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, with1-2 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.

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.

Graphical user interface

Description automatically generated

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 in 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]

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

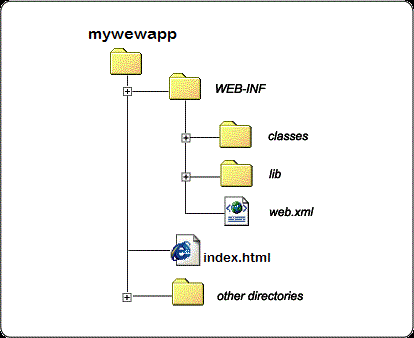
4. How are microservices different from web services? What is the relationship between microservices and serverless computing on the cloud? please explain. [10 points]

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]

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

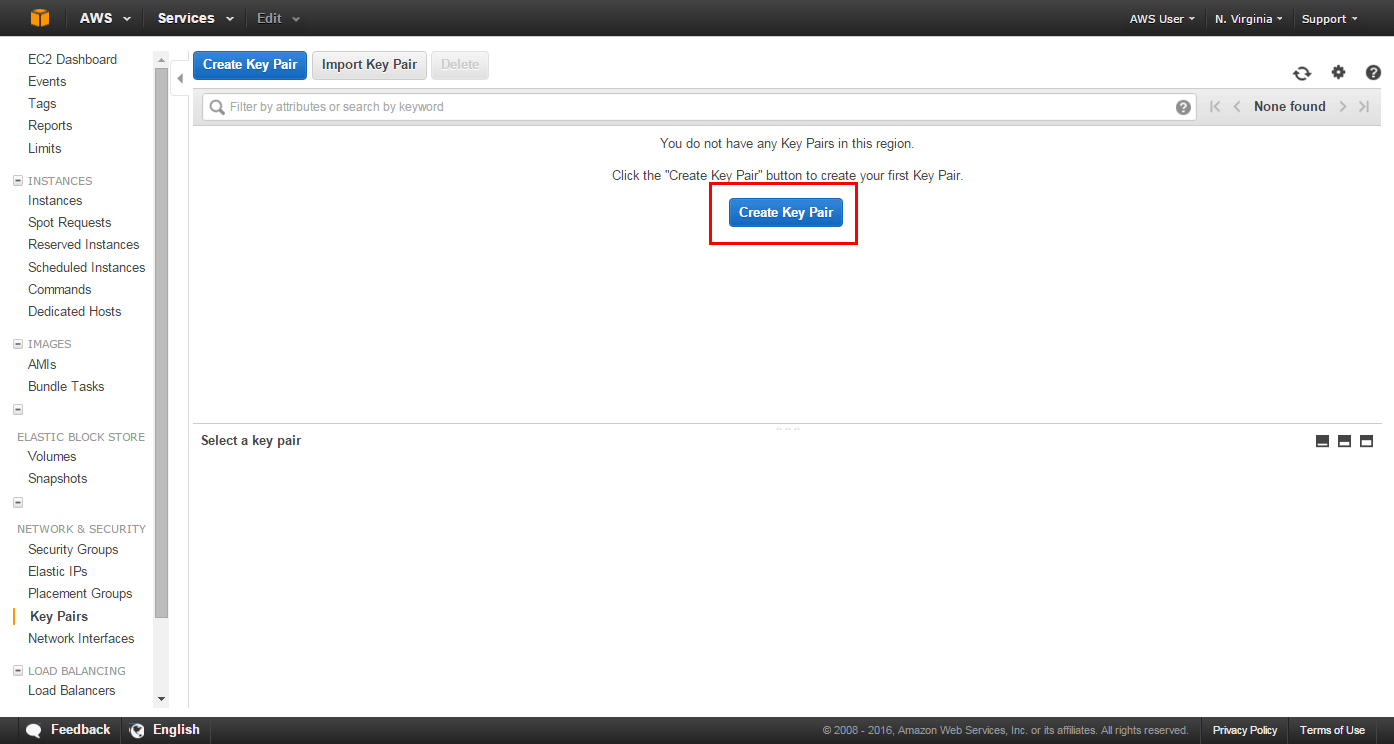


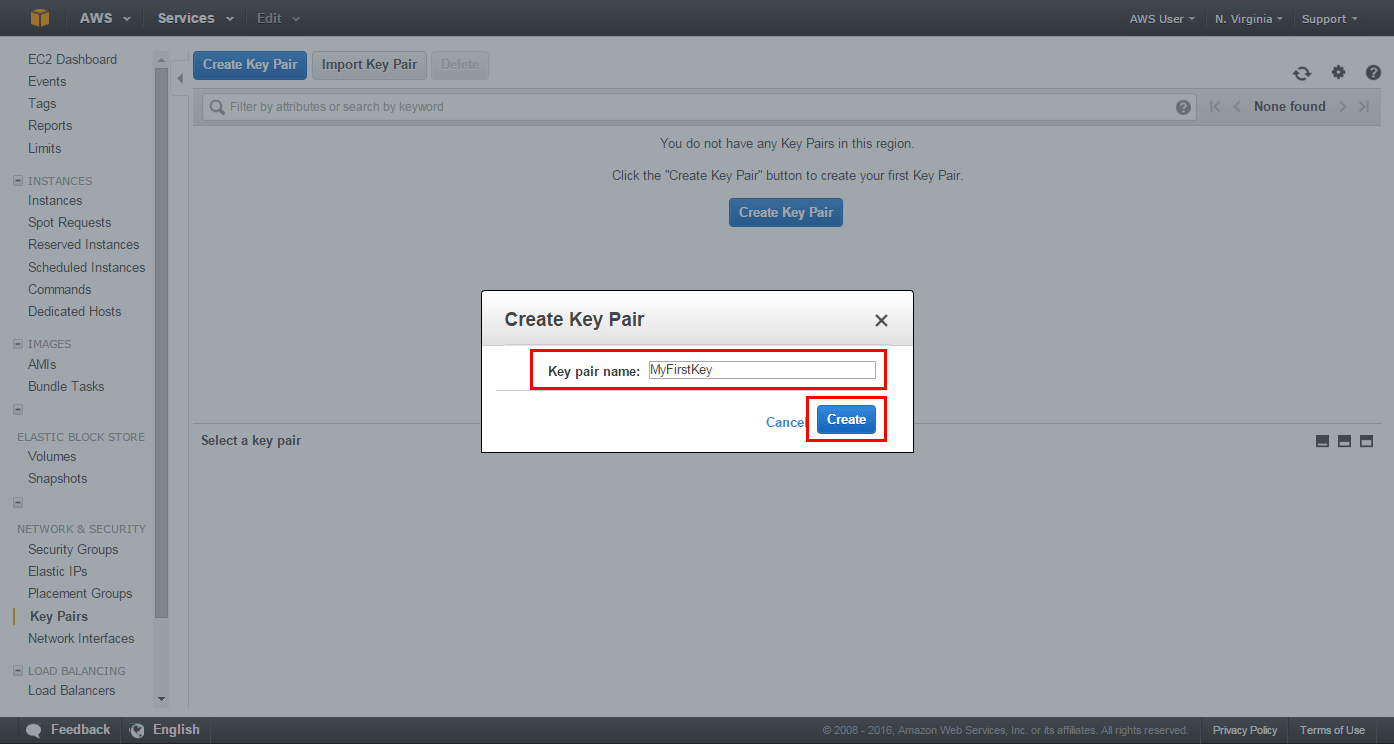
**Deploying an archive to AWS**

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.

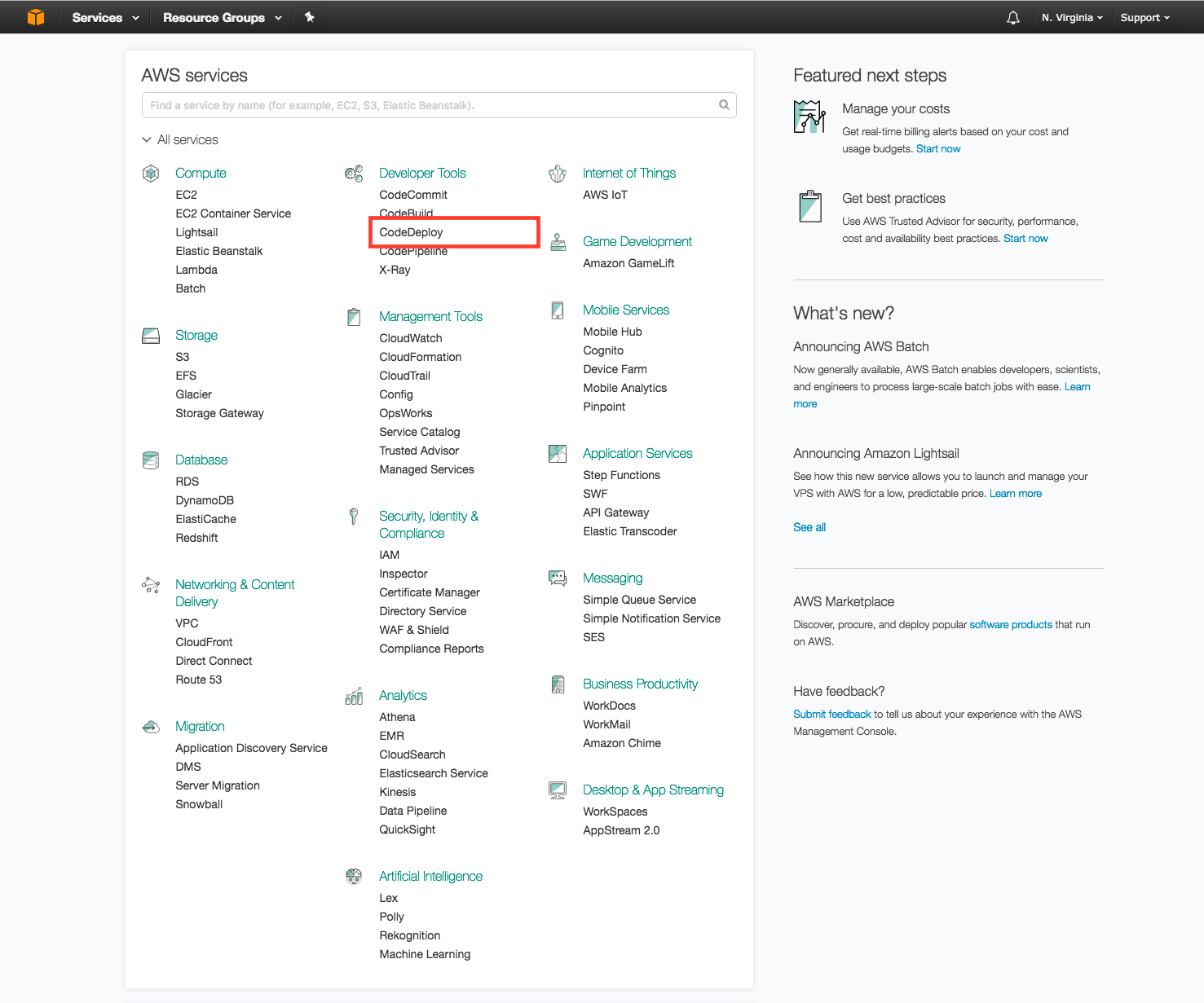
**Step 1: Create a Key Pair**

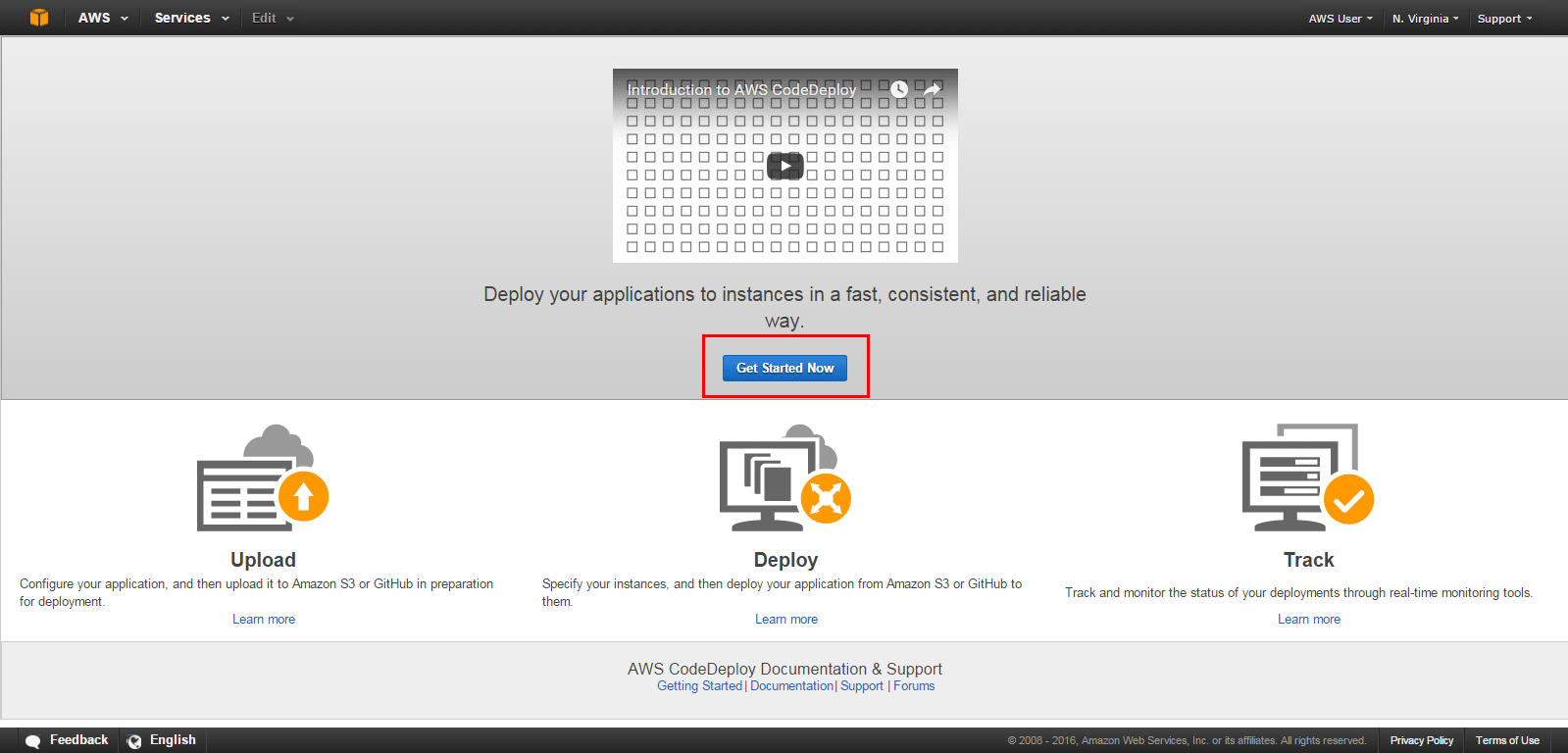
This key pair is to access the virtual machine with Amazon EC2.

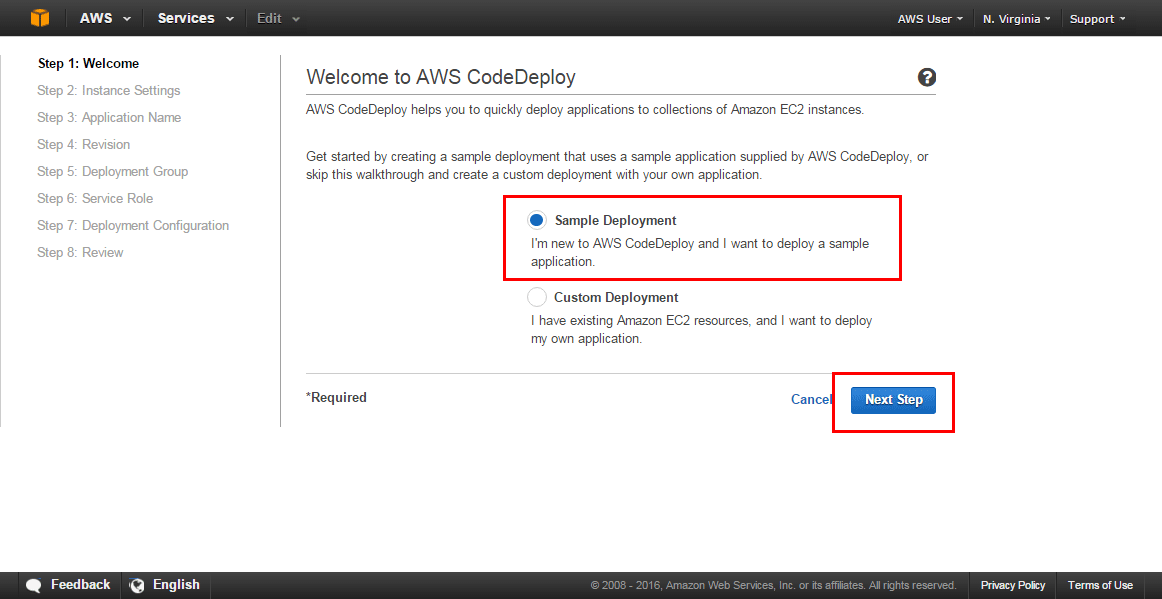




**Step 2: Enter the CodeDeploy Console**

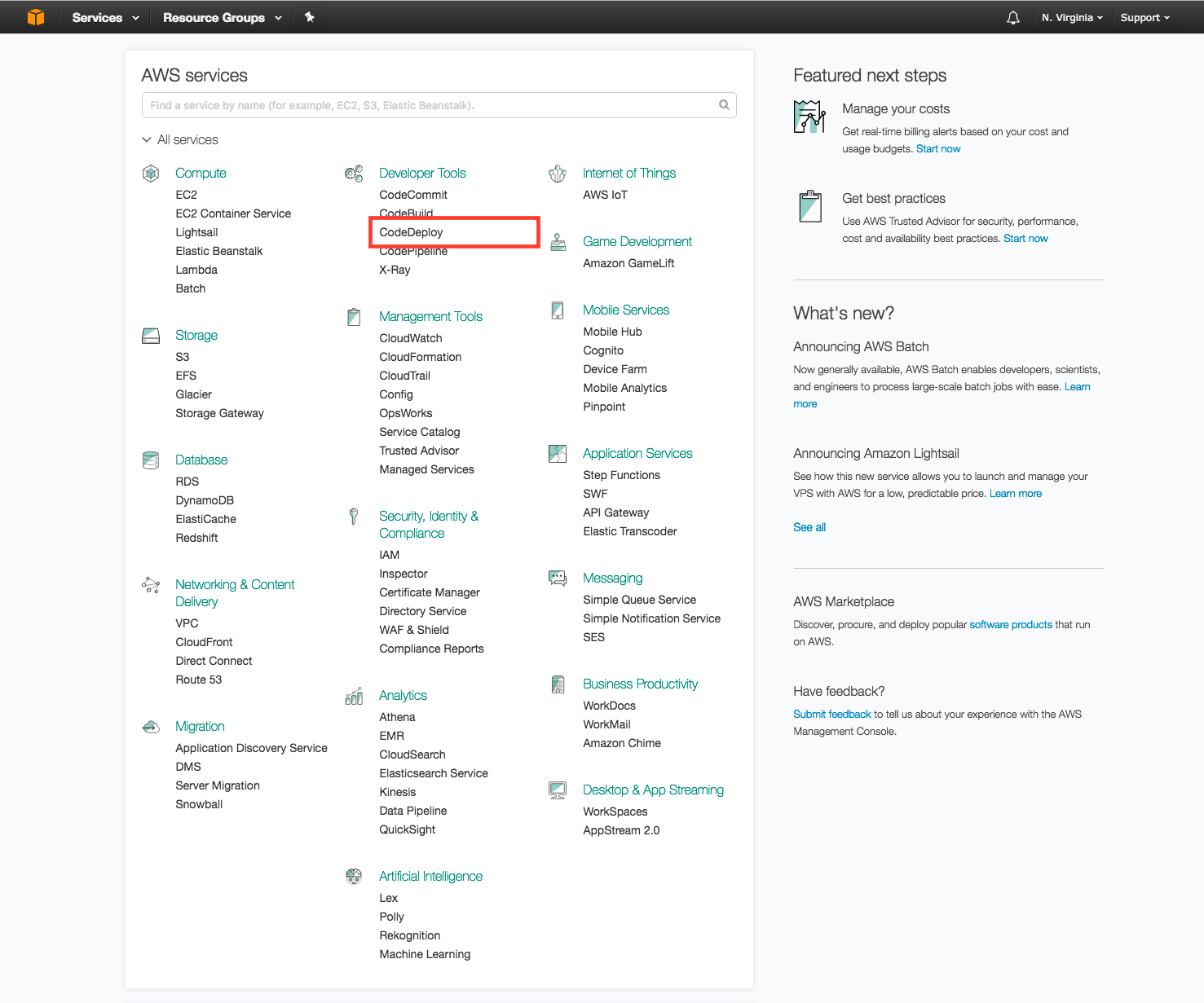


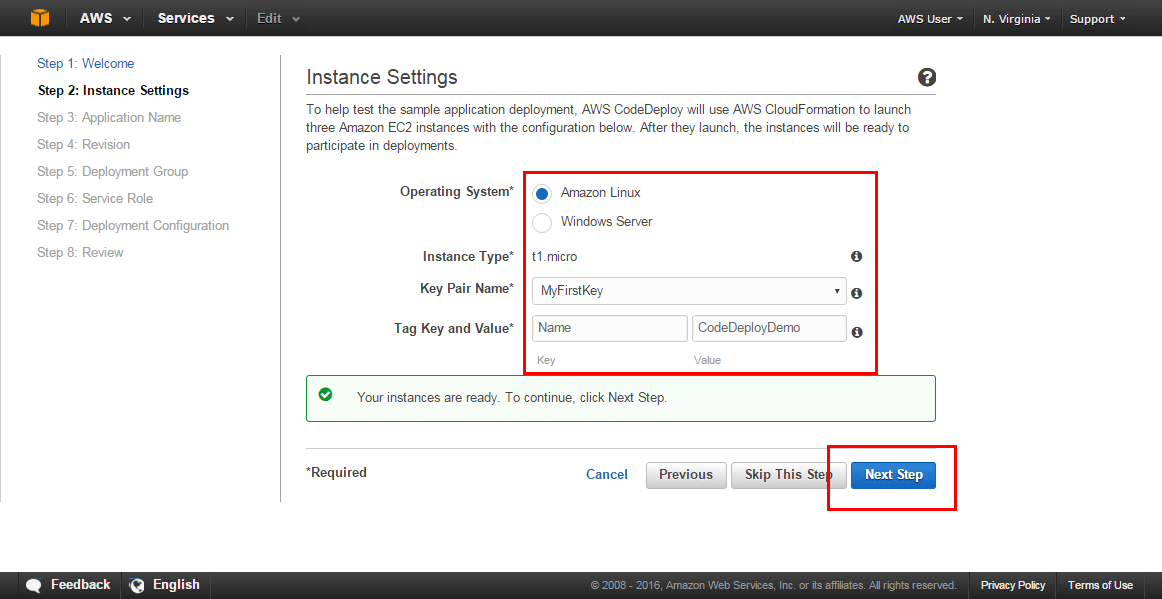




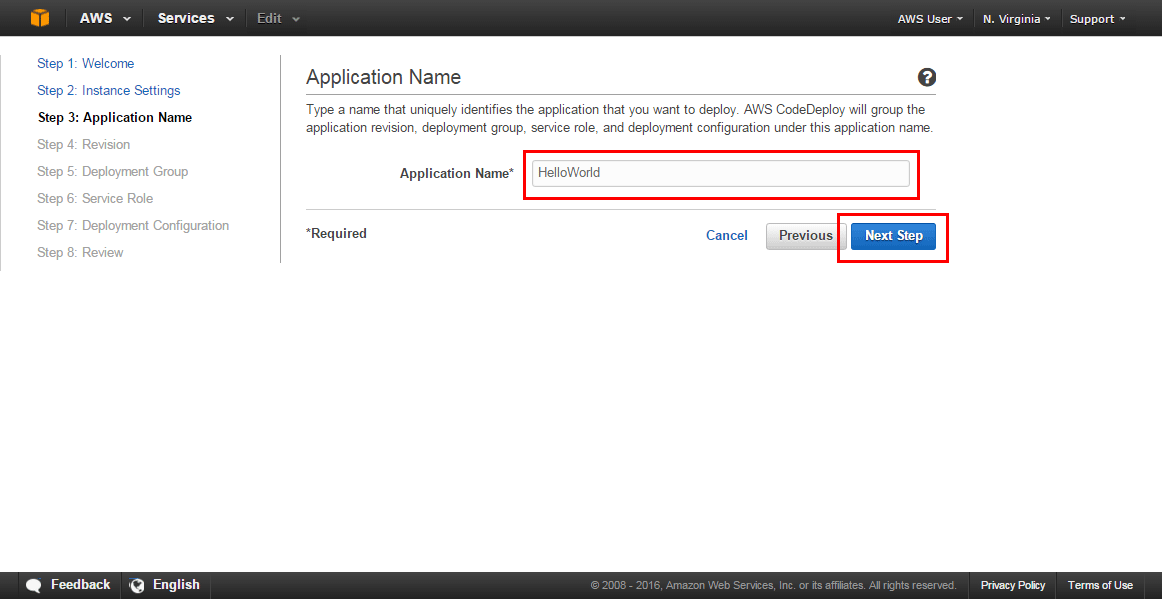
**Step 3: Launch a Virtual Machine**

Launch an AWS virtual machine to deploy my code on

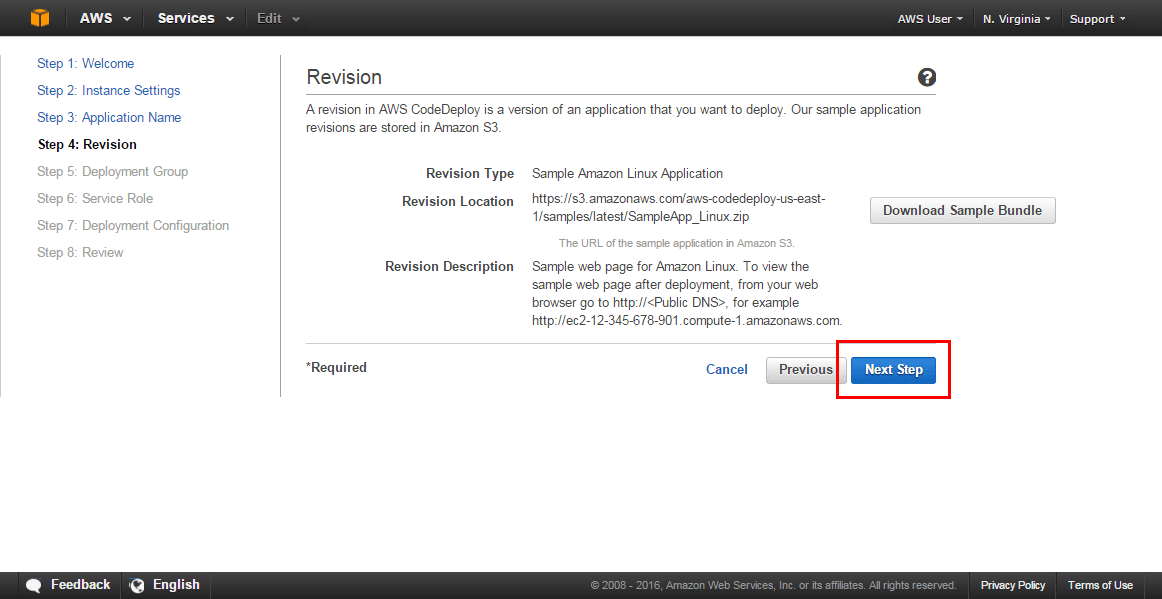




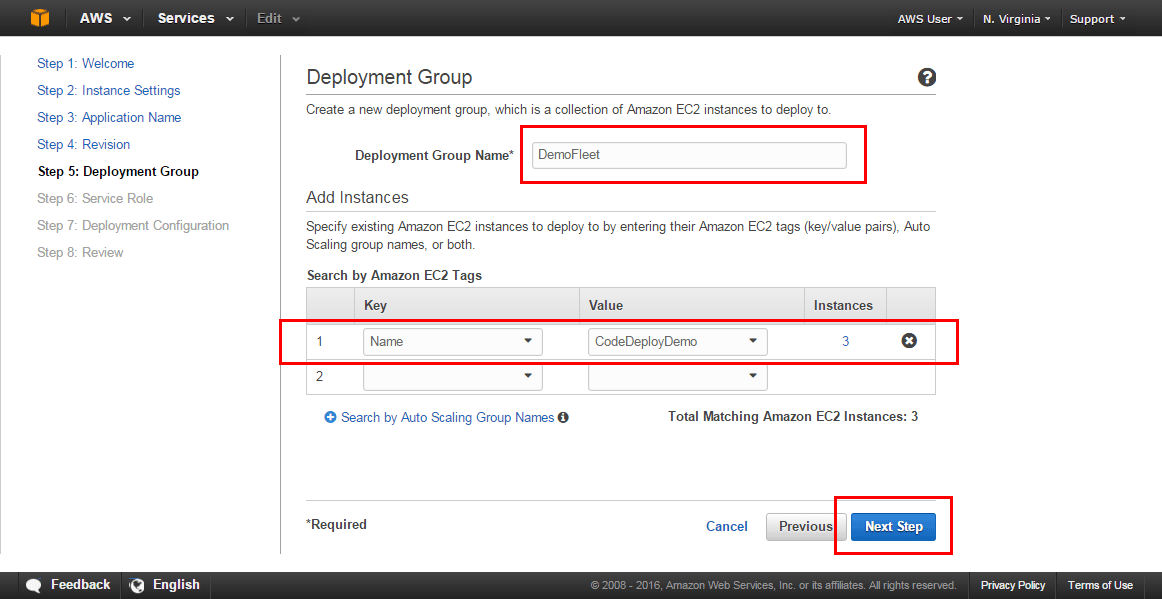
**Step 4: Name the Application and Review the Application Revision**



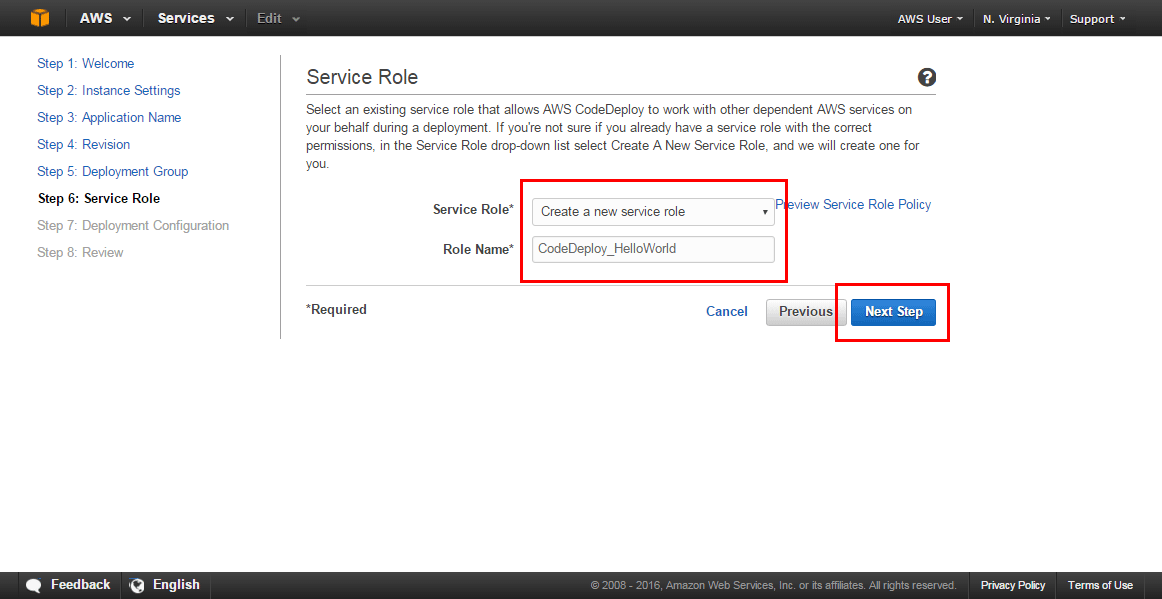
In this view, I can review information about the application revision you’d like to deploy to EC2. An application revision is an archive file containing source content, which was created before deploying.



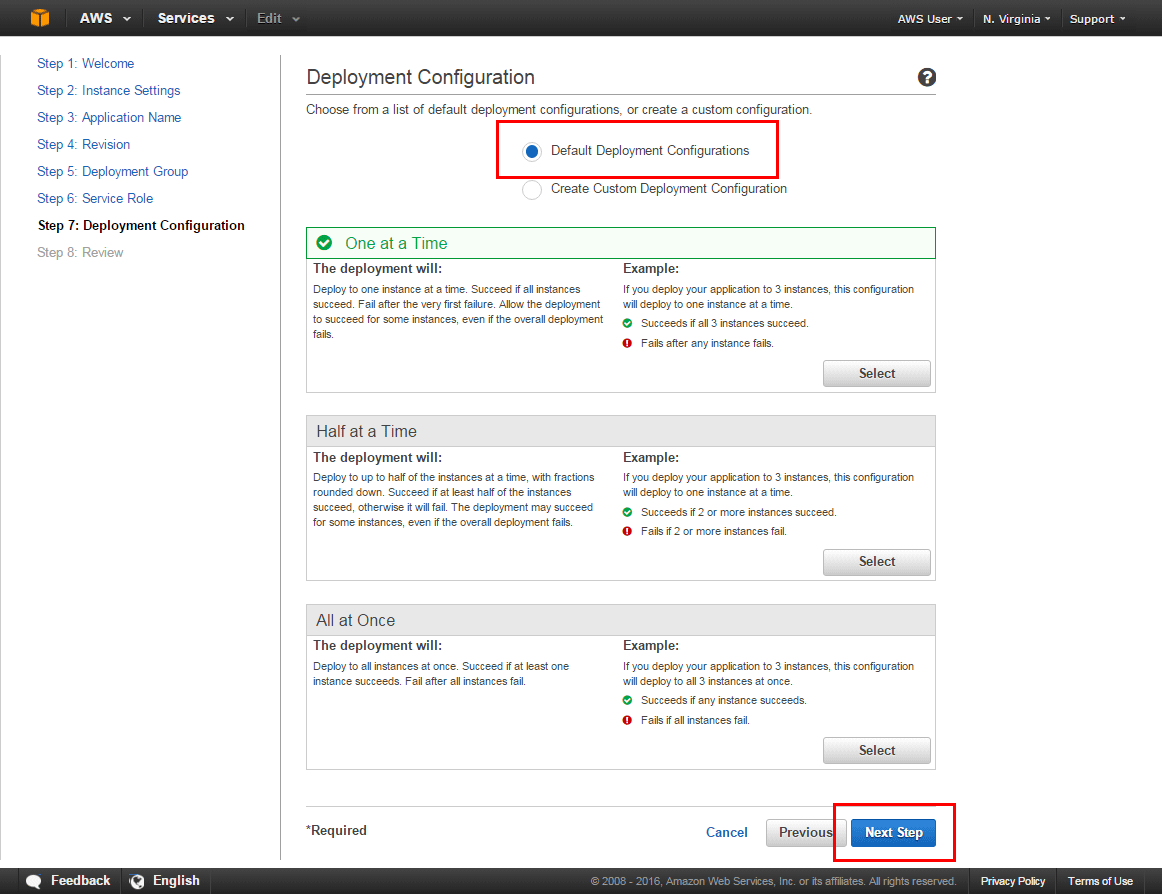
**Step 5: Create a Deployment Group**

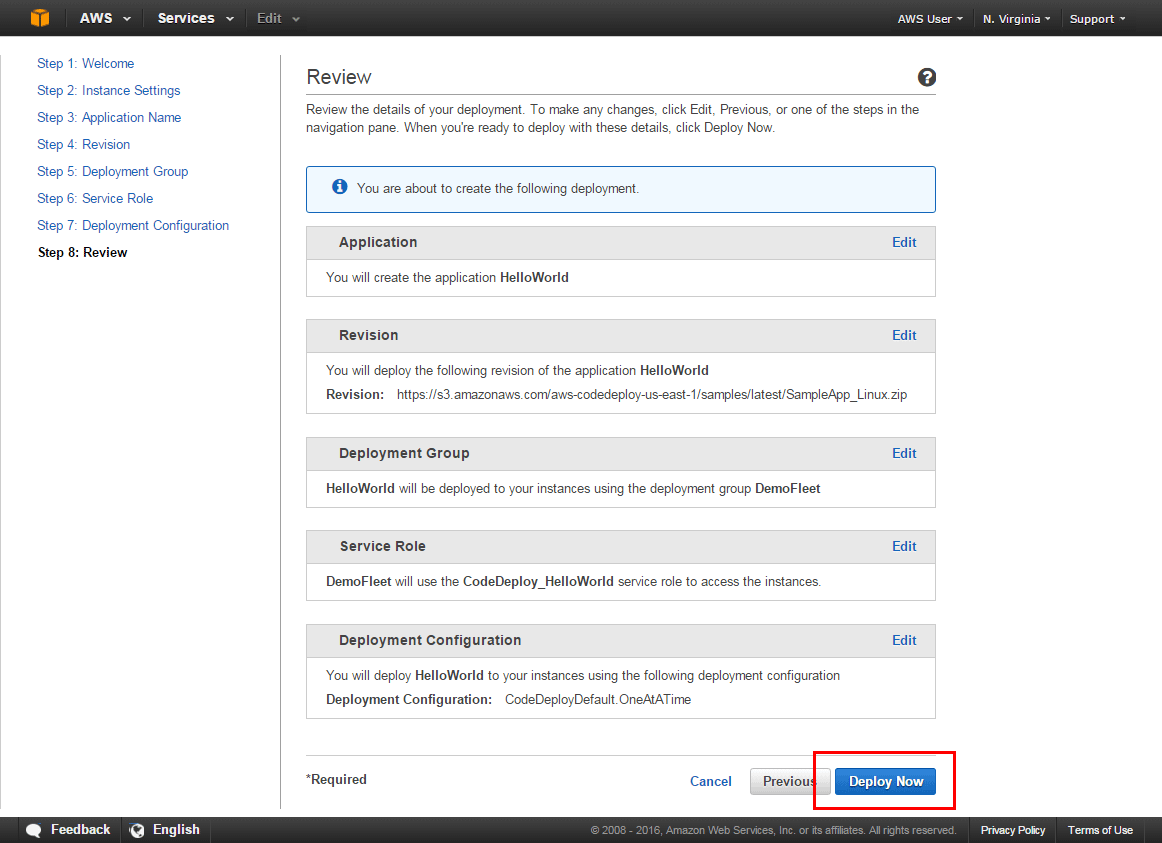


**Step 6: Create a Service Role**



**Step 7: Deploy the Application**





**Test the deployed service**