknike janë si mëposhtë:

Ushqimi me rrymë	3.3-6V DC (Direct Current)
Sinjali Output-it	Sinjal dixhital përmes një bus-i
Intervali i vlerave	0-100% RH; TEMP 40-80 °C
Saktësia	+2%RH(Max +5%RH); TEMP <+-0.5 °C
Intervali i ndjeshmërisë së të dhënave	Mesatarisht: 2 sekonda
Dimensionet	Gjerësi: 15.4 mm; Gjatësi: 25.3
Numri i pineve	4
<b>Pin</b>	<b>Funksioni</b>
1	GND – Ground (pika referimit ; pika me potencialin më të ulët)
2	NULL
3	Sinjali i të dhënave (DATA)
4	VDD – Ushqyesi i rrymës (Drain supply)

- RaspberryPI 3 Model B
- Tre tela përcues për të transferuar sinjalet elektrike mes sensorit dhe RaspberryPI
- Breadboard (opsional) për të lidhur sensorin me RaspberryPI në rastin kur telat përcues kanë socket nga njëra anë, pine nga tjetra
- MicroSD Card 8GB për të ngarkuar sistemin e operimit Rasbian (versioni Linux Debian për RaspberryPI)
- Kabëll USB Micro për të ushqyer RaspberryPI me elektricitet
- Dy Tomcat (Servlet Containers) tek të cilët do të bëhet deploy aplikacioni web
- NginX load balancer që do të bëjë shërndarjen e trafikut tek dy instancat Tomcat
- MySQL Database ku do të ruhen rekordet e përdoruesëve të aplikacionit

### Kërkesat Software

- Libraria Adafruit për sensorin DHT22
- MQTT Client për të bërë lidhjen me Cloud
- JavaEE për ndërtimin e aplikacionit ëeb
- Javascript për shfaqjen e grafikëve në kohë reale

- HTML, CSS për ndërfaqen e përdoruesit

## Rrjeti

- Raspberry do të lidhet përmes Wi-Fi USB me rrjetin Wireless, atij i vendoset një IP statike brenda range-it të rrjetit ku do të vihet në punë ; Ose Raspberry lidhet me Ethernet me router-in dhe përsëri mund t'i vendoset një IP statike si më lartë.
- Gjatë lidhjes së RaspberryPI me AWS, paketat me të dhëna do të lëvizin në Internet .
- Lidhja me IoT bëhet përmes protokollit IoT MQTT(S) pasi përdoret SSL/TLS për të krijuar një kanal komunikimi të sigurt mes *Cloud Provider* dhe *"Server Room"*
- Lidhja nga AWS për në aplikacionin web arrihet përmes protokollit HTTP përdoren API e AWS për JAVA dhe API e AWS për Javascript

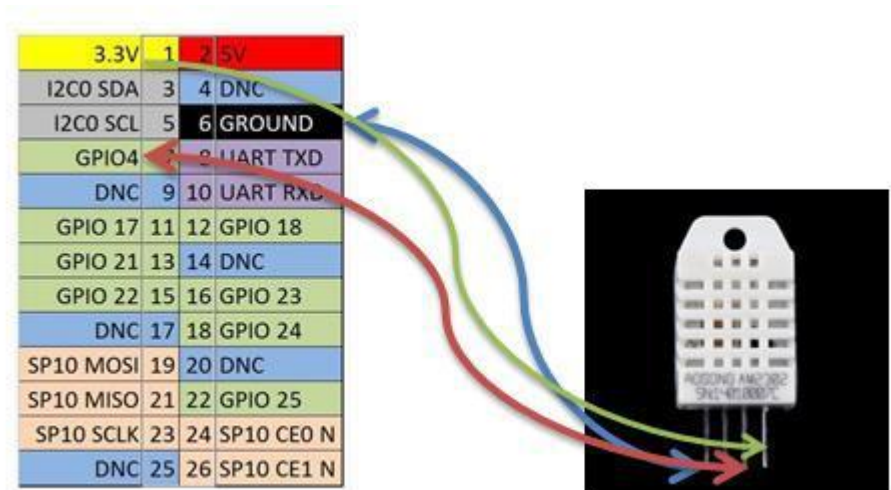
## Protokollet

**MQTT(S)** -> Në rastin e aplikacionit tonë, ku të dhënat do të dërgohen në AWS në një interval prej cdo 20 sekondash është e nevojshme të përdoret një protokoll i cili harxhon sa më pak bandëidh dhe në të njëjtën kohë ofron performancë të lartë për rastin në fjalë. MQTT është protokoll shumë *lightweight* ai përdor teknikën e mesazheve(topics) publish/subscribe. Në mënyrë që ky protokoll të funksionojë duhet që në mes të qëndrojë një broker. Në rastin tonë ky broker është AWS IoT.

**HTTP** -> Protokollit HTTP është përdorur për të komunikuar nga aplikacioni web në JAVA me DynamoDB për marrjen e të dhënave

## Lidhja mes RaspberryPI dhe sensorit DHT22

Në mënyrë që të merrren të dhënat nga sensori duhet që të kryhet lidhja mes pineve të Raspberry me ato të sensorit. Raspberry ka 26 pine gjithësej ku 8 prej tyre shërbejnë si GPIO (General Purpose Input Output) nga ku mund të merren të dhënat e sensorit. Ky pin do të lidhet me pinin numër 2 të sensorit pasi ai është pini nga i cili del voltazhi mbi të dhënat e temperaturës dhe lagështisë. Një pamje më e qartë jepet më poshtë:



- Pini 1 (**ekzekutohet** me një voltazh Vcc 3.3 V) i raspberry do të lidhet me pinin 4 të sensorit që ishte pini për të marrë elektricitet.
- GPIO 4 (pini numer 7) i raspberry do të lidhet me pinin nga ku dalin të dhënat tek sensor, që është pini 3
- GROUND (pini numer 6) i raspberry do të lidhet me pinin 1 të sensorit që shërben si pikë referimi.

### Kërkesat

Në këtë seksion do të flitet mbi burimet që ne kemi përdorur në këtë projekt si dhe specifikimet funksionale dhe jo funksionale të sistemit.

### Shtresa 1

Në dhomën e serverit do të vendoset paisja RaspberryPI e lidhur me sensorin DHT22. Softwarënë pjesën e Raspberryryt duhet të sigurojë dy gjëra:

### Marrjen e të dhënave nga sensor

Për këtë qëllim na duhet një librari e cila ka akses në pinet dixhitale të Raspberry, ne zgjodhëm të përdorim Adafruit pasi ofron lehtësi në zhvillimin e kodit si dhe është produkt që ka më shumë kohë në treg se sa libraritë e tjera, gjithësesi në mënyrë alternative mund të përdoren dhe librari të tjera si psh: PI4J, WiringPI, PIGPIO etj. Mënyra se si kapen të dhënat është duke përdorur funksionin:

- `read_retry(sensor_type, raspberry_gpio_pin)`



Parametri i parë specifikon llojin e sensorit që në rastin tonë është DHT22 AM2302, si rrjedhojë ky atribut do të marrë *vlerën* -> 2302, nëse do të ishte DHT11 do të merrte vlerën 11. Parametri i dytë është pini i input-output nga i cili futen të dhënat nga përcuesi që është i lidhur me sensorin, ky është pini 7, i njohur ndryshe si GPIO 4, prandaj ky atribut merr *vlerën* -> 4.

- Të dhënat procesohen sipas vlerave të lejuara në një *Server Room* (referencë: Dorrëzimi 1). Kodi ekzekutohet gjatë gjithë kohës pra kemi një cikël *while()*, pasi monitorimi do të jetë i vazhdueshëm. Kodi ekzekutohet në intervale prej 20 sekondash, pra ne kapim vlerën e temperaturës dhe lagështisë së mjedisit cdo 20 sekonda, prandaj në fund të ciklit *while()* thërrasim funksionin ***sleep(20)*** (20 -> *vlera e sekondave*).
- Temperatura dhe lagështia kanë tipin Number dhe kështu ruhen dhe në databazën DynamoDB, precizion: **1 shifër pas presjes**.

### Hedhja e të dhënave në Cloud (AWS IoT)

Duke qenëse ne zgjodhëm shërbimin IoT të Amazon për dërgimin e të dhënave, atëherë duhet që fillimisht të konfigurojmë Raspberry-në në Amazon. Fillimisht ne krijuam një account në Amazon Free Tier e cila na jep akses falas në disa nga shërbimet më përdorura të Amazon. Më pas shkojmë në seksionin AWS IoT dhe konfigurojmë Raspberry-në. Më poshtë japim disa sqarime mbi komponente që janë pjesë e AWS IoT dhe janë përdorur nga ne.

Komponente	Shpjegim
Message broker	Ofron një mekanizëm të sigurtë për të bërë publish në paisjet e konfiguruar. Për këtë qëllim mund të përdoret protokollin MQTT duke përdorur Web Socket për të arritur një komunikim të sigurt duke shtuar protokollin SSL/TLS. Në mënyrë alternative mund të përdoret HTTP/REST.
Security and Identity service	Meqë ne përdorim MQTT mbi SSL/TLS, atëherë autentikimi nuk do të bëhet me <i>username, password</i> , por me certifikata. Kështu, komunikimi mes Raspberry në <i>Server Room</i> dhe AWS IoT është i mbrojtur duke përdorur <b>certifikata X.509</b> si dhe <b>private/public key authentication</b> . Pasi krijohet certifikata ajo <b>aktivizohet</b> dhe bëhet <b>doënlod</b> në RaspberryPI nga ana jonë. Në momentin që ne duam të komunikojmë nga <i>Server Room</i> për të bërë publish të dhënat, ajo cka përdorim si mjet për të verifikuar identitetin është certifikata. Pasi certifikata dhe çifti i celësave është bërë doënlod në Raspberry, ato fshihen në mënyrë automatike nga Amazon.

Rules Engine	<p>Një ndër arsytet se pse ne kemi përdorur AWS si Cloud provider është dhe Rules engine. Ky mekanizëm ofron:</p> <ol style="list-style-type: none"> <li>1. Procesimin e mesazheve që bëhen <i>publish</i> në Broker-in e AWS. Përdoret sintaksa SQL për të selektuar të dhënat që dëshirojmë të procesojmë.</li> <li>2. Komunikimin e disa shërbimeve të AWS me njëri-tjetrin. PSH: Në momentin që <i>Select</i>-i kthen vlerë atëherë do të trigerohet një funksionalitet i caktuar, në rastin tone trigerohet <i>Insert</i>-i në DynamoDB dhe dërgimi i e-mail përmes SNS.</li> </ol>
--------------	--

Për të krijuar një kanal të sigurt komunikimi me AWS IoT ndiqen këto hapa:

a. Krijohet një instancë e tipit `Mqtt_Client` -> `mqttc = mqtt.Client (client_id="raspberry-pi")` . Parametri `client_id` duhet të përputhet me emrin e raspberry-t tek shërbimi AWS IoT.

b. Konfigurojmë enkriptimin e rrjetit dhe opsionet e autentikimit. Në mënyrë që Mqtt të ofrojë komunikim të sigurtë duhet të bëjmë enable SSL/TLS për të enkriptuar të dhënat. Në këtë lloj komunikim raspberry në Amazon është e mbrojtur përmes certifikatave.

Certifikata dixhitale X.509 përdor infrastrukturën e celësive publik/privat. Celësi publik nënkupton një identitet me një certifikatë. Certifikatat X.509 lëshohen nga certification authority(CA) dhe kanë një datë expiry, në rastin tonë CA është AWS IoT dhe certifikata nuk skadon kurrë. Benefitet e këtij lloj autentikimi është se celësat janë shumë më të vështirë për t'u hackuar se sa *username-password* apo *token*. Certifikata gjenerohet vetëm njëherë dhe ruhet në paisjen nga ku do të iniciohet komunikimi pra nuk kemi pse të fusim gjithmonë *username* dhe *password*. Funksioni që përdorim për këtë qëllim (pjesë e MQTT Client) është si mëposhtë:

```
mqttc.tls_set("/home/pi/Desktop/amazon_certs/root-ca.crt",
certfile="/home/pi/Desktop/amazon_certs/d0d2de1176-certificate.pem.crt",
keyfile="/home/pi/Desktop/amazon_certs/d0d2de1176-private.pem.key",
tls_version=ssl.PROTOCOL_TLSv1_2, ciphers=None)
```

## Shtresa 2

Tashmë që të dhënat ndodhen në AWS IoT ne fillojmë t'i procesojmë të dhënat duke përdorur Rules Engine. Shërbimet që kemi përdorur dhe kërkesat për secilën janë si mëposhtë:

### Dërgimi i notifikimeve në formën e E-mail (G-mail) përmes shërbimit SNS

**Rule Nr. 1** -> Dërgohet E-mail kur të dhënat e marra nga sensori janë jashtë vlerave normale:

a. Kur do të ekzekutohet ky Rule (mbi cfarë eventit)?

- Në rastin tonë Rule për të dërguar e-mail drejt personave të paracaktuar do të ekzekutohet nëse variabli **status** që vjen së bashku me **payload** e ka vlerën të ndryshme nga

“Normal”. Në këtë rast ne bëjmë select këtë atribut nga i gjithë mesazhi i publikuar në topic (screenshot mëposhtë)

**Rule query statement** Edit

The source of the messages you want to process with this rule.

```
SELECT state FROM 'topic/sensordata' WHERE state <> 'Normal'
```


Using SQL version 2015-10-08

b. *Cfarë do të ndodhë në momentin kur event i ndodh?*

- Në këtë rast do të trigërohet action-i për të dërguar e-mail tek të gjithë endpointet e specifikuar me këtë action të SNS.

#### Actions

Actions are what happens when a rule is triggered. [Learn more](#)

 **Send a message as an SNS push notification** Edit Remove

**Message format:** RAW

**SNS target:** arn:aws:sns:us-west-2:297431228336:sendMail

SNS Target është si mëposhtë:

Topic details: sendMail

**Publish to topic** **Other topic actions**

**Topic ARN** arn:aws:sns:us-west-2:297431228336:sendMail

**Topic owner** 297431228336

**Region** us-west-2

**Display name**

#### Subscriptions

**Create subscription** **Request confirmations** **Confirm subscription** **Other subscription actions**

Filter

Subscription ID	Protocol	Endpoint	Subscriber
arn:aws:sns:us-west-2:297431228336:sendMail:2d34f3ce-23d8-4d8a-9573-031b53b4562e	email	kejsistruga6@gmail.c...	297431228336
arn:aws:sns:us-west-2:297431228336:sendMail:d62b92c4-fbb9-4292-b41d-922c921b5151	email	kejsi.struga@fshnstu...	297431228336
PendingConfirmation	email	kejsistuga@yahoo.co...	297431228336
PendingConfirmation	email	xhensila.poda@fshn...	297431228336

**Rule Nr. 2 -> Dërgohet E-mail kur RaspberryPI nuk dërgon të dhëna**

a. *Kur do të ekzekutohet ky Rule (mbi cfarë event i)?*

- Nëse kapet event i *Disconnected* kjo do të thotë se Raspberry nuk po dërgon më të dhëna. E-mail i shkon endpointit vetëm nëse endpoint-i ka bërë subscribe në të kundërt jo. E-mail do t'i shkojë klientit pas 1-2 minutash pasi event i ka ndodhur

**Rule query statement** Edit

The source of the messages you want to process with this rule.

```
SELECT eventType FROM '$aws/events/presence/disconnected/raspberry-pi'
```

Using SQL version 2015-10-08

b. *Cfarë do të ndodhë në momentin kur event i ndodh ?*

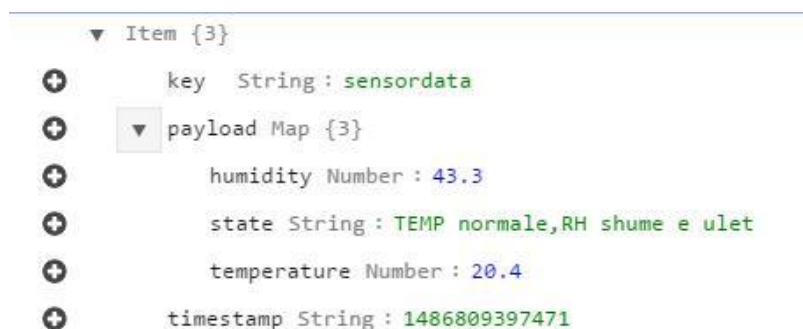
E njëjta gjë si në Rule Nr. 1.

## Hedhja e të dhënave në DynamoDB

Sic u tha dhe në hyrje, të dhënat do të ruhen në databazën NoSQL DynamoDB. Përsëri përdoret Rules Engine për të bërë Select të gjitha të dhënat që vinë nga RaspberryPI (*Payload*). Arkitektura e një databaze NoSQL është si mëposhtë:

1. Tabela - {item\_1, item\_2, .. , item\_n}
2. Items – {attribute\_1, attribute\_2, ... attribute\_n} (cdo item është unik !)
3. Attributes – {value} (skalar => kanë vtm nje vlere )
4. Të dhënat ruhen në format JSON
5. Për të indentifikuar në menyre unike nje rekord tebele perdoren **Primary Key** (në rastin tonë primary key është composite key) që përbëhet nga:
  - a. *Partition Key* –Njihet dhe si **HashAttribute**, mund të kemi disa Partition Key, dhe nuk është unik , psh. në tabelën tonë të gjitha rekordet e kanë partition key të njëjtë, sepse të gjitha janë sensor data, pra i përkasin një domain.
  - b. *Sort Key* – është unik për secilin nga rekordet, në rastin tonë si sort key është timestamp.

Një rekord do dukej si mëposhtë, ku key është Partition Key, payload është objekti që përmban të dhënat që vinë nga sensori dhe timestamp është Sort Key.



## Shtresa 3

Shtresa e tretë ka për qëllim ta bëjë sistemin më të plotë, deri diku kjo shtresë mund të konsiderohet si një sistem më vetë pasi në të ndërthuren disa teknologji, gjithësesi qëllimi i saj është të ofrojë një Dashboard për persona të autorizuar të kompanisë si psh. punonjës nga departamenti IT të cilëve mund t'u duhet të monitorojnë dhomën e serverit në kohë reale, si dhe për të patur akses tek të dhënat. Në faqen kryesore shfaqen dy grafikë, i pari i korrespondon RH dhe i dyti TEMP. Për dizajnin e grafikëve kemi përdorur librarinë *Chart.js* e cila i paraqet Query që bëjmë për të marrë të dhënat është si mëposhtë:

```

var params = {
  TableName: 'serverRoomSensor',
  KeyConditionExpression: '#key = :key',
  ExpressionAttributeNames: {
    "#key": "key"
  },

```

```
ExpressionAttributeValues: {  
  ":key": { "S" : "sensordata"}  
},  
Limit: 10,  
ScanIndexForward: false  
};  
  
dynamodb.query(params, function (err, data)  
{ if (err) {  
  console.log(err);  
  return null;  
} else {  
  // display graphics ..  
}
```

Ndërkohë që grafikët shfaqen në Client Side, user-at e autentifikuar do të kenë mundësi edhe të bëjnë download historikun e të dhënave në DynamoDB , për këtë do të përdorim servletin i cili do të bëjë query DynamoDB duke përdorur JavaSDK të DynamoDB e cila është një bashkësi *.jar files* që mundëson krijimin e klientëve DynamoDB nga programet në Java. Të dhënat do të bëhen download në format CSV.

### Vlerësimi i efektivitetit të sistemit

Objektivi parësor është që pwrmbushja e kwrkesave të biznesit. Më poshtë po përmendim disa karakteristika të domosdoshme në arkitekturën e sistemeve enterprise, dhe se si ato shfaqen në aplikacionin tonë.

#### ➤ Performanca

Cdo vizitues i paregjistruar mund të shikojë vetëm pjesën home të faqes. Po në këtë faqe ndodhet një buton login ku personat e autorizuar mund të logohen në sistem.

Te gjitha funksionet e programit do të jenë të ndara në menu dhe me një klikim në elementët e menisë kalohet në ndërfaqen tjetër që kryen funksionin.

#### ➤ Disponueshmëria (Availability)

Sistemi duhet të jetë 24 orë online. Kjo gjë është e arritshme, sepse serverat në të cilët hostohet sistemi ynë garantojnë siguri që serverat janë 99% uptime.

#### ➤ Siguria

Hedhja e të dhënave në Amazon përmes krijimit të një lidhjeje të enkriptuar duke përdorur mënyrën e autentikimit me çelës publik/privat, ofron siguri më të lartë për sistemin gjatë kohës kur paketat udhëtojnë në rrjet.



#### Portabiliteti

Aplikacioni duhet t'i ofroje përdoruesve mundësinë, për të realizuar të gjitha funksionet dhe objektivat e lartpërmendura, por gjithmonë ka vend për ndryshime dhe përmirësime. Programi do të strukturohet në mënyrë të tillë që ndryshimet të jenë të lehta duke mos kërkuar shumë kohë dhe buxhet.



#### Besueshmëria (Reliability)

Nëse ndodh një gabim në program, atëherë mjafton që kompania përkatëse të na njoftojë dhe rregullimi bëhet direkt pa qene nevoja te shkohet tek kompania perkatese.

- Sistemi ynë duhet të jetë i gatshëm që të na ofrojë shërbim sa më të shpejtë për veprimet që bën përdoruesi. Zhvilluesit duhet të marrin parasysh për vonesat që mund të lindin për faktin që mund të ketë ngarkesë sistemi, por kjo është marrë parasysh kur është bërë modelimi i sistemit.
- Sistemi është online gjatë gjithë kohës, dhe si i tillë duhet të jetë gjatë gjithë kohës aktiv dhe i disponueshëm për përdoruesin. Sistemi është i sigurt, pra të gjithë përdoruesit e sistemit kanë një account për tu log-uar. Kjo bëhet për të mbrojtur të dhënat dhe nuk është e nevojshme që përdoruesit të marrin informacion më shumë sesa u takon. Më parë se sistemi të vihet në punë duhet analizimi i mirë i softit të tij (i kodit) në mënyrë që :
  - Mos të kemi vonesa, të jetë i shpejtë në ekzekutim.
  - Kodi të jetë i saktë.
  - Kodi të jetë i modifikueshëm.
  - Të dhënat që marrim nga sensori të jenë të sakta.

### Rreziqet e mundshme të sistemit

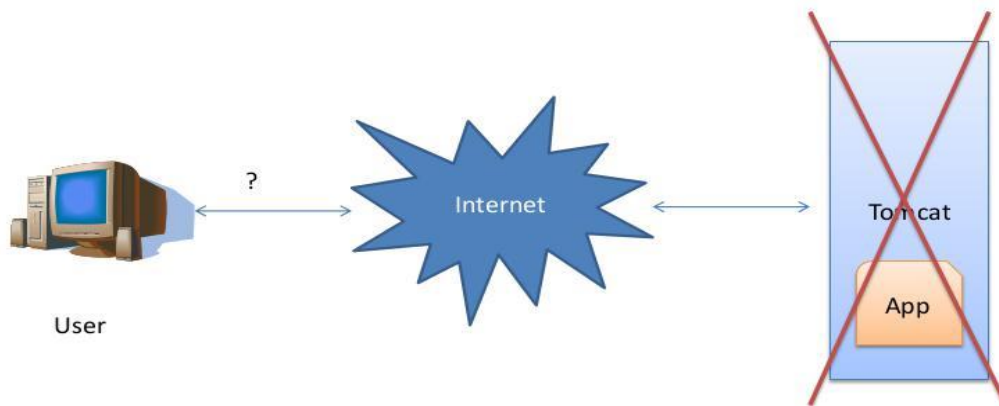
- Raspberry Pi duhet të jete i lidhur me internet ne menyre qe te dhenat te dergohen ne DynamoDB.
- Raspberry Pi duhet të lidhet me një burim rryme jo më të madh se 3.3V dhe nuk duhet të ketë asnjë luhajtje.
- Mosfunksionimi i duhur i sensorit sjell dhe anomali në vlerat e lejuara të temperaturës dhe lagështisë.
- Gjenerimi i të dhënave nga sensori bëhet çdo 20 sek dhe gjatë një dite të vetme databaza ne DynamoDB do të ketë afërsisht 4300 vlera. Duhet gjetur një mënyrë për pastrimin e tabelave në mënyrë periodike (ose jo, pasi DynamoDB bëhet scale në mënyrë horizontale dhe kjo sjell përfitime pasi nuk kërkohet shtesë në pajisje hardware si RAM apo shtesë në servera).
- Databaza e përdoruesve në MySQL është *single point of failure*.

## Përmirësime potenciale të arkitekturës.

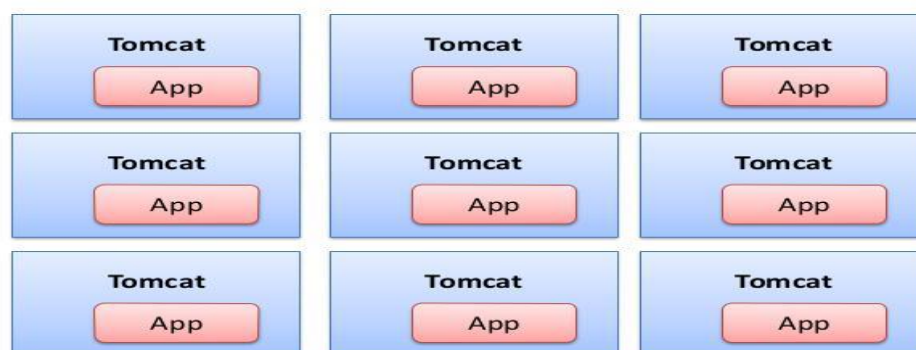
### Përmirësimi i konfigurimit të load-balancer-it.

#### Tomcat Clustering

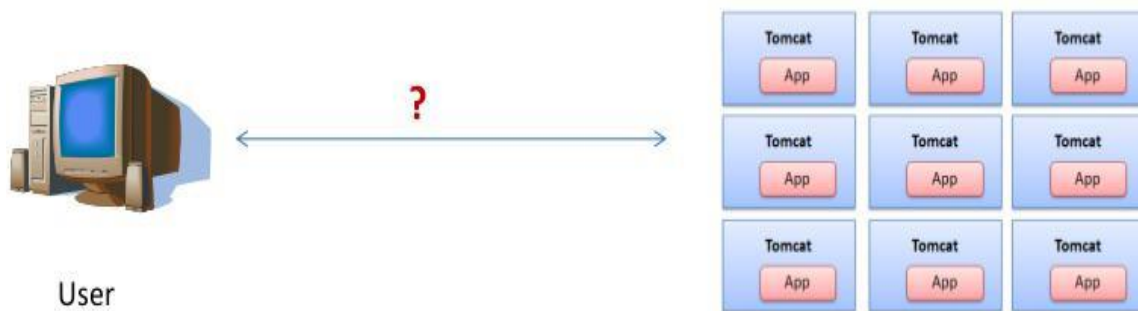
Nëse aplikacioni ynë do të ekzekutohej në një server të vetëm, në rastin e një dështimi të serverit përdoruesve nuk do t'u jepej mundësia e aksesimit të web application-it.



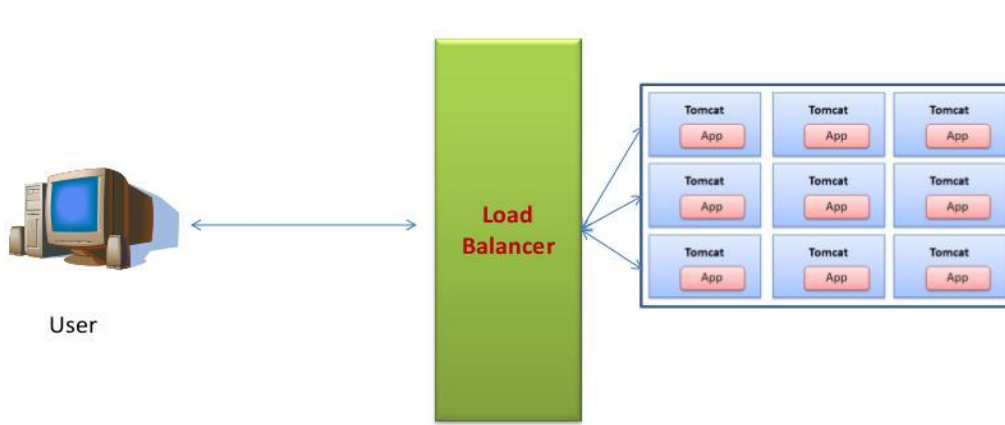
Për të zgjidhur këtë problem vendosëm të krijojmë një grup ëbserverash Tomcat (Tomcat Cluster) në të njëjtën makinë. Cdo Tomcat përmban webapplication-in brenda file-ve të tij kështu që cdo Tomcat mund ti përgjigjet kërkesave të klientit. Në një situatë të tillë, në dukje nëse një nga serverat bie, serverat e mbetur përballojnë kërkesat.



Por me këtë zgjidhje një problem akoma më i madh ngrihet. Nga perspektiva e klientit, cili Tomcat do të përdoret për të përballuar kërkesat dhe si do ta "organizojnë" punën këta ëbserver-a midis tyre.



Zgjidhja bëhet e mundur duke shtuar një server përpara cluster-it, i cili sillet si load-balancer, pranon kërkesa nga klienti dhe i ridrejton këto kërkesa sipas një algoritmi të përcaktuar drejt ëbserver-ave. Ne kemi përdorur NGINX si load-balancer.



## Nginx load-balancer

Nginx është një alternativë open source. Nginx si load balancer është një mundësi shumë e mirë për aplikacionin tone sepse është shumë performant dhe me shumë pak konfigurime mund të arrish rezultate shumë të mira.

### Konfigurimi i Nginx si load balancer

Konfigurimet e Nginx bëhen në filin nginx.conf. Për ta arritur këtë duhet të kryejmë dy hapa.

Të shtojmë ne file-in nginx.conf nje bllok upstream i cili është i ndërtuar si më poshtë, dhe përmban adresat e tre ëbservera Tomcat që bejnë pjesë ne cluster.

Këta tre ëeb server-a janë përgjegjës përprocesimin e kërkesave të klientëve :

```
upstream tomcatcluster{
```

```
server 127.0.0.1:8081;
```

```
server 127.0.0.1:8082;
```

```
server 127.0.0.1:8083; }
```



- a. Së dyti duhet që të gjitha kërkesat ti ridrejtojmë drejt këtij blloku upstream.

```
location / {  
proxy_pass http://tomcatcluster/ ; }
```

### Një problem që duhet zgjidhur

Aplikacioni ynë përmban sesione me anë të së cilave ne e mbajmë përdoruesin të loguar, nëse provojmë ta ekzekutojmë aplikacionin në këtë gjendje do të hasim problemet e mëposhtme:

1. User-i akseson një faqe në të cilën përdoren sesionet.
2. Load balanceri "kap" kërkesën dhe përdor algoritmin Round Robin për ta ridrejtuar tek njëri prej faqe server-ave.
3. Supozojmë se kërkesa drejtohet tek Tomcat , i cili krijon objektin e tipit Session për këtë përdorues.
4. I kthehet përgjigjia klientit, më pas klienti bën një kërkesë tjetër e cila me anë të load balancer-it ridrejtohet tek Tomcat 2, por ky i fundit nuk e ka sessionin e përdoruesit dhe e krijon nga e para.
5. Duke ndjekur këtë logjikë cdo Tomcat do të krijojë hap pas hapi sesione të reja për përdoruesin.

### Session affinity/Sticky Session

Session affinity e zgjidh në këtë mënyrë situatën: Nëse kërkesa e parë e një përdoruesi shkon drejt Tomcat 1, dhe ky i fundit krijon sessionin për përdoruesin atëherë të gjitha kërkesat në vazhdim të këtij përdoruesi do të shkojnë vetëm tek Tomcat 1. Kjo arrihet duke përdorur komandën **jvmRoute** në file-et server.xml të cdo tomcati pjesë e cluster-it. Në këtë mënyrë cdo tomcat gjeneron vlerën e session\_id si përbërje e dy vlerave :

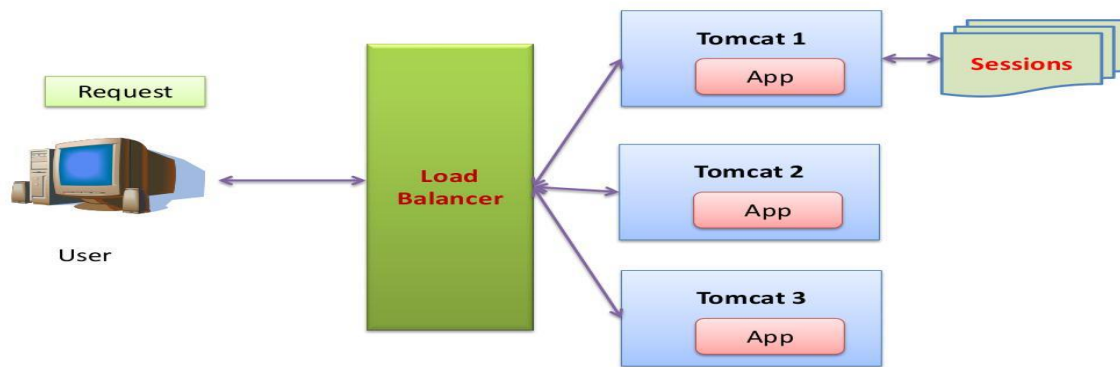
**<Random Value like before> . <jvmRoute value> .**

Kështu load balancer-i e kupton se cilit prej Tomcat-eve duhet t'ia dergojë kërkesën pasardhëse.

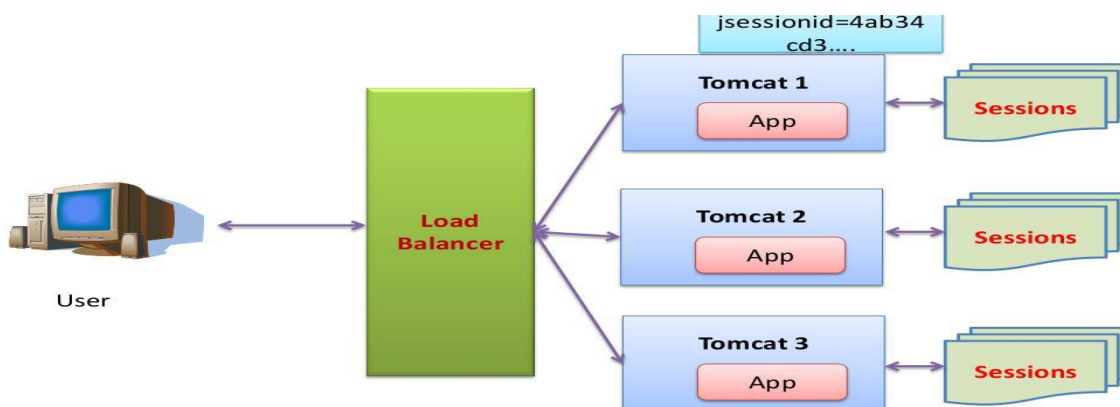
**Shembull:**

**Cookie:JSESSIONID=40025608F7B50E42DFA2785329079227.Tomcat 1**

Por kjo zgjidhje sërish nuk i kënaq kërkesat tona. Supozojmë se Tomcat 1 po proceson të gjitha kërkesat e user\_1, problemi lind nëse Tomcat 1 bie, kërkesat do të ridrejtohen tek Tomcat 2 por aty nuk do të ketë sesione të ruajtura për këtë përdorues.



### Zgjidhja: Session Replication



### Session Manager në Tomcat

Session Manager ne Tomcat është përgjegjës për krijimin dhe menaxhimin e sesioneve.

Kemi tre tipe te Session Manager:

- Standard Manager
- Persistent Manager
- Delta Manager
- Backup Manager

Në aplikacionin tone do të përdorim Delta Manager, sepse na jep mundësinë që të replikojmë sesionin e krijuar në një instancë tek cdo instancë tjetër. Shtojmë tag-un Cluster në elemintin Engine te file-eve server.xml si më poshtë:

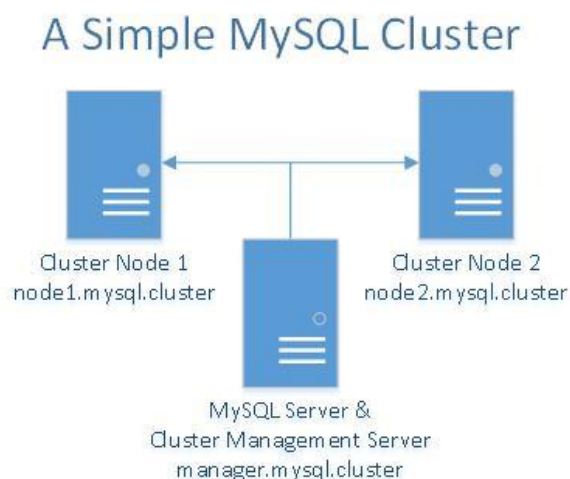
```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

Duke përdorur këtë konfigurim në lejojmë replikimin e cdo sesioni tek cdo instance pjesë e Cluster-it. Këtë mënyrë zgjidhjeje e kemi gjetur si më efikase për problemin që trajtojmë por ekzistojnë disa disavantazhe në lidhje me këtë zgjidhje.

1. Kjo zgjidhje funksionon më së miri për një cluster me përmasa të vogla.
2. Sesionet do të replikohen në të gjitha nyjet pjesmarrëse në cluster edhe tek ato që nuk e përmbajnë aplikacionin.

## 2. Përmirësimi i sigurisë dhe high-availability i databazës MySQL ku ruhen të dhënat e përdoruesve.

Një nga zgjidhjet më performante është përdorimi i MySQL Cluster. **MySQL Cluster** është një teknologji e cila si bazë të funksionimit të saj ka *N*-nyje NDB( **N**etwork **D**atabase), të sinkronizuara me njëra-tjetrën. Arkitektura e MySQL Cluster është *shared-nothing architecture* që do të thotë se nyjet janë plotësisht të pavarura nga njëra tjetra.



Të dhënat replikohen në mënyrë të sinkronizuar midis njeve duke mundësuar që kopje të shumta të të njejtave të dhëna të jenë gjithmone available.

## Anallogji me sisteme të tjera

### **AVTECH - Protect your assets through environment monitoring**

AVTECH është një kompani që ka mbi 27 vite eksperiencë (që nga viti 1988) në monitorimin e Data Center-ave më të rëndësishëm në botë. Kjo kompani ofron paketa sensorësh të montuar në një pajisje të vetme por edhe jep mundësinë edhe për të shtuar sensorë të jashtëm.

Monitoron kushte të ndryshme të një mjedisi duke filluar që nga temperatura, lagështia, zjarri, tymi, niveli i ujit, rrymat e ajrit, hyrjet daljet në një mjedis etj. Një nga shërbimet më të rëndësishmëm që ofron kjo kompani është ai që njihet edhe si **GoToMyDevices**.

**GoToMyDevices** është një webapplication cloud-based që i jep mundësinë përdoruesve të menaxhojnë dhe të monitorojnë mjedise të caktuara. Ky shërbim i fuqishëm të jep mundësi aksesimi nëpërmjet broëser-it nga cdo pajisje smartphone kompjuter apo tablet.

Është një software që nuk ka nevojë për instalim, konfigurim apo menaxhim. Disa nga funksionalitetet bazë të këtij software janë njoftimet për rastet e rrezeve të mundshme, krijimi i raporteve, gjenerimi i grafikëve real-time, cdo veprim kryhet në cloud dhe nuk ka nevojë t'u bëjmë backup të dhënave tona.

**GoToMyDevices** lehtëson marrjen e vendimeve, reagim në kohë dhe parandalim të pasojave, mbi të gjitha mundëson atë që të gjithë bizneset e duan uptime në nivel shumë të lartë.



[Solutions](#) [Room Alert](#) [Sensors](#) [GoToMyDevices](#) [Support](#) [Company](#)

**GoToMyDevices**  
MONITOR, ALERT, LOG, REPORT

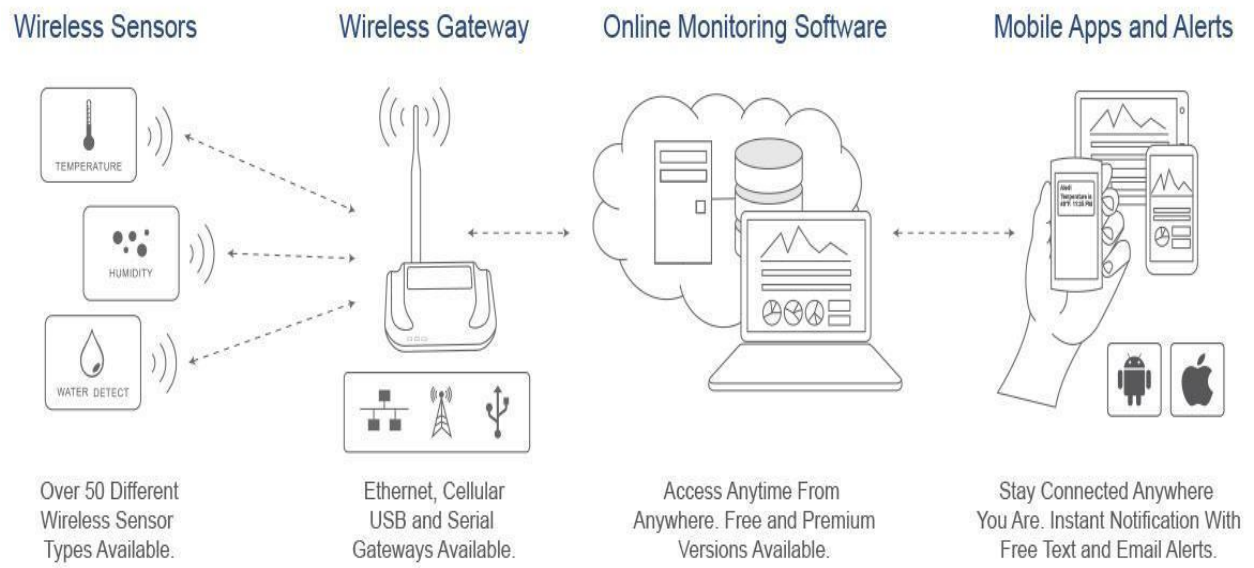
Manage Devices... Anywhere... Anytime!  
AVTECH has you covered!

Our experienced Product Specialists are ready to help you find the perfect monitoring solution.

[Live Sales Chat](#) [Model Comparison](#) [Quote Request](#) [Product Catalog](#)

### **Monni , Wireless Sensor Networks**

Monnit CRP është një lider global në fushën e prodhimit të pajisjeve IoT dhe në sistemet që ofrojnë për monitorimin e mjediseve dhe infrastrukturave të ndryshme. Me qendër në Nuremberg e themeluar në vitin 1999.





## Referenca

**[1] Dorëzimi 1 , Propozim Projekti**

**[2] Amazon IoT Developer Guide:** <http://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf>

**[3] Amazon DynamoDB Documentation :** <https://aws.amazon.com/documentation/dynamodb/>

**[4] Enterprise Applications Administration The Definitive Guide to Implementation and Operations,** by Jeremy Faircloth