

# Northeastern University – Silicon Valley

## CS 6620 Cloud Computing

### Homework Set #4 [100 points]

---

***INSTRUCTIONS:*** Please provide clear explanations in your own sentences, directly answering the question, demonstrating your understanding of the question and its solution, in depth, with sufficient detail. Submit your solutions [PDF preferred]. Include your full name. Do not email the solutions.

#### **PART I: Concepts and Theory, Algorithms [60 points]**

Please provide accurate, concise (maximum 10 sentence) answers. **All questions carry 10 points.**

Based on your own thinking and study of text/references, explain what the below concepts are and why and how they are used in an example eCommerce web application like Amazon.com etc. please provide example of how they are used for eCom.

1. Consider the basic e-commerce web application from homework 3. How many virtual machines (VMs) would you need to deploy this application? Using this as an example, please answer the following questions, with 1-2 para explanations for each question: [30 points]
  - A. How is a VM different from a physical server? what are the advantages of using VMs in cloud computing (SaaS)? what are the disadvantages?
  - B. How is a docker container (or any container) different from a VM? Please explain the key Differences between VMs and containers - in terms of how they work, and other essential differences.
  - C. Can you think of SaaS application workloads which will not run well in VMs? please provide two example use cases, explaining why VMs are not suitable, and why you might need an actual physical server to run them.
2. Explain using text and diagrams: what is serverless computing? how is it different from the regular cloud computing (i.e., IaaS) using VMs? How is it different from the cloud computing (i.e., IaaS) using Containers? [15 points]
3. What are web services? How do they work? How are they different from the traditional MVC web apps? [10 points]
4. How are microservices different from web services? What is the relationship between microservices and serverless computing on the cloud? please explain. [10 points]
5. Suppose you're deploying the basic ecommerce web application as SaaS on the cloud. Please explain the steps involved in deploying this app into VMs on AWS (with details on the directory structure of the files deployed etc.) [10 points]

#### **PART II: LAB [25 points]**

**Coding needed:** Write code to deploy the basic ecommerce Web App you built for Homework 3 on AWS using Lambda (serverless). Keep it simple, following the below references.

You can choose any one of the tools below to deploy. Please package your code and all the configuration files. Include a pdf file with the testing results: the app runs locally and runs on AWS with aws ip address.

**Stretch goal:** deploy your dataset on to the AWS S3 bucket, and test CRUD operations. (Not required for grade)

#### **References**

Basic AWS Lambda app deployment

1. Deploying a Django project on AWS Lambda using Serverless  
<https://dev.to/vaddimart/deploy-django-app-on-aws-lambda-using-serverless-part-1-1i90>

2. Using Zappa and AWS Lambda to deploy serverless Django apps  
<https://blog.logrocket.com/zappa-and-aws-lambda-for-serverless-django/>

## ANS (Qichen An)

*1. Consider the basic e-commerce web application from homework 3. How many virtual machines (VMs) would you need to deploy this application? Using this as an example, please answer the following questions, with 1-2 paragraph explanations for each question: [30 points]*

*A. How is a VM different from a physical server? what are the advantages of using VMs in cloud computing (SaaS)? what are the disadvantages?*

*B. How is a docker container (or any container) different from a VM? Please explain the key Differences between VMs and containers - in terms of how they work, and other essential differences.*

*C. Can you think of SaaS application workloads which will not run well in VMs? please provide two example use cases, explaining why VMs are not suitable, and why you might need an actual physical server to run them.*

### A.

The architecture of a physical server is quite plain. Each server has its own hardware: Memory, network, processing and storage resources. On this hardware, the server operating system is loaded. From the OS users can then run the applications.

Virtual Machines are made possible by installing a hypervisor on top of a “bare-metal” server. A common approach is to virtualize the hardware of the underlying physical server and present this virtualized hardware to the operating system.

#### **Advantages of VMs in cloud computing (SaaS)**

For software development, the advantages that virtual machines offer in terms of cost, physical footprint, lifespan, migration, performance, efficiency, and disaster recovery/high availability are far greater than running a single workload on a single physical server.

#### **Disadvantages of VMs in cloud computing (SaaS)**

Virtual machines are less efficient than real machines because they access the hardware indirectly. Running software on top of the host operating system means that it will have to request access to the hardware from the host. That will slow the usability. Also, a virtual machine can be infected with the weaknesses of the host machine. A regular computer devoid of virtual machines would then only be affected. But a computer with a number of virtual machines would then infect each of those “machines” as well.

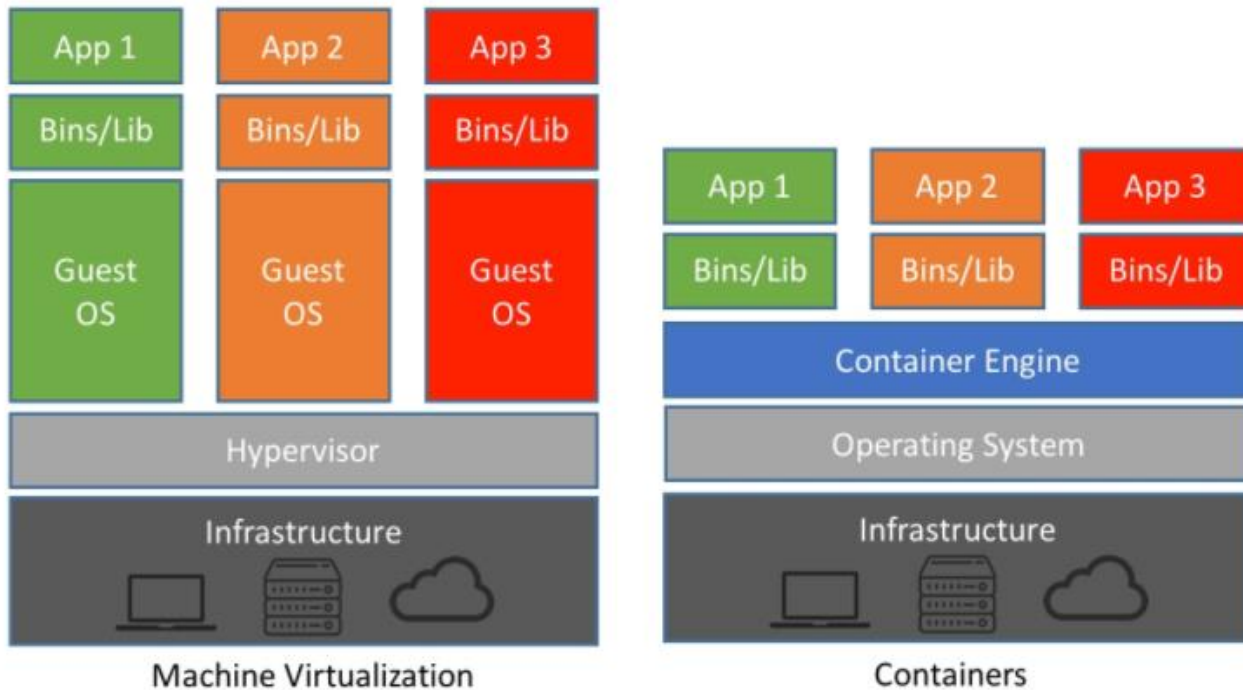
### B.

A virtual machine is a system which acts exactly like a computer, whereas Docker is a software development tool and a virtualization technology that makes it easy to develop, deploy, and manage applications by using containers.

## Main difference: architecture

Within each virtual machine runs a unique guest operating system. Each VM has its own binaries, libraries, and applications that its services, and the VM may be many gigabytes in size.

While containers sit on top of a physical server and its host OS—for example, Linux or Windows. Each container shares the host OS kernel and, usually, the binaries and libraries, too. Shared components are read-only. So, they are only megabytes in size and take just seconds to start, versus gigabytes and minutes for a VM.



Containers also reduce management overhead. Because they share a common operating system, only a single operating system needs care and feeding for bug fixes, patches, and so on.

## Security

Virtual machines are stand-alone with their kernel and security features. Therefore, applications needing more privileges and security run on VM. For Docker containers, providing root access to applications and running them with administrative premises is not recommended because containers share the host kernel.

## Portability

Virtual machines are isolated from their OS, and so, they are not ported across multiple platforms without incurring compatibility issues. For Dockers, containers packages are self-contained and can run applications in any environment, and since they don't need a guest OS, they can be easily ported across different platforms.

## Performance

Virtual machines are more resource-intensive than Docker containers as the VM need to load the entire OS to start. Scaling up and duplicating containers is simple and easy as compared to VM because there is no need to install an operating system in them.

### C.

As mentioned in the previous two questions, VMs vs. physical servers / containers, in some cases, VMs might not be the most suitable way for SaaS application workloads.

For some SaaS application workloads, using VMs imposes a large performance penalty. Every virtual machine, which must run its own execution environment and copy of the operating system, uses up server processing cycles that you otherwise could use to run the applications. In this situation, Docker containers would be a greater alternative. Container efficiency also allows for higher server utilization rates. And from a performance perspective, containers are a much better execution foundation because containers are more efficient at initialization thanks to its size.

Use cases: Netflix, for example, runs its entire microservices-based offering on Amazon Web Services, using AWS instances. Also, Spotify use containers.

There are some situations that might need an actual physical server to run SaaS application. For example, organizations running services and operations which require highly productive computing hardware for their implementation. Physical servers can provide more performance management.

*2. Explain using text and diagrams: what is serverless computing? how is it different from the regular cloud computing (i.e., IaaS) using VMs? How is it different from the cloud computing (i.e., IaaS) using Containers?*

*[15 points]*

#### **what is serverless computing?**

Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on demand and manages servers on behalf of the customer. Serverless computing does not hold resources in volatile memory; computing is rather done in short bursts with the results persisted to storage. When an app is not in use, there are no computing resources allocated to the app. It can be a form of utility computing. Developers of serverless applications are not concerned with capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers.

#### **How is it different from the regular cloud computing (i.e., IaaS) using VMs?**

In serverless computing, the application logic is executed in an environment without visible processes, operating systems, servers or virtual machines. Such an environment is actually running on the top of an operating system and uses physical servers or virtual machines, but the responsibility for provisioning and managing the infrastructure belongs entirely to the service provider. As a result, developers simply write small, focused bits of

independent code to accomplish a single task. The service provider then runs the developer's piece of code just when a user makes a request for that service. The service provider will automatically allow the appropriate number of these independent tasks to run simultaneously to meet demand. Efficiencies are gained because the provider is free to decide how to best utilize its infrastructure to serve requests from all of its clients.

A virtual server runs on an actual, physical server. It runs an operating system, and contains database software, a web server, and other applications to deliver your site or services to customers. The virtual server is maintained by connecting to it remotely over a network. Once connected, the administrator updates software and performs other software maintenance tasks as if the virtual server were a dedicated physical server. A single physical server can run hundreds of virtual servers. With the virtual server model, when the user wants to expand and scale as business grows, it requires you to maintain the virtual server. The users have to consciously increase the storage capacity and tell it how to handle the increased volume. It is a good solution if you have long tasks running on the server, as they can become expensive in the serverless model.

### **How is it different from the cloud computing (i.e., IaaS) using Containers?**

Physical machine:

'serverless' computing actually runs on servers, but it is up to the serverless vendor to provision server space as it is needed by the application; no specific machines are assigned for a given function or application. On the other hand, each container lives on one machine at a time and uses the operating system of that machine, though they can be moved easily to a different machine if desired.

Scalability:

In a container-based architecture, the number of containers deployed is determined by the developer in advance. In contrast, in a serverless architecture, the backend inherently and automatically scales to meet demand.

Cost:

Containers are constantly running, and therefore cloud providers have to charge for the server space even if no one is using the application at the time.

There are no continued expenses in a serverless architecture because application code does not run unless it is called. Instead, developers are only charged for the server capacity that their application does in fact use.

Maintenance:

Containers are hosted in the cloud, but cloud providers do not update or maintain them. Developers have to manage and update each container they deploy.

From a developer's perspective, a serverless architecture has no backend to manage. The vendor takes care of all management and software updates for the servers that run the code.

Time of deployment:

Containers take longer to set up initially than serverless functions because it is necessary to configure system settings, libraries, and so on. Once configured, containers take only a few seconds to deploy. But because serverless functions are smaller than container microservices and do not come bundled with system dependencies, they only take milliseconds to deploy.

Testing:

It is difficult to test serverless web applications because the backend environment is hard to replicate on a local environment. In contrast, containers run the same no matter where they are deployed, making it relatively simple to test a container-based application before deploying it to production.

*3. What are web services? How do they work? How are they different from the traditional MVC web apps?  
[10 points]*

Web Service is a way to expose the functionality of an application to other application, without a user interface. It commonly provides an object-oriented Web-based interface to a database server, utilized for example by another Web server, or by a mobile app, that provides a user interface to the end-user. Many organizations that provide data in formatted HTML pages will also provide that data on their server as XML or JSON, often through a Web service to allow syndication. Another application offered to the end-user may be a mashup, where a Web server consumes several Web services at different machines and compiles the content into one user interface.

### **How do they work?**

A web service works by supporting interoperable machine-to-machine communication using a network. As such, web services tend to be connected with SOA or Service Oriented Architecture. This allows for different features to be separated then made available as various services within a network.

A web service supports communication among numerous apps with HTML, XML, WSDL, SOAP, and other open standards. XML tags the data, SOAP transfers the message, and WSDL describes the service's accessibility.

Here's an instance of how it works: A web service sits between two sets of java, .net, or PHP apps providing a way for these applications to communicate over a network. On one side, for example, a java app interacts with the java, .net, and PHP apps on the other end by way of the web service communicating an independent language.

### **Difference with MVC Framework web apps.**

The above part of this question describes web services. Here are also some advantages of using MVC framework.

- a. It supports faster development for web apps.
- b. It provides multiple views.

- c. It supports for asynchronous technique.
- d. The modification does not affect the entire model.
- e. MVC returns the data without formatting.
- f. It's SEO friendly development platform.

*4. How are microservices different from web services? What is the relationship between microservices and serverless computing on the cloud? please explain. [10 points]*

Microservice is a small, autonomous application that performs a specific service for a larger application architecture. Web service is a strategy to make the services of one application available to other applications via a web interface.

Both microservices and web services can be used to build application architectures. Here's a brief description of both architectural styles:

**Microservices application architecture:** A modular, services-oriented application architecture comprised of loosely connected, independently running microservices. These microservices usually offer APIs so other microservices and apps can integrate with them.

**Web services application architecture:** A modular, services-oriented application architecture where the applications that comprise the architecture connect via web services. Developers can use web services to connect microservices, monolithic applications, and more to form a larger application.

### **Relationship between microservices and serverless computing on the cloud**

In serverless computing, the application logic is executed in an environment without visible processes, operating systems, servers or virtual machines. Such an environment is actually running on the top of an operating system and uses physical servers or virtual machines, but the responsibility for provisioning and managing the infrastructure belongs entirely to the service provider."

As a result, developers simply write small, focused bits of independent code to accomplish a single task – like handling a login form or completing a search. The service provider then runs the developer's piece of code just when a user makes a request for that service. The service provider will automatically allow the appropriate number of these independent tasks to run simultaneously to meet demand. Efficiencies are gained because the provider is free to decide how to best utilize its infrastructure to serve requests from all of its clients.

Applications built as Micro Services can be broken into multiple component services and this service can be a Web Service, which should run unique process and then redeployed independently without compromising the integrity of an application.

focus on, say, UIs, databases, technology layers, or server-side logic, Micro Services architecture utilizes cross-functional teams. The responsibilities of each team are to make specific products based on one or more individual services communicating via message bus. It means that when changes are required, there won't necessarily be any reason for the project, as a whole, to take more time or for developers to have to wait for budgetary approval before individual services can be improved. Most development methods focus on projects: a piece of code that has

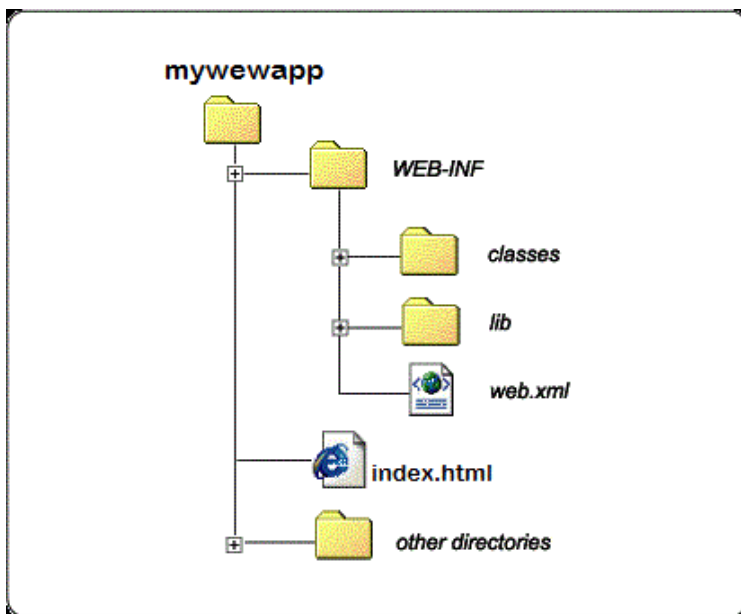
to offer some predefined business value must be handled over to the client, and is then periodically maintained by a team. But in Micro Services, a team owns the product for its lifetime.

*5. Suppose you're deploying the basic ecommerce web application as SaaS on the cloud. Please explain the steps involved in deploying this app into VMs on AWS (with details on the directory structure of the files deployed etc.)*  
[10 points]

To deploy web services (Java-based for example), I need to prepare the web service that have been created and tested, and ready to be deployed. This takes the form of simply wrapping the various files of the service up into a single 'archive' file.

### Creating an archive

Before running the application you need to package the resources inside it (servlets, JSP's,xml files etc.) in a standardized way as shown below. Web service package structure and contents



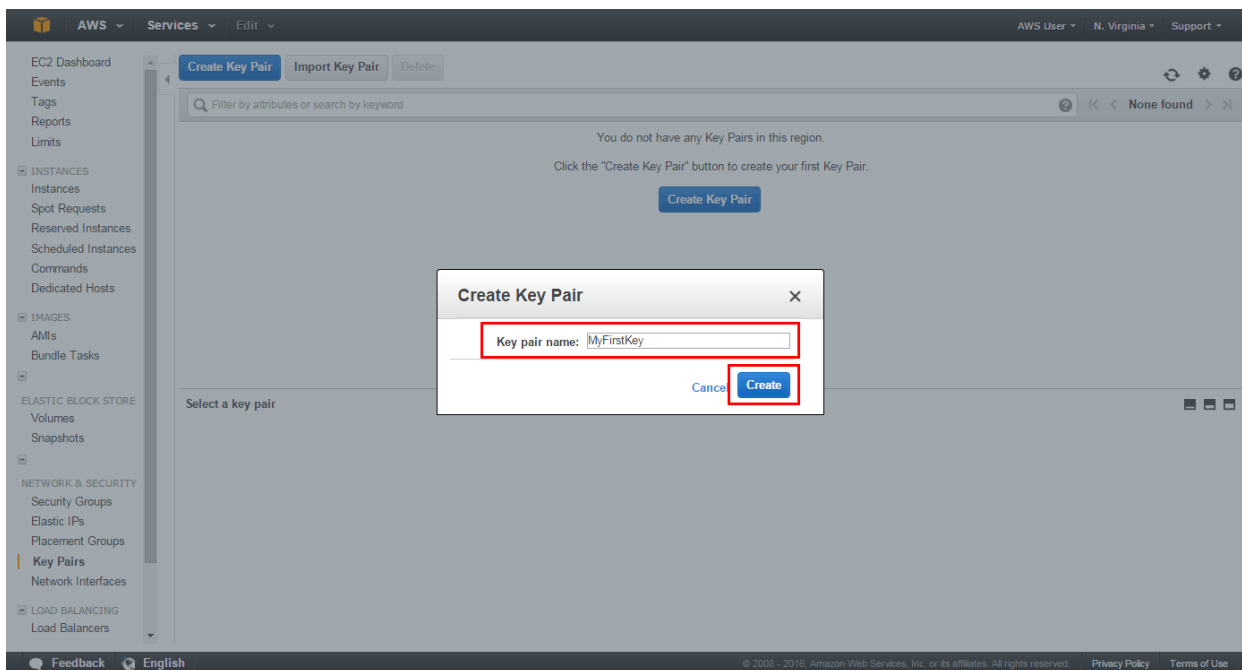
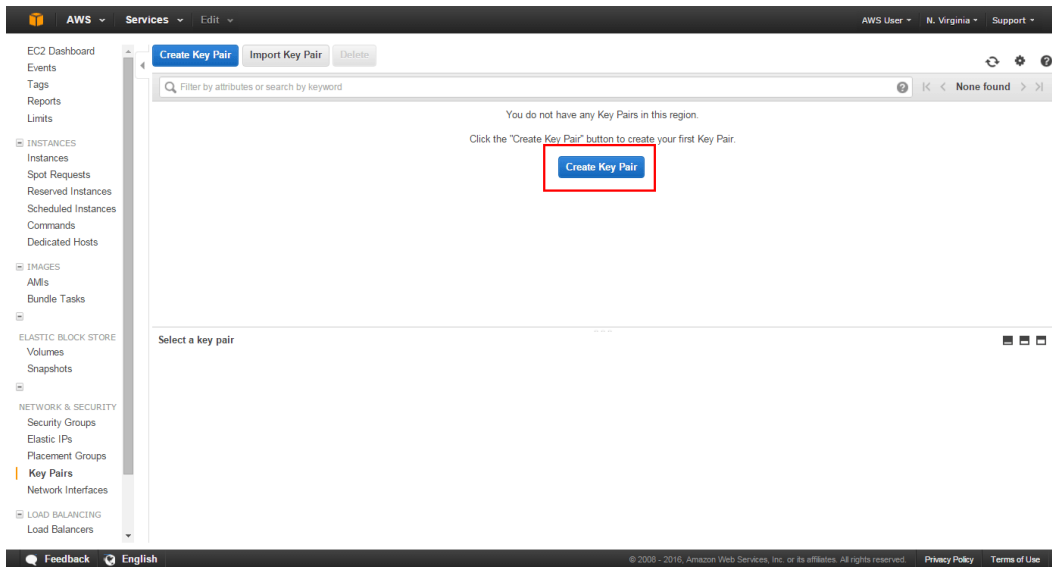
### Deploying an archive to AWS

For this question, I will use AWS CodeDeploy, a service that automates code deployments to AWS or on-premises servers, to deploy code to virtual machines that I create and manage with Amazon EC2.

#### Step 1: Create a Key Pair

This key pair is to access the virtual machine with Amazon EC2.





## Step 2: Enter the CodeDeploy Console

Services
Resource Groups
N. Virginia
Support

### AWS services

▼ All services

**Compute**  
EC2  
EC2 Container Service  
Lightsail  
Elastic Beanstalk  
Lambda  
Batch

**Developer Tools**  
CodeCommit  
CodeBuild  
**CodeDeploy**  
CodePipeline  
X-Ray

**Internet of Things**  
AWS IoT  
**Game Development**  
Amazon GameLift

**Storage**  
S3  
EFS  
Glacier  
Storage Gateway

**Management Tools**  
CloudWatch  
CloudFormation  
CloudTrail  
Config  
OpsWorks  
Service Catalog  
Trusted Advisor  
Managed Services

**Mobile Services**  
Mobile Hub  
Cognito  
Device Farm  
Mobile Analytics  
Pinpoint

**Database**  
RDS  
DynamoDB  
ElastiCache  
Redshift

**Security, Identity & Compliance**  
IAM  
Inspector  
Certificate Manager  
Directory Service  
WAF & Shield  
Compliance Reports

**Application Services**  
Step Functions  
SWF  
API Gateway  
Elastic Transcoder

**Networking & Content Delivery**  
VPC  
CloudFront  
Direct Connect  
Route 53

**Analytics**  
Athena  
EMR  
CloudSearch  
Elasticsearch Service  
Kinesis  
Data Pipeline  
QuickSight

**Business Productivity**  
WorkDocs  
WorkMail  
Amazon Chime

**Migration**  
Application Discovery Service  
DMS  
Server Migration  
Snowball

**Artificial Intelligence**  
Lex  
Polly  
Rekognition  
Machine Learning

**Desktop & App Streaming**  
WorkSpaces  
AppStream 2.0

### Featured next steps

**Manage your costs**  
Get real-time billing alerts based on your cost and usage budgets. [Start now](#)

**Get best practices**  
Use AWS Trusted Advisor for security, performance, cost and availability best practices. [Start now](#)

### What's new?

**Announcing AWS Batch**  
Now generally available, AWS Batch enables developers, scientists, and engineers to process large-scale batch jobs with ease. [Learn more](#)

**Announcing Amazon Lightsail**  
See how this new service allows you to launch and manage your VPS with AWS for a low, predictable price. [Learn more](#)  
[See all](#)

**AWS Marketplace**  
Discover, procure, and deploy popular software products that run on AWS.

**Have feedback?**  
[Submit feedback](#) to tell us about your experience with the AWS Management Console.

Services
Edit
AWS User
N. Virginia
Support

Deploy your applications to instances in a fast, consistent, and reliable way.

[Get Started Now](#)

**Upload**  
Configure your application, and then upload it to Amazon S3 or GitHub in preparation for deployment.  
[Learn more](#)

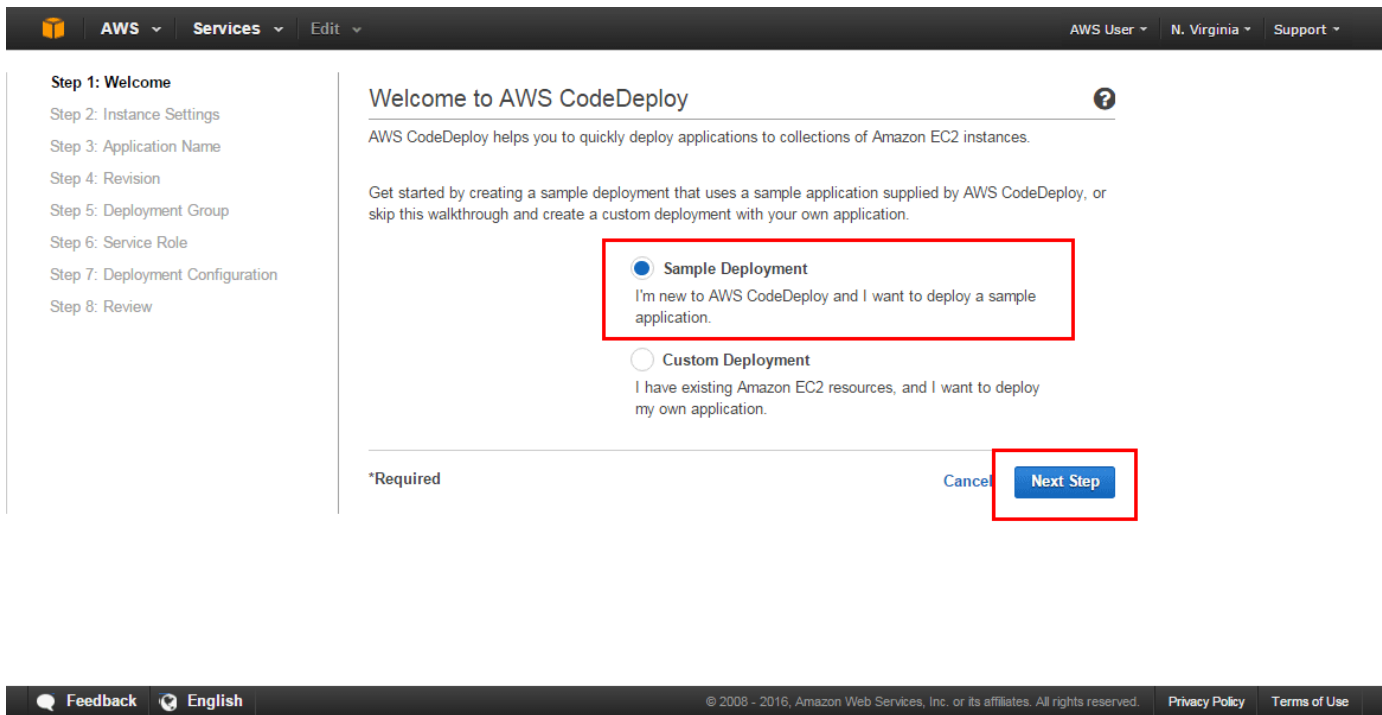
**Deploy**  
Specify your instances, and then deploy your application from Amazon S3 or GitHub to them.  
[Learn more](#)

**Track**  
Track and monitor the status of your deployments through real-time monitoring tools.  
[Learn more](#)

AWS CodeDeploy Documentation & Support  
[Getting Started](#) | [Documentation](#) | [Support](#) | [Forums](#)

[Feedback](#)
[English](#)

© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
[Privacy Policy](#)
[Terms of Use](#)



### Step 3: Launch a Virtual Machine

Launch an AWS virtual machine to deploy my code on

ServicesResource GroupsN. VirginiaSupport

AWS services

Find a service by name (for example, EC2, S3, Elastic Beanstalk).

All services

Compute

EC2

EC2 Container Service

Lightsail

Elastic Beanstalk

Lambda

Batch

Storage

S3

EFS

Glacier

Storage Gateway

Database

RDS

DynamoDB

ElastiCache

Redshift

Networking & Content Delivery

VPC

CloudFront

Direct Connect

Route 53

Migration

Application Discovery Service

DMS

Server Migration

Snowball

Developer Tools

CodeCommit

CodeBuild

CodeDeploy

CodePipeline

X-Ray

Management Tools

CloudWatch

CloudFormation

CloudTrail

Config

OpsWorks

Service Catalog

Trusted Advisor

Managed Services

Security, Identity & Compliance

IAM

Inspector

Certificate Manager

Directory Service

WAF & Shield

Compliance Reports

Analytics

Athena

EMR

CloudSearch

Elasticsearch Service

Kinesis

Data Pipeline

QuickSight

Artificial Intelligence

Lex

Polly

Rekognition

Machine Learning

Internet of Things

AWS IoT

Game Development

Amazon GameLift

Mobile Services

Mobile Hub

Cognito

Device Farm

Mobile Analytics

Pinpoint

Application Services

Step Functions

SWF

API Gateway

Elastic Transcoder

Messaging

Simple Queue Service

Simple Notification Service

SES

Business Productivity

WorkDocs

WorkMail

Amazon Chime

Desktop & App Streaming

WorkSpaces

AppStream 2.0

Featured next steps

Manage your costs

Get real-time billing alerts based on your cost and usage budgets. [Start now](#)

Get best practices

Use AWS Trusted Advisor for security, performance, cost and availability best practices. [Start now](#)

What's new?

Announcing AWS Batch

Now generally available, AWS Batch enables developers, scientists, and engineers to process large-scale batch jobs with ease. [Learn more](#)

Announcing Amazon Lightsail

See how this new service allows you to launch and manage your VPS with AWS for a low, predictable price. [Learn more](#)

[See all](#)

AWS Marketplace

Discover, procure, and deploy popular software products that run on AWS.

Have feedback?

[Submit feedback](#) to tell us about your experience with the AWS Management Console.

AWS ServicesEdit

AWS UserN. VirginiaSupport

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Instance Settings

To help test the sample application deployment, AWS CodeDeploy will use AWS CloudFormation to launch three Amazon EC2 instances with the configuration below. After they launch, the instances will be ready to participate in deployments.

Operating System\*

☒ Amazon Linux

☐ Windows Server

Instance Type\*

t1.micro

Key Pair Name\*

MyFirstKey

Tag Key and Value\*

Name

CodeDeployDemo

Your instances are ready. To continue, click Next Step.

\*Required

Cancel

Previous

Skip This Step

Next Step

## Step 4: Name the Application and Review the Application Revision

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Application Name

Type a name that uniquely identifies the application that you want to deploy. AWS CodeDeploy will group the application revision, deployment group, service role, and deployment configuration under this application name.

Application Name\* HelloWorld

\*Required

Cancel

Previous

Next Step

Feedback

English

© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Privacy Policy

Terms of Use

In this view, I can review information about the application revision you'd like to deploy to EC2. An application revision is an archive file containing source content, which was created before deploying.

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Revision

A revision in AWS CodeDeploy is a version of an application that you want to deploy. Our sample application revisions are stored in Amazon S3.

Revision Type

Sample Amazon Linux Application

Revision Location

https://s3.amazonaws.com/aws-codedeploy-us-east-1/samples/latest/SampleApp\_Linux.zip

Download Sample Bundle

Revision Description

The URL of the sample application in Amazon S3.  
Sample web page for Amazon Linux. To view the sample web page after deployment, from your web browser go to http://<Public DNS>, for example http://ec2-12-345-678-901.compute-1.amazonaws.com.

\*Required

Cancel

Previous

Next Step

Feedback

English

© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Privacy Policy

Terms of Use

## Step 5: Create a Deployment Group

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Deployment Group

Create a new deployment group, which is a collection of Amazon EC2 instances to deploy to.

Deployment Group Name\*

DemoFleet

Add Instances

Specify existing Amazon EC2 instances to deploy to by entering their Amazon EC2 tags (key/value pairs), Auto Scaling group names, or both.

Search by Amazon EC2 Tags

	Key	Value	Instances	
1	Name	CodeDeployDemo	3	✕
2				

Search by Auto Scaling Group Names

Total Matching Amazon EC2 Instances: 3

\*Required

Cancel

Previous

Next Step

## Step 6: Create a Service Role

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Service Role

Select an existing service role that allows AWS CodeDeploy to work with other dependent AWS services on your behalf during a deployment. If you're not sure if you already have a service role with the correct permissions, in the Service Role drop-down list select Create A New Service Role, and we will create one for you.

Service Role\*

Create a new service role

Preview Service Role Policy

Role Name\*

CodeDeploy\_HelloWorld

\*Required

Cancel

Previous

Next Step

## Step 7: Deploy the Application

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Deployment Configuration

Choose from a list of default deployment configurations, or create a custom configuration.

Default Deployment Configurations

Create Custom Deployment Configuration

One at a Time

The deployment will:

Deploy to one instance at a time. Succeed if all instances succeed. Fail after the very first failure. Allow the deployment to succeed for some instances, even if the overall deployment fails.

Example:

If you deploy your application to 3 instances, this configuration will deploy to one instance at a time.

Succeeds if all 3 instances succeed.

Fails after any instance fails.

Half at a Time

The deployment will:

Deploy to up to half of the instances at a time, with fractions rounded down. Succeed if at least half of the instances succeed, otherwise it will fail. The deployment may succeed for some instances, even if the overall deployment fails.

Example:

If you deploy your application to 3 instances, this configuration will deploy to one instance at a time.

Succeeds if 2 or more instances succeed.

Fails if 2 or more instances fail.

All at Once

The deployment will:

Deploy to all instances at once. Succeed if at least one instance succeeds. Fail after all instances fail.

Example:

If you deploy your application to 3 instances, this configuration will deploy to all 3 instances at once.

Succeeds if any instance succeeds.

Fails if all instances fail.

\*Required

Cancel

Previous

Next Step

Step 1: Welcome

Step 2: Instance Settings

Step 3: Application Name

Step 4: Revision

Step 5: Deployment Group

Step 6: Service Role

Step 7: Deployment Configuration

Step 8: Review

Review

Review the details of your deployment. To make any changes, click Edit, Previous, or one of the steps in the navigation pane. When you're ready to deploy with these details, click Deploy Now.

You are about to create the following deployment.

Application

You will create the application HelloWorld

Edit

Revision

You will deploy the following revision of the application HelloWorld

Revision: https://s3.amazonaws.com/aws-codedeploy-us-east-1/samples/latest/SampleApp\_Linux.zip

Edit

Deployment Group

HelloWorld will be deployed to your instances using the deployment group DemoFleet

Edit

Service Role

DemoFleet will use the CodeDeploy\_HelloWorld service role to access the instances.

Edit

Deployment Configuration

You will deploy HelloWorld to your instances using the following deployment configuration

Deployment Configuration: CodeDeployDefault\_OneAtATime

Edit

\*Required

Cancel

Previous

Deploy Now

## Test the deployed service in the final

I can use some tools to test the deployed web application.

15

## Part 2: LAB

Activate the virtual environment

```
C:\>cd D:\IDE\Djang\Django
D:\IDE\Djang\Django>venv\Scripts\activate.bat
(venv) D:\IDE\Djang\Django>
```

Install kappa (0.6.0)

```
(venv) D:\IDE\Djang\Django>cd D:\IDE\Djang\Django\kappa-0.6.0\kappa-0.6.0
(venv) D:\IDE\Djang\Django\kappa-0.6.0\kappa-0.6.0>python setup.py install
Warning: 'classifiers' should be a list, got type 'tuple'
```

Install zappa (0.53.0)

```
(venv) D:\IDE\Djang\Django\kappa-0.6.0\kappa-0.6.0>cd D:\IDE\Djang\Django
(venv) D:\IDE\Djang\Django>pip install zappa==0.53.0
Collecting zappa==0.53.0
```

Zappa initialization

```
(venv) D:\IDE\Djang\Django>zappa init
d:\ide\pycharm\anaconda\ins\lib\site-packages\setuptools\distutils_patch.py:25: UserWarning: Distutils was imported before Setuptools. This usage is discouraged and may exhibit undesirable behaviors or errors. Please use Setuptools' object directly or at least import Setuptools first.
  warnings.warn(

  ZAPPA
Welcome to Zappa!
```

Zappa deployment

```
(venv) D:\IDE\Djang\Django>zappa deploy dev
Important! A new version of Zappa is available!
```



Then, configure the api gateway to the 'settings.py'

```
ALLOWED_HOSTS = ['127.0.0.1', 'oakh71wulb.execute-api.us-west-1.amazonaws.com',]
```

Zappa update

```
D:\IDE\Djang\Django\ecommerce>zappa update
```

Some other configurations

1. change add\_description to set\_description

```
self.cf_template = troposphere.Template()
self.cf_template.set_description("Automatically generated with Zappa")
self.cf_template.add_resource(resource)
```

2. zappa\_settings.json

```
{
  "dev": {
    "django_settings": "ecommerce.settings",
    "profile_name": "default",
    "project_name": "django",
    "aws_region": "us-west-1",
    "runtime": "python3.8",
    "s3_bucket": "zappa-tn4u6j73t"
  }
}
```

3. credentials

This PC > Local Disk (C:) > Users > Administrator > .aws						
	Name	Date modified	Type	Size		
is	credentials	2021/10/15 23:55	File	1 KB		

```
[default]
aws_access_key_id = AKIAQEEOWQHERVWM3N7U
aws_secret_access_key = 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
```

## 4. IAM

aws

Services ▾

Search for services, features, marketplace products, and docs

[Alt+S]

scyqa1 ▾

Global ▾

Support ▾

Identity and Access Management (IAM)

Dashboard

Access management

- User groups
- Users
- Roles
- Policies
- Identity providers
- Account settings

Access reports

- Access analyzer
- Archive rules
- Analyzers
- Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

IAM > User groups

User groups (1) Info

A user group is a collection of IAM users. Use groups to specify permissions for a collection of users.

Filter User groups by property or group name and press enter

< 1 >

⚙️

<input type="checkbox"/>	Group name	Users	Permissions	Creation time
<input type="checkbox"/>	django	1	Defined	7 hours ago

Feedback

English (US) ▾

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Privacy Policy

Terms of Use

Cookie preferences

aws

Services ▾

Search for services, features, marketplace products, and docs

[Alt+S]

scyqa1 ▾

Global ▾

Support ▾

Identity and Access Management (IAM)

Dashboard

Access management

- User groups
- Users
- Roles
- Policies
- Identity providers
- Account settings

Access reports

- Access analyzer
- Archive rules
- Analyzers
- Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

Introducing the new Users list experience

We've redesigned the Users list experience to make it easier to use. Let us know what you think.

IAM > Users

Users (1) Info

An IAM user is an Identity with long-term credentials that is used to interact with AWS in an account.

Find users by username or access key

< 1 >

⚙️

<input type="checkbox"/>	User name	Groups	Last activity	MFA	Password age	Active key age
<input type="checkbox"/>	django	django		None	i	i

Feedback

English (US) ▾

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Privacy Policy

Terms of Use


Cookie preferences

Final Result:

oakh71wulb.execute-api.us-west-1.amazonaws.com/cart

EcomStore


Hi, Could6620\_Qichen



Headphones

Add to CartView


\$179.99



Mount of Olive's Book

Add to CartView


\$14.99



Project Source Code

Add to CartView


\$19.99



Watch

Add to CartView


\$259.0



Shoes

Add to CartView

\$89.99



T-Shirt

Add to CartView

\$25.99



Hi, Could6620\_Qichen


EcomStore




Hi, Could6620\_Qichen


Continue Shopping

Items: 1Total: \$19.99

Checkout

Item	Price	Quantity	Total
 Project Source Code	\$19.99	1	\$19.99


	Item	Price	Quantity	Total
	Project Source Code	\$19.99	3  	\$59.97



Services

Search for services, features, marketplace products, and docs

[Alt+S]

 scyo1

N. California

Support

AWS Lambda

Dashboard

Applications

Functions

Additional resources

Code signing configurations

Layers

Related AWS resources

Step Functions state machines

Lambda > Functions

Functions (2)

Last fetched in 0 seconds

Actions

Create function

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Code size	Last modified
<input type="checkbox"/>	HelloWorld	-	Zip	Python 3.8	223.0 byte	18 days ago
<input type="checkbox"/>	django-dev	Zappa Deployment	Zip	Python 3.8	16.1 MB	21 minutes ago

Feedback

English (US)

© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Privacy Policy

Terms of Use

Cookie preferences