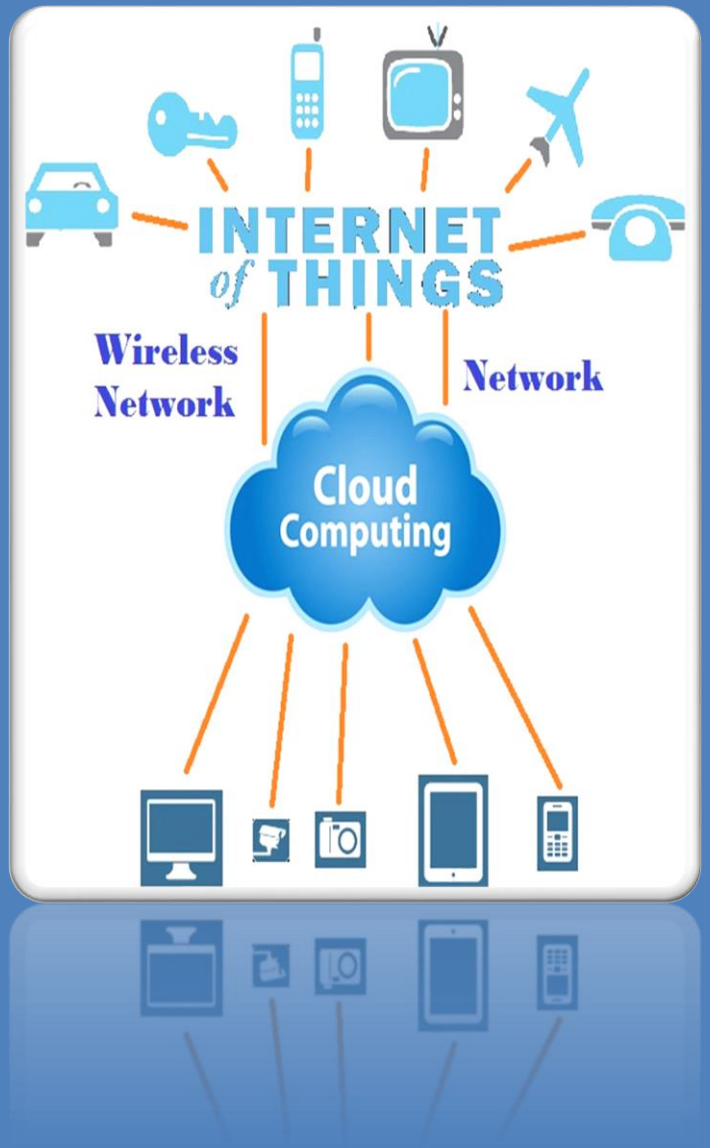
IoT Physical Servers & Cloud Offerings

Presented by

Dr. Amany AbdElSamea



Outline

- Cloud Storage Models and Communication APIs
- WAMP for IOT
- Python Web Application Framework – Django
- Amazon Web Services

Cloud Storage Models and Communication APIs

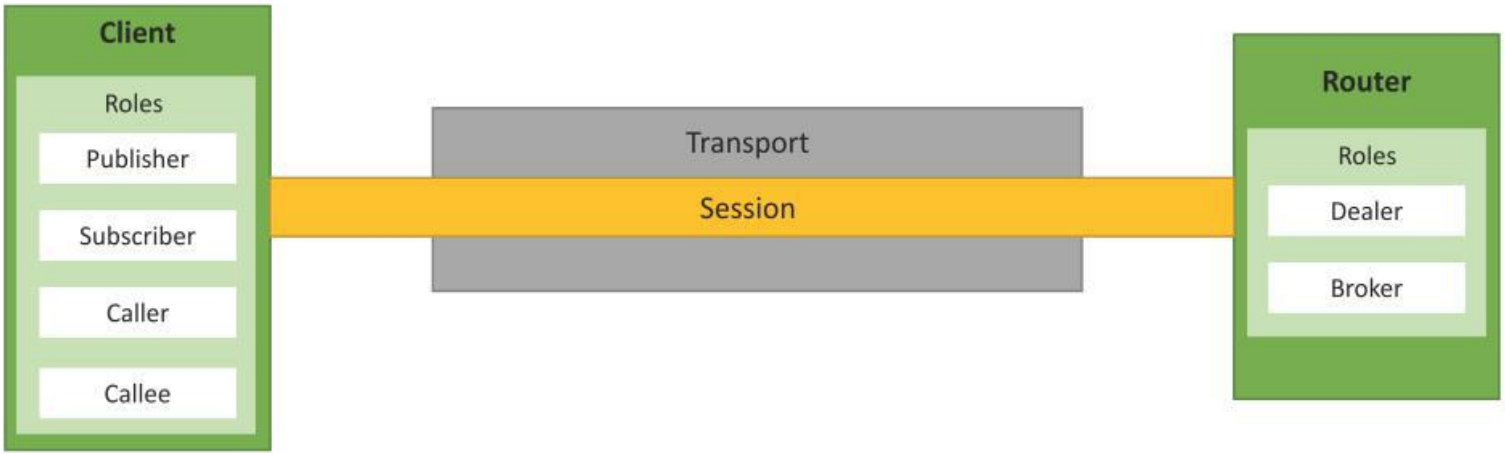
- Cloud computing is a transformative computing paradigm that involve delivering applications and services over the Internet based on pay per use basis.
- We will see WAMP in IoT
- Also AWS & their applications for IoT

WAMP for IoT

- Web Application Messaging Protocol
- Sub-protocol of web socket which provides

- Publish –Subscribe and

-Remote Procedure Call (RPC) messaging patterns



WAMP Concepts

- **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.

WAMP Concepts cont.,

• **Router:** Routers are peers that perform generic call and event routing. In publish-subscribe model Router has the role of a Broker which routes messages published to a topic to all subscribers subscribed to the topic.

In RPC model Router has the role of a dealer:

– 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).

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 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 cloud applications.
- Django consists of an object-relational mapper, a web templating system and a regular-expression based URL dispatcher.

Django Architecture

Model

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

Template

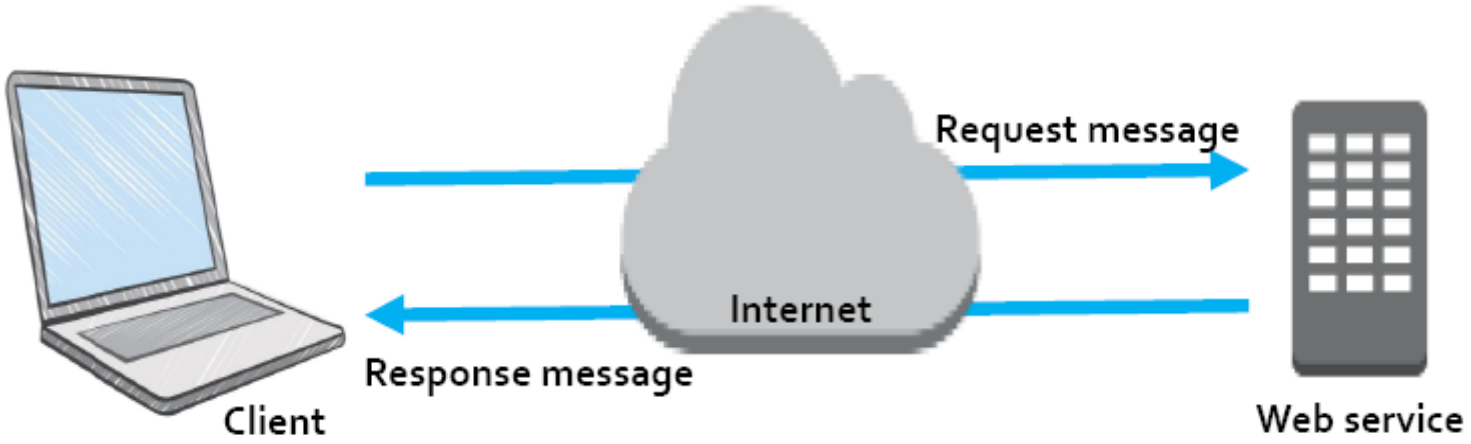
- In a typical Django web application, the template is simply an HTML page with a few extra placeholders
- Django's template language can be used to create various forms of text files (XML, email, CSS, JavaScript, CSV, etc.)

View

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

Web Services

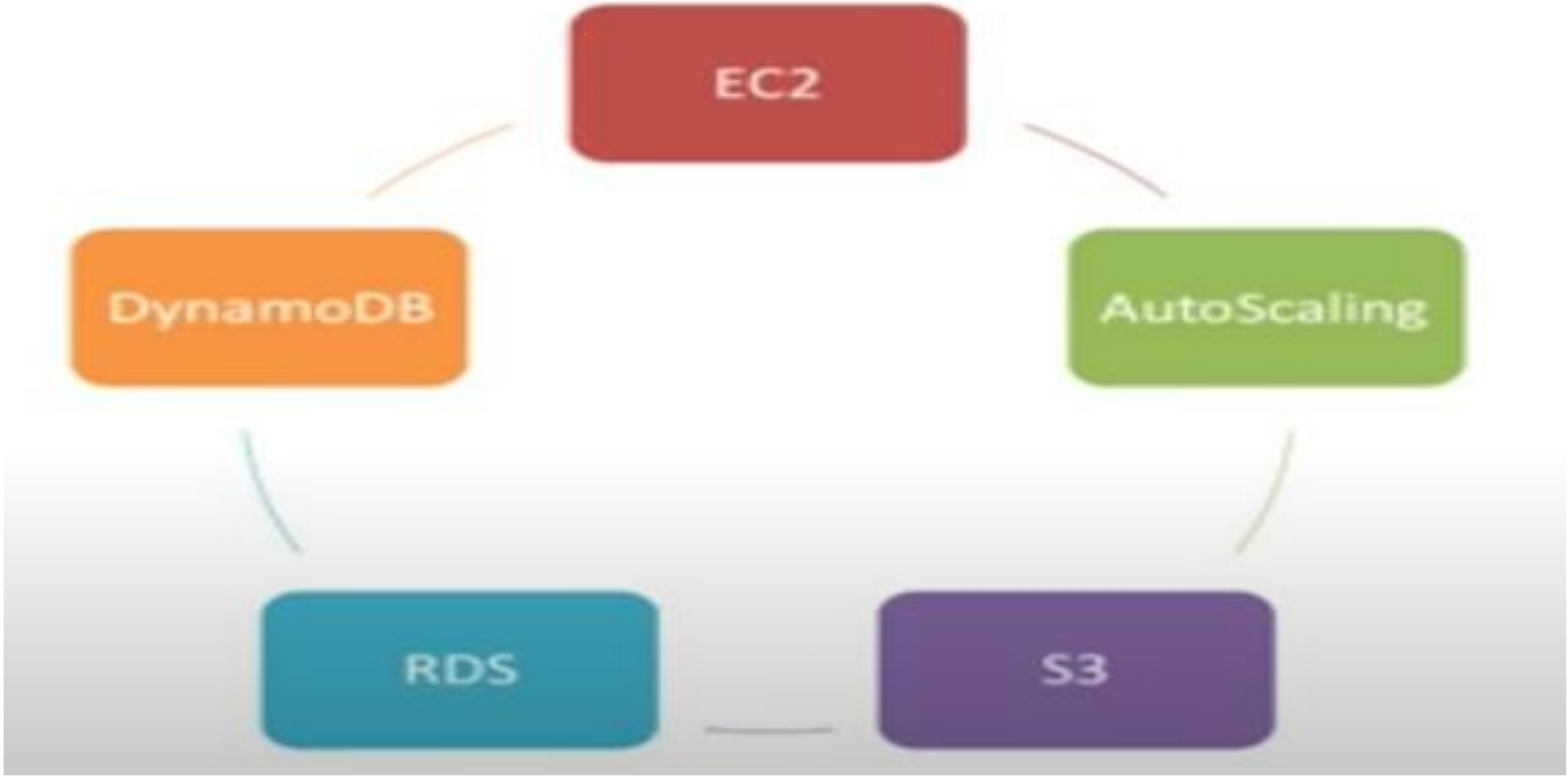
A **web service** is any piece of software that makes itself available over the internet and uses a **standardized format** (XML or JSON) for the request and the response of an **API interaction**.



Amazon Web Services

- **Amazon Web Services (AWS)** is a collection of **remote computing services (web services)** that together make up a **cloud computing platform**, offered over the Internet by Amazon.com.
- Website: http://aws.amazon.com
- AWS is located in **9 geographical 'Regions'**. Each Region is **wholly contained within a single country** and all of its data and services **stay** within the designated Region.
- Each Region has **multiple 'Availability Zones'**, which are **distinct data centers** providing AWS services.

Amazon Web Services for IoT



Amazon EC2

- An IaaS provided by Amazon
- EC2 delivers scalable, pay-as-you-go compute capacity in the cloud.
- Web service that provides computing capacity in the form of virtual machine
- EC2 can be used for several purposes for IoT systems

Amazon EC2 Python Example

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

- 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.

```
#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, ] )
```

Amazon S3

- Online cloud based data storage infrastructure for storing and retrieving large amount of data.
- Offers reliable, scalable, fast, fully redundant and affordable storage infrastructure
- Serve as raw datastore for IoT systems for storing raw data such as sensor data, log data, image, audio, video, etc.

Amazon S3 Python Example

- 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.

```
# Python program for uploading a file to an S3 bucket
import boto.s3

conn = boto.connect_s3(aws_access_key_id='<enter>',
   aws_secret_access_key='<enter>')

def percent_cb(complete, total):
  print ('.')

def upload_to_s3_bucket_path(bucketname, path, filename):
        mybucket = conn.get_bucket(bucketname)
        fullkeyname=os.path.join(path,filename)
        key = mybucket.new_key(fullkeyname)
        key.set_contents_from_filename(filename, cb=percent_cb, num_cb=10)
```

Amazon Autoscaling

- Allows automatically scaling EC2 capacity up (vertical scaling entails installing more powerful systems or upgrading to more powerful components) 0r down (horizontal scaling adds to resources by expanding the number of servers or other processing units) according to user condition.
- Users can increase number of EC2 instances.
- Autoscaling can be used for auto scaling IoT applications and IoT platforms deployed as Amazon EC2.

Amazon RDS

- Web service that allows to create instances of MySQL, Oracle or MS SQL Server in cloud
- Developers can easily setup, operate and scale a relational database in cloud
- Serve as a scalable datastore for IoT systems
- With RDS, IoT system developers can store any amount of data in scalable relational databases

Amazon RDS Python Example

- 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.

```
#Python program for launching an RDS instance (excerpt)
import boto.rds

ACCESS_KEY="<enter>"
SECRET_KEY="<enter>"
REGION="us-east-1"
INSTANCE_TYPE="db.t1.micro"
ID = "MySQL-db-instance-3"
USERNAME = 'root'
PASSWORD = 'password'
DB_PORT = 3306
DB_SIZE = 5
DB_ENGINE = 'MySQL5.1'
DB_NAME = 'mytestdb'
SECGROUP_HANDLE="default"

#Connecting to RDS
conn =  boto.rds.connect_to_region(REGION,
  aws_access_key_id=ACCESS_KEY,
  aws_secret_access_key=SECRET_KEY)


#Creating an RDS instance
db = conn.create_dbinstance(ID, DB_SIZE, INSTANCE_TYPE,
USERNAME, PASSWORD,  port=DB_PORT, engine=DB_ENGINE,
db_name=DB_NAME, security_groups = [ SECGROUP_HANDLE, ] )
```

Amazon DynamoDB

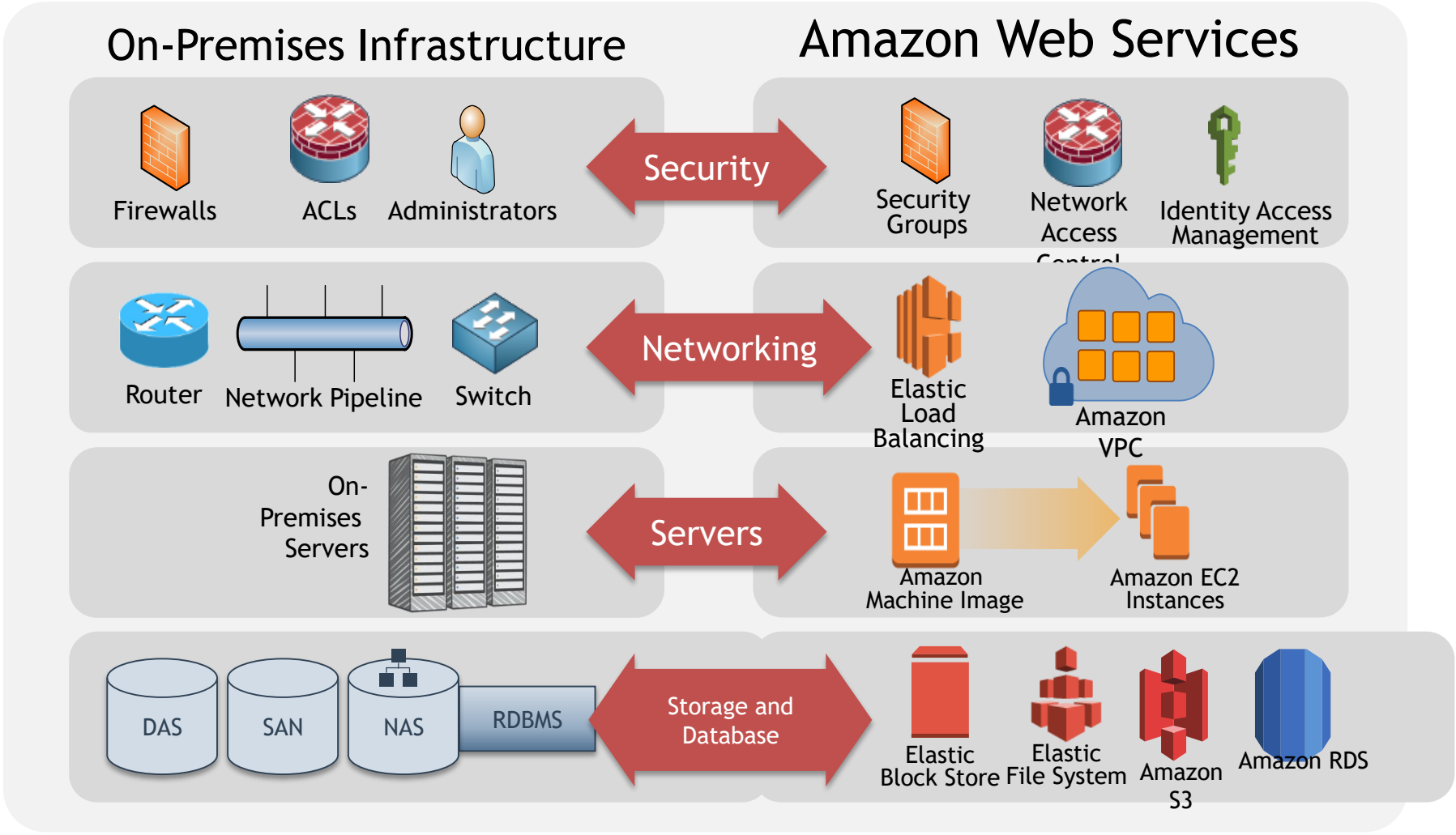
- Fully-managed, scalable, high performance No-SQL database service
- Serve as scalable datastore for IoT systems
- With DynamoDB, IoT system developers can store any amount of data and serve any level of requests for the data.

Amazon DynamoDB Python Example

- 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.

```
# Python program for creating a DynamoDB table (excerpt)
import boto.dynamodb

ACCESS_KEY="<enter>"
SECRET_KEY="<enter>"
REGION="us-east-1"

#Connecting to DynamoDB
conn = boto.dynamodb.connect_to_region(REGION,
  aws_access_key_id=ACCESS_KEY,
  aws_secret_access_key=SECRET_KEY)

table_schema = conn.create_schema(
      hash_key_name='msgid',
      hash_key_proto_value=str,
      range_key_name='date',
      range_key_proto_value=str
   )

#Creating table with schema
table = conn.create_table(
      name='my-test-table',
      schema=table_schema,
      read_units=1,
      write_units=1
   )
```

On-Premises and AWS Comparison



AWS by Category: Core Services

Compute


Amazon EC2


AWS Lambda


Auto Scaling


AWS Elastic Beanstalk


Amazon Elastic Container Registry


Amazon Elastic Container Service


Amazon Lightsail


AWS Batch

Networking


Amazon VPC


Amazon Route 53


AWS Direct Connect


Elastic Load Balancing

Storage


Amazon S3


Amazon EBS


Amazon CloudFront


Amazon Glacier


Amazon Elastic File System


AWS Snowball


Storage Gateway


AWS Snowmobile

Database


Amazon RDS


Amazon DynamoDB


Amazon Redshift


AWS Database Migration Service


Amazon ElastiCache

AWS by Category: Foundational Services

Analytics


Amazon EMR


AWS Data Pipeline


Amazon Elasticsearch


Amazon Kinesis


Amazon Machine Learning


Amazon QuickSight


Amazon Redshift


Amazon Athena

Enterprise Apps


Amazon WorkSpaces


Amazon WorkMail


Amazon WorkDocs

Mobile Services


AWS Mobile Hub


Amazon SNS


Amazon Cognito


AWS Device Farm


Amazon Mobile Analytics


AWS Mobile SDKs


Amazon Pinpoint

Internet of Things


AWS IoT


AWS Greengrass

Questions