



Software Systems / Process Informatics Group
Research in Computer and Systems Engineering

Research Project Report
Comparison of Deployment Methods on AWS Cloud

June 6, 2020

Documented By: Kiran Gosavi (Matrikelnummer: 60935)

Academic Supervisor: Dr.-Ing. Detlef Streitferdt

Company Supervisor: PD Dr.-Ing. habil. Jürgen Nützel (jn@4fo.de)

Company: 4FriendsOnly.com Internet Technologies AG (www.4fo.de)

Address: Bahndamm 8, 98693, Ilmenau



4FriendsOnly.com
Internet Technologies AG

Contents

1 Acknowledgement	3
2 Introduction	4
2.1 Purpose	5
2.2 Scope	5
3 Important AWS Technologies	6
3.1 Identity Access Management(IAM) : Authentication and Authorization Service	6
3.2 Simple Storage Service (S3): Storage Service	7
3.3 Elastic Cloud Computing(EC2): Computing Service	9
3.4 CloudFormation: Management of Infrastructure	10
4 Important Deployment Methods in AWS	11
4.1 Method 1: Deployment of Node.js Server with AWS EC2	11
4.2 Docker Basics	14
4.3 Limitations of Deployment Method 1:	15
4.4 Method 2: Deployment of Tomcat Server on Kubernetes with AWS EC2 Instance	16
4.5 Method 3: Using AWS ECS for Container Orchestration	21
4.6 Method 4: Serverless Deployment with AWS Lambda	25
5 Comparison of Deployment Methods	29
6 Summary	32
7 Future Work	33
8 Declaration	33
References	34

1 Acknowledgement

The research project work presented in this document wouldn't have been possible without the kind and generous support from my academic as well as company supervisors. This work is a result of collaboration with the company "4FriendsOnly.com Internet Technologies (4FO) [1]". The "4FriendsOnly.com Internet Technologies" is a spin-off of the Fraunhofer IDMT and was founded in the year 2000. The company has been working in the field of e-commerce, mobile solutions, and cloud solutions ever since. The 4FO company is an official AWS partner.

I would like to thank Dr.-Ing. habil. Juergen Nuetzel, my company supervisor for his guidance and support throughout this research project. Dr. Nuetzel, CEO of 4FO, allowed me to work on the AWS cloud solution. This work wouldn't have been possible without his expertise and insights in the field of AWS cloud solutions. He has been working in this field for more than 10 years. I am very grateful that the 4FO company did provide me all the resources and support I required for this research project.

I am very grateful to my university professor and mentor Dr.-Ing. Detlef Streitferdt, who helped me find a suitable research topic in my field of interest. He has always supported out-of-the-box thinking and welcomed new, innovative ideas of the students. Prof. Detlef is an expert in the field of software architecture.

Last but not least, my thanks and appreciation go to my colleagues, who helped me with their expertise in this field.

2 Introduction

Embracing cloud technologies have become quite essential for organizations to sustain, compete, and drive their business in the future. Recognizing this evolving technology trend, learning and analyzing the cloud technology solution is an extremely important skill nowadays. Some cloud technologies are completely transforming the way the IT industry works and will continue to evolve more. Founded in 1997, Netflix is using cloud technologies for storage and computation. As a result of high availability and other advantages of cloud computing solutions, it has revolutionized TV and has become popular, almost eradicating cable TV watching.



Figure 1: Amazon Web Services (AWS) [2]

AWS (Amazon Web Services), founded in 2006, was the first cloud solutions provider and now is one of the dominant players in the cloud solutions market. AWS provides many cloud solutions such as Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) in the field of AI, database, machine learning, networking, storage, analytics, containerization, and serverless deployment. At present, there are around 212 AWS services in the market [3]. It is practically impossible to understand and learn all the available services. However, some of the most important and frequently used AWS services in the fields of storage, computing, security identity management, and infrastructure management, will be implemented and analyzed in this document.

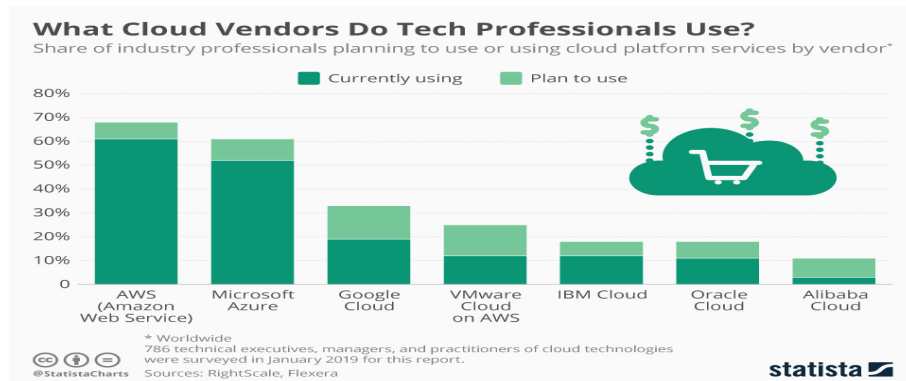


Figure 2: Amazon Web Services(AWS) in the cloud solution market [4]

The 4FriendsOnly.com Internet Technologies AG (4FO) company provides E-commerce solutions to its customers. The company mainly uses AWS for cloud solutions such as Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS). The company has plans to launch a new product named 'Xperience.Cloud' for its customers. This product will have a centralized dashboard handling all other services (repricing, recommendation etc.) provided by 4FO. We have done an analysis and comparison of different deployment strategies in collaboration with 4FO, to assess their possible future deployment strategy.

2.1 Purpose

This work serves the purpose of documenting the implementation of different AWS deployment strategies and their analysis based on levels of infrastructure task automation, scalability, availability, and learning curves. The evolving field of DevOps is the automation and integration of software development processes and tools. All DevOps infrastructure-related tasks such as server provisioning, load balancing, database provisioning, replication, and coordination and maintenance of all these components, we will implement and analyze in this document. Comparing these different strategies of AWS deployment is quite helpful for organizations in the process of decision making.

2.2 Scope

Recognizing cloud technology evolution, deployment of an application on the AWS cloud is a hot topic in DevOps. An application can be deployed in many ways on the cloud (e.g. using containerization or AWS serverless architecture), particularly as there are many new emerging technologies coming into the market every day (for instance, AWS Fargate was recently launched in December 2019). Choosing a suitable deployment method is quite essential for an organization as it has a direct impact on the cost, availability, reliability, and complexity of managing that deployment configuration and coordination. This document covers step-by-step implementation details of four different methods of deploying an application on the AWS cloud. In addition to that, it covers a comparison of these deployment methods depicting limitations and benefits of different deployment methods on AWS cloud.

3 Important AWS Technologies

AWS has a wide range of services. However, some of them are frequently used and required for all AWS deployment methods. Hence, we will begin with the frequently used basic AWS technologies.

3.1 Identity Access Management(IAM) : Authentication and Authorization Service

IAM handles users and their level of access to AWS consoles and services. It gives centralized access to your AWS account and manages the shared access. Using IAM you can manage granular access permissions and user rights. Important features behind IAM are the following:

- Identity federation: Users can use the same credentials to log in to AWS as Facebook, LinkedIn, etc.
- Multi-factor authentication: It provides multi-factor authentication using multiple devices e.g. phone, laptop, etc.
- Integrates with many AWS services.

The IAM allows access to different AWS resources. First, users need to validate with IAM using their credentials to get a token. They can get AWS credentials in exchange for this token. Once users have AWS credentials, they can access another AWS resource such as an S3 bucket or AWS Quicksight dashboard service. Following key concepts play an important role while using IAM for implementation of an AWS deployment strategy:

User: End-user of the AWS service.

Group: Set of users. Each user of the group will inherit the permissions of that group.

Policies: JSON (JavaScript Object Notation) document specifies what rights the user/-group/role has.

Role: Roles are generally assigned to AWS resources. Role grants permissions to the entities to access the AWS resources.

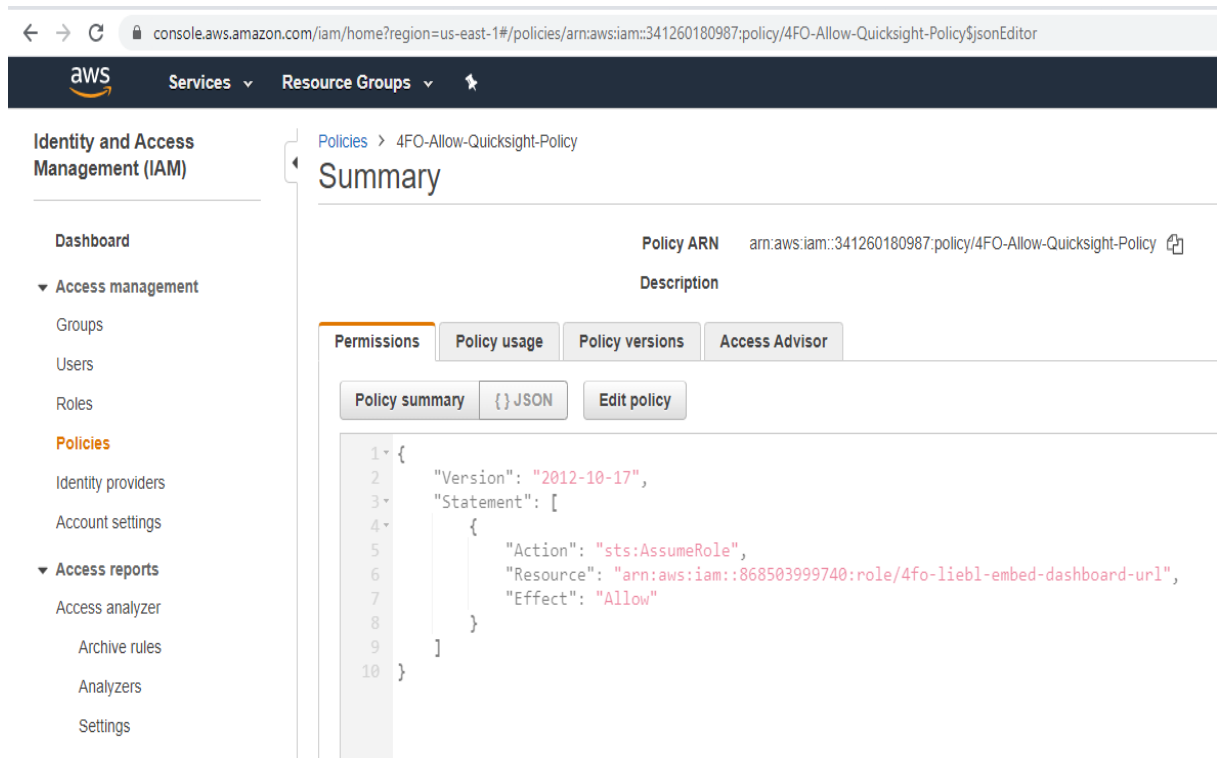


Figure 3: Snapshot of IAM policy

The above snapshot shows a policy that allows access to the given dashboard URL in the 'Resource' section. Users can use the default set of policies (e.g. Administrator) provided by AWS or can write a customized one.

3.2 Simple Storage Service (S3): Storage Service

Simple Storage Service (S3) is an AWS service, which provides simple, secure, durable, and highly scalable object storage (i.e. allows you to upload static content on the cloud). S3 is object-based cloud storage. Objects can be viewed as a file. It provides file storage up to 5 Tb. S3 stores files in a bucket, which can be viewed as a folder. Bucket names are universal and must be unique. It returns HTTP 200 in case of a successful upload of the static content to the bucket.



Figure 4: AWS S3 storage [15].

S3 objects consist of following key components :

Key : It is the name of the object that is being stored in the S3 bucket.

Value : Value is the data assigned to a particular key.

Version ID : Versioning information is maintained using Version ID.

Metadata : Data about data is stored as metadata.

Subresources : Subresources like ACL (Access Control List), which controls access to the S3 objects.

How data consistency is achieved? S3 has achieved data consistency with the two rules. First, 'read after write' rule for operation PUTS is followed if new objects are added. It means that in case of the creation of a new object in S3, it needs to be replicated across all the data centers located in different regions. The object key is not listed or visible until the replication across all the regions is successful. Second, the 'eventual consistency' rule is followed by overwrite PUTS and DELETE operations. It means that in the event of deletion or update, some data centers may show old object values as they will be updated eventually.

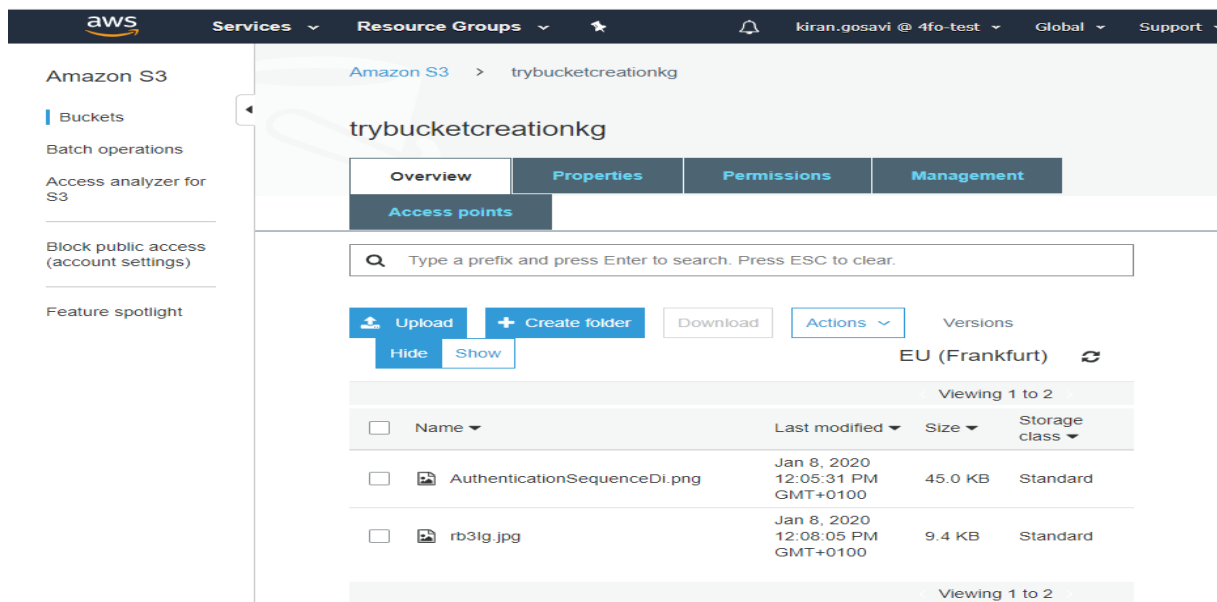


Figure 5: Snapshot of objects in S3 bucket

3.3 Elastic Cloud Computing(EC2): Computing Service

Elastic Compute Cloud (EC2) is an AWS service that provides customizable computing capacity in the cloud. It eliminates the time required to obtain and boot physical servers and to maintain them, allowing you to scale capacity faster as your computing requirements change. We will use EC2 instances in deployment strategies to deploy an application server. To allow access to that application server from the outside world, a key concept of security group is required.

Security Groups: Security groups have similar behavior as a firewall for an application server running inside an EC2 instance. All inbound traffic to the application server running inside an EC2 instance is blocked by default. However, all outbound traffic from an application server to the outside world is allowed. Security groups help to open ports for applications running in EC2 instances. If users change any rules of a security group, they will be effective immediately. Users can attach multiple EC2 instances to a single security group. Security groups are stateless i.e. if user open port 80 it will be open for all inbound and outbound traffic requests.

IAM Role in EC2: Users can attach an IAM role to EC2 instances. The IAM role allows the sharing of EC2 resources amongst the users with different AWS accounts. In addition to that, it controls each user's permissions. In case of authentication using access key and secret access key, users need to store their private key on an EC2 Linux instance. This can be easily hacked by other users and can be decrypted to get all login information. In addition to that, recovery is not possible in case of loss of the private key. However, you can connect to an EC2 Linux instance by following steps provided by AWS in case of private key loss. Thus, an IAM role can be created with administrator access and attached to an EC2 Linux instance. This is more secure than using EC2 with private and public-key cryptography.

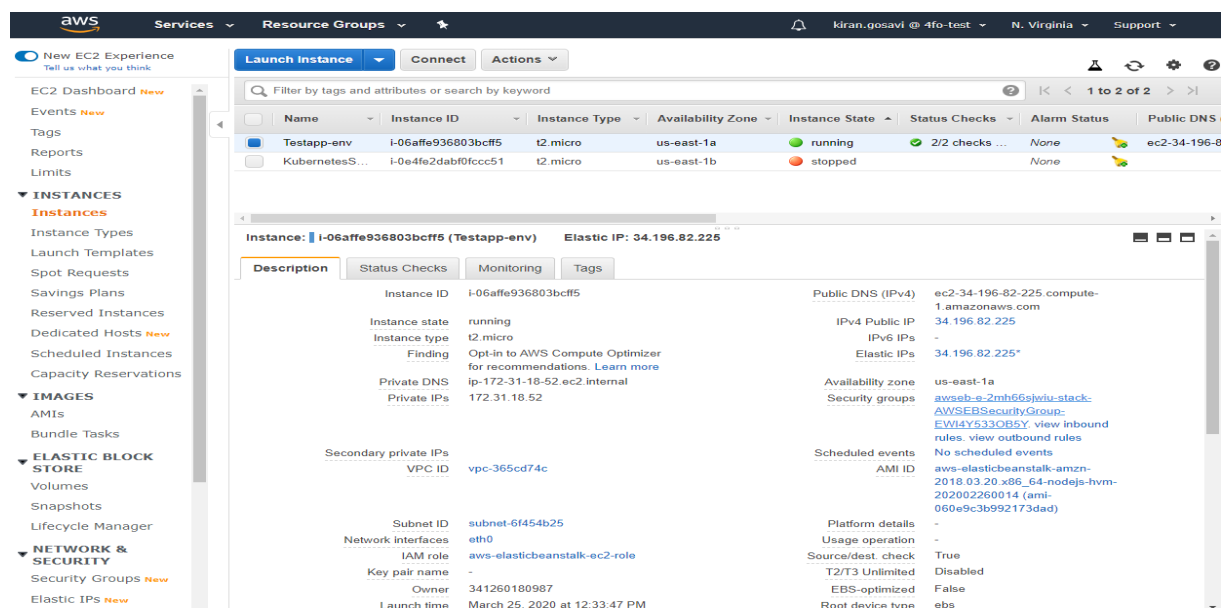


Figure 6: Snapshot of Testapp-env EC2 instance with all configuration details.

3.4 CloudFormation: Management of Infrastructure

CloudFormation is an AWS service that automates the deployment of your cloud environment using scripting languages JSON (JavaScript Object Notation) or YAML. Quick-start templates are available which are designed by AWS solution architects which allow users to efficiently build a complex environment. Features of CloudFormation are listed below:

- **Infrastructure replication:** In the case of infrastructure failure, the user needs to recreate all the resources in the infrastructure manually. As a result, this process requires more time than the time required to replicate infrastructure exercising CloudFormation.
- **Tracking and management:** Applying CloudFormation YAML documents, managing complex infrastructure becomes convenient and manageable.

Stack: A stack is a single unit made up of different resources. By creating an updating stack you can manage all your resources. For example, to create a WordPress blog we will need a stack with resources like WordPress database, web server, elastic load balancer.

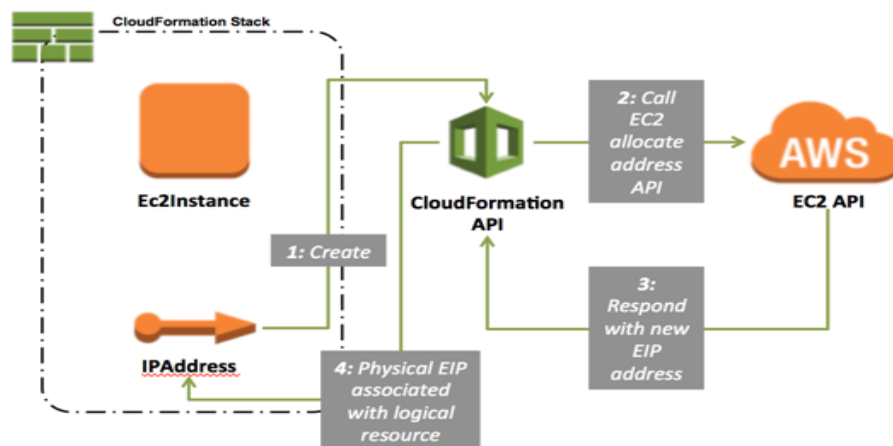


Figure 7: Working of a CloudFormation with AWS EC2 [12].

Template: It is a text file in YAML or JSON format, which is responsible for configuring and provisioning your stack resources. It describes the resources.

```

1 AWSTemplateFormatVersion : "version date"
2 Description                : string
3 Metadata                  : template metadata
4 Parameters                 : set of parameters
5 Mappings                  : set of mappings
6 Conditions                 : set of conditions
7 Resources                  : set of resources
8 Outputs                    : set of outputs
  
```

Listing 1: AWS YAML template

4 Important Deployment Methods in AWS

4.1 Method 1: Deployment of Node.js Server with AWS EC2

The Xperiance.Cloud project uses AWS services for deployment. AWS Elastic Cloud Computing (EC2) provides scalable computing on the cloud. The Xperiance.Cloud application server is built using Node.js. Node.js is an open-source language, generally used to develop a web application.

Deployment Details The Xperiance.Cloud console is deployed on AWS cloud services. The Xperiance.Cloud uses EC2 (Elastic Compute Cloud) AWS service for frontend and backend deployment. Users can start an EC2 instance when a computation is required and can stop/pause an instance afterwards. They will be charged only for the time the instance is active, thus making it cost-efficient compared to traditional physical server deployment. Authentication in EC2 instance is done using public-key cryptography or using an IAM role. It uses a 2048-bit SSH-2 RSA with public and private keys to log in. Steps to create EC2 instance are as below:

- Select a pre-configured machine template with desired AMI (Amazon Machine Image).
- Choose instance type with desired capacity and computing power.
- Optional step: users can configure the EC2 instance in additional security settings by choosing the IAM role and network access.
- Launch an instance.

Strapi: The Xperiance.Cloud console will use Strapi as a Content Management System (CMS). Strapi is headless CMS. Content Management System allows users with minimal technical expertise to add, modify, and delete content from a website. CMS has two major parts: Interactive UI and Content Delivery Application (CDA). CDA compiles and updates the content on a website that has been updated or added by the CMS user via UI.

After the launch of an EC2 instance, you can deploy your Strapi application in that instance. For monitoring this Node.js server, we have used the PM2 monitoring tool.

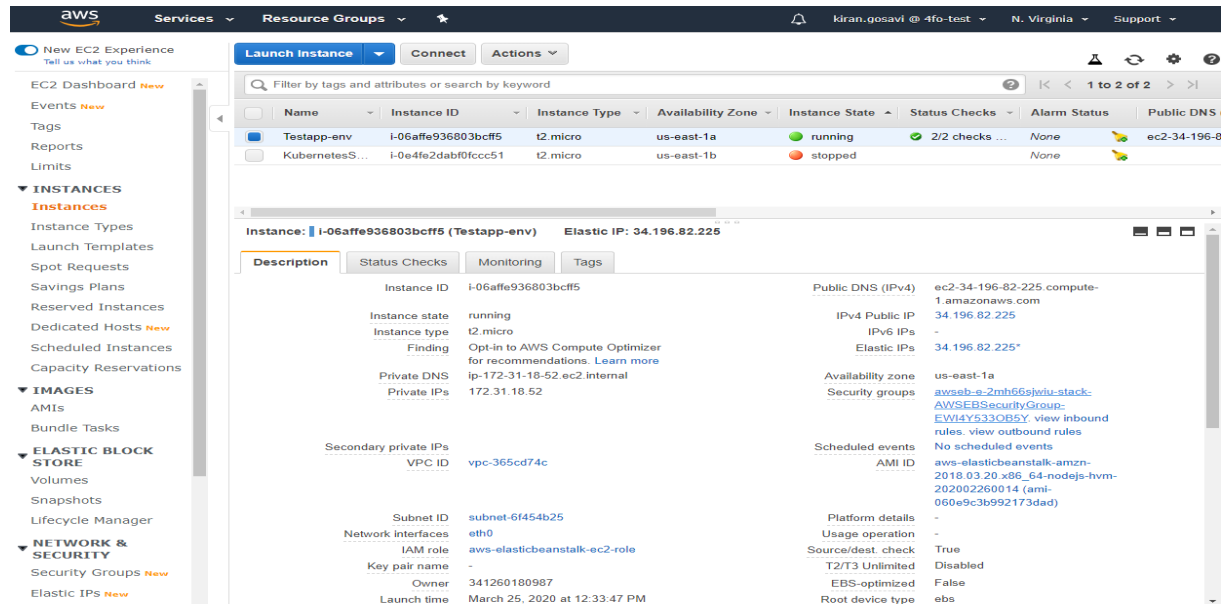


Figure 8: Snapshot of EC2 instance with all configuration details used in deployment method 1.

PM2 monitoring: PM2 is a daemon process application that will keep your application running on an EC2 instance. PM2 automatically starts the application if it stops for some reason. Following commands are used to start the Strapi server:

- Systemctl start pm2-strapi
- Systemctl stop pm2-strapi
- Systemctl status pm2-strapi

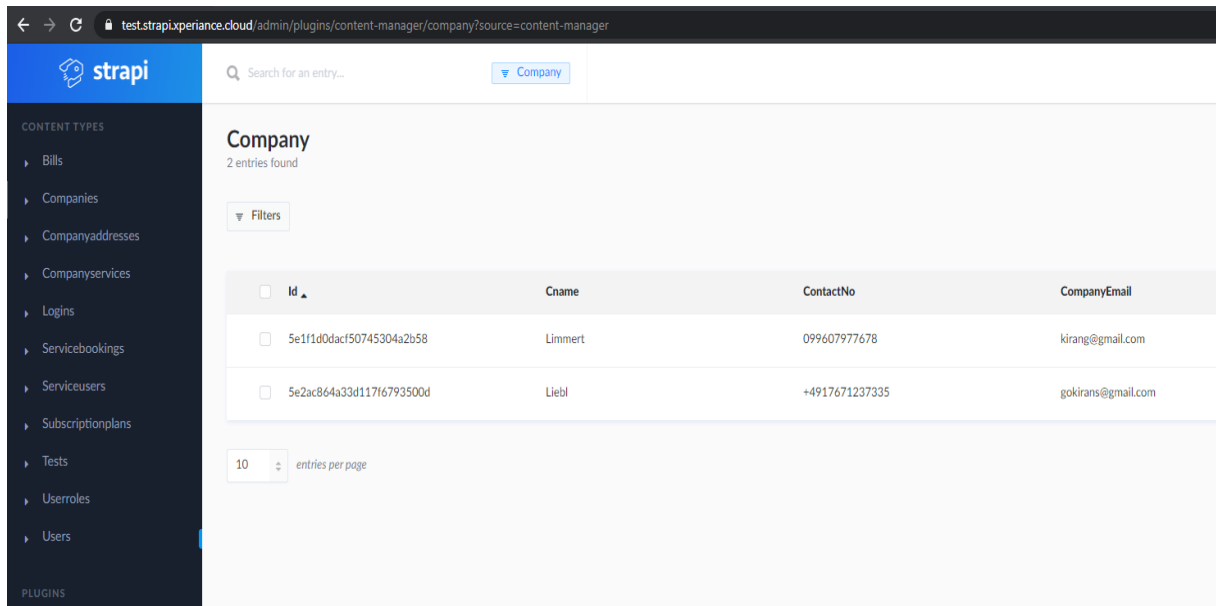


Figure 10: Snapshot of Strapi UI(User Interface) accessible via registered domain.

4.2 Docker Basics

Docker is one of the leading vendors in containerization technology. It was developed by Docker Inc in 2013. Docker helps the packaging and running of an application in an isolated environment called a container. The traditional deployment method involves installing all the required software dependencies and applications separately, in addition to the actual application code. With the help of docker users can reduce the deployment time as everything is bundled into a single unit. To understand container deployment on the AWS cloud, basic docker concepts are quite important. Following are basic terminologies:

Docker daemon: It is the host Operating System that runs docker.

Docker client: A way to connect or interact with the docker engine. Docker clients send the actual docker command such as docker run to the Docker daemon.

Docker image: It is a read-only template to run the Docker container .

Docker registry: Registry is a repository which stores Docker images. Docker hub is a public registry that anyone can use to pull or push the images.

Docker container: A runnable instance of an image is called a container. A container can be viewed as an isolated environment for running your application.

Docker hub: Public repository for storage and retrieval of docker images.

Command : `docker run image-name [options]`
 Command to build a docker image: `docker build -t imagename:tagname dir`

```

[...]/error writing to container: console disabled
root@ip-172-31-43-13:~# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

root@ip-172-31-43-13:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
de8b666f6dd8   hello-world    "/hello"                13 seconds ago    Exited (0) 12 seconds ago           eager_austin
424fa3979d66   hello-world    "-i -t /bin/bash"       31 seconds ago    Created                                eager_margulis
c60d838c27f1   hello-world    "-it /bin/bash"         About a minute ago    Created                                crazy_shamir
5ebcb5743405   gcr.io/hello-minikube-zero-install/hello-node    "/bin/sh -c 'node se..."  9 days ago      Up 9 days                                k8s_hello-node_he
22789273540a_0   k8s.gcr.io/pause:3.2    "/pause"                9 days ago      Up 9 days                                k8s_POD_hello-nod
35d0a_0
  
```

Figure 11: Snapshot of basic docker commands and its output.

4.3 Limitations of Deployment Method 1:

The deployment method of the Node.js server with an EC2 instance is simple and suitable for applications with a limited number of users. However, if the application is evolving and needs additional resources, then the following problems could arise:

- **Scalability:** Scaling this deployment strategy is quite complex and time-consuming since it requires users to manually add load balancers, replication, etc components and monitor them.
- **Complex management and monitoring:** Since there is no central monitoring of infrastructure, users will need to monitor and manage each component and tasks related to it such as the health of a DNS server or load balancer. Users may need to add a new load balancer if the load increases. Users will need to replicate complete infrastructure in case of failure of infrastructure.
- **Complex rollout and deployment:** Deployment of new versions of applications is further complicated with such traditional infrastructure because it requires manual deployment on each infrastructure component individually.
- **Time consumption:** The software lifecycle process with the traditional deployment method is quite time-consuming because of complex management, monitoring, rollout, and scaling plus not suitable for rapid development in agile processes.

4.4 Method 2: Deployment of Tomcat Server on Kubernetes with AWS EC2 Instance

Kubernetes: Kubernetes (commonly known as k8s [10]) is an open-source container orchestration technology developed by Google in 2014. Orchestration of a container is automating co-ordination, co-operation, and management of many containers that form a single unit of an application. To understand Kubernetes' deployment on AWS, we will learn Kubernetes basics first.

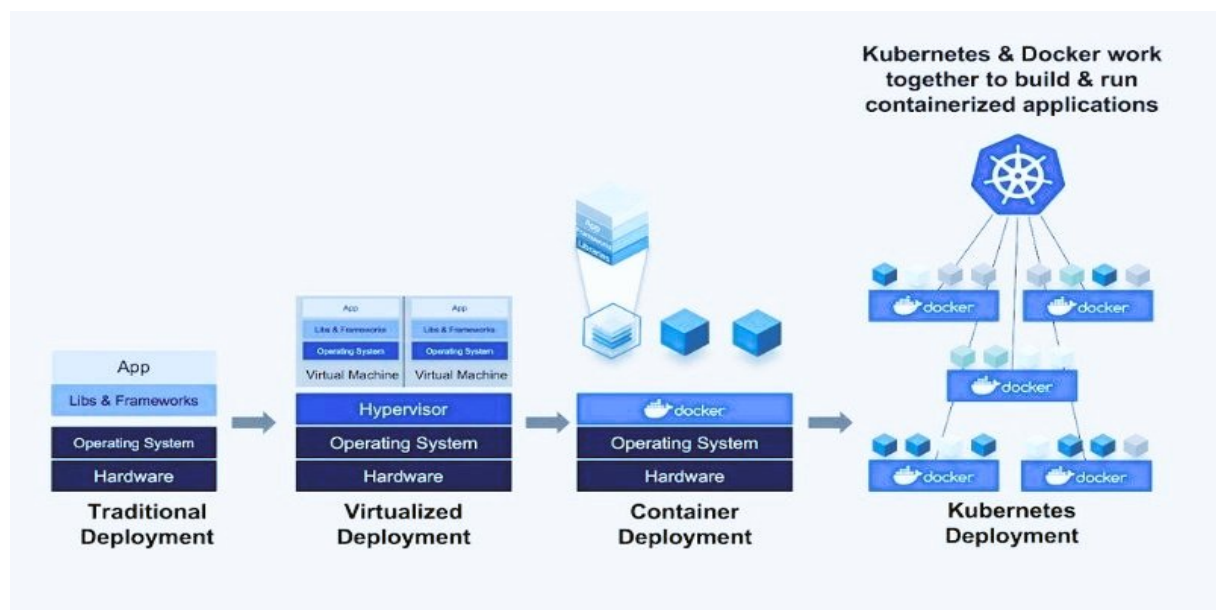


Figure 12: Evolution of Containerization technology [5]

Traditional deployment: Organizations used to purchase expensive physical machines and servers for the deployment of an application. Due to the absence of automation managing and maintaining a large fleet of physical servers in a traditional deployment was difficult. This deployment strategy has many issues such as scalability, availability, and cost.

Virtualization: Virtualization allows you to run multiple VM (Virtual Machine) on a single physical machine. It provides isolation between applications running on different VMs. It uses physical resources better and reduces cost as compared to traditional deployment.

Container deployment: Containers provide ways to isolate applications using underlying OS's basic services. The difference between containers and VMs is that guest OS is absent in containers and they contain mostly application code. They run only the necessary processes that the container is started for. It is cost-efficient and maximizes the use of resources.

	Container	Container Orchestration
Functions	Keeps software in a clean isolated view of an OS	Defines the relationship between containers, where they come from, how they scale, network details etc.
Alternatives	VM, Physical machine	Homegrown scripts, manual setup
Vendors	Docker, RKT, Garden	Kubernetes, Docker swarm, Amazon EKS, MESOS

Kubernetes features are listed below:

- **Load balancing:** Kubernetes can provide load balancers for your application, which will distribute the load across multiple servers in your infrastructure.
- **Service discovery:** Kubernetes exposes containers with DNS or IP addresses to the outside world.
- **Convenient rollout and rollback:** You can describe the desired state of your deployment, then Kubernetes can automatically change the actual state to the desired state in case of any problems in the infrastructure.
- **Management of configuration:** Kubernetes lets you manage SSH key, password, etc. conveniently.

Container orchestration is a comparatively new space than containerization technology. Kubernetes is considered as a standard for container orchestration [11]. Kubernetes setup types are as follows:

- **Evaluation / training purpose:** minikube (local version of k8s)
- **Development:** minikube, development cluster
- **Deployment:** cloud provider, bare metal

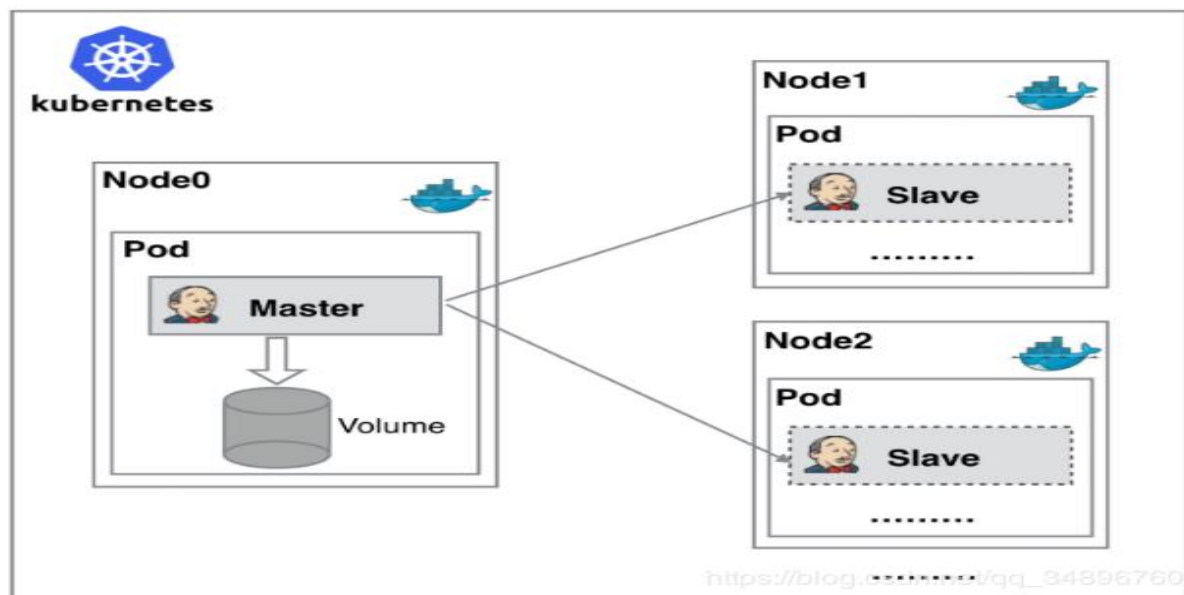


Figure 13: Kubernetes cluster example.[6]

Minikube: It is all in one install for Kubernetes. It takes all the distributed components of Kubernetes and packages them as a single virtual machine to run locally. However, minikube does not provide cloud-specific features like load balancers, persistent volumes, Ingress, etc. Minikube requires virtualization on Linux.

Deployments: Kubernetes deployments are high-level constructs that define an application. They are defined as a collection of resources and references. Typically described in YAML/JSON format.

Pods: It is the basic execution unit in the Kubernetes cluster. It is an instance of a container within a deployment. A pod can have one or many containers running inside.

The following command lists all existing pods: `kubectl get pods`

This command provides details for a given pod: `kubectl describe [pod-name]`

Now, let's see how users can deploy the tomcat server on Kubernetes, which is directly running on an EC2 instance. Following are the steps of deployment:

Step 1: Define the deployment using a YAML file. The 'cat deployment' command lists the contents of the deployment YAML file. After defining a deployment in a YAML file, the below command is used to create the deployment in a cluster.

Command: `kubectl apply -f ./deployment.yaml`

```
root@ip-172-31-43-13:/home/ubuntu# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-minikube                      1/1     Running   0           10d
hello-node-677b9cfc6b-zb4r8        1/1     Running   0           9d
tomcat-deployment-754dd4f6fc-hpmhh 1/1     Running   0           30m
root@ip-172-31-43-13:/home/ubuntu# kubectl describe pods tomcat-deployment-754dd4f6fc-hpmhh
Name:          tomcat-deployment-754dd4f6fc-hpmhh
Namespace:     default
Priority:       0
Node:          ip-172-31-43-13/172.31.43.13
Start Time:    Tue, 14 Apr 2020 11:46:39 +0000
Labels:        app=tomcat
               pod-template-hash=754dd4f6fc
Annotations:   <none>
Status:        Running
IP:            172.17.0.6
IPs:
  IP:          172.17.0.6
Controlled By: ReplicaSet/tomcat-deployment-754dd4f6fc
Containers:
  tomcat:
    Container ID:  docker://e5a469922b8c601ffc239d8d52afa936d52d5d0f0c5f60115deb195b630e5eb5
    Image:         tomcat:9.0
    Image ID:      docker-pullable://tomcat@sha256:2254679f4958efe7c9810d6a66d449f58b588ecd253e83e9d3afb68c51637cf4
    Port:         8080/TCP
    Host Port:    0/TCP
    State:        Running
      Started:    Tue, 14 Apr 2020 11:46:57 +0000
```

Figure 14: Snapshot of the Kubernetes ‘get pod’ command and its output.

```
root@ip-172-31-43-13:/home/ubuntu# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  selector:
    matchLabels:
      app: tomcat
  replicas: 1
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
        - name: tomcat
          image: tomcat:9.0
          ports:
            - containerPort: 8080
root@ip-172-31-43-13:/home/ubuntu#
root@ip-172-31-43-13:/home/ubuntu# kubectl apply -f deployment.yaml
deployment.apps/tomcat-deployment unchanged
root@ip-172-31-43-13:/home/ubuntu#
root@ip-172-31-43-13:/home/ubuntu#
```

Figure 15: Snapshot of the Kubernetes ‘apply’ command and its outcome.

Step 2: Expose service running inside a container to the outside world on a particular port using the following command:

Command: `expose deployment tomcat-deployment --port=80 --target-port=8000`

```

root@ip-172-31-43-13:/home/ubuntu# kubectl expose deployment nginx --port=80 --target-port=8000
service/nginx exposed
root@ip-172-31-43-13:/home/ubuntu# kubectl get services

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-node	LoadBalancer	10.103.215.205	<pending>	8080:32466/TCP	19d
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	19d
nginx	ClusterIP	10.110.155.81	<none>	80/TCP	7s
tomcat-deployment	NodePort	10.109.133.18	<none>	8080:32163/TCP	9d

```

root@ip-172-31-43-13:/home/ubuntu#
root@ip-172-31-43-13:/home/ubuntu#

```

Figure 16: Snapshot of the Kubernetes 'expose' command and its outcome.

Step 3: After exposing the application service, get the full URL to access the service running inside a container using the following command:
 Command: `minikube service tomcat-deployment --url`

Step 4: To execute any command inside a container or to check running users can

```

Run 'minikube --help' for usage.
root@ip-172-31-43-13:/home/ubuntu# minikube service tomcat-deployment --url
http://172.31.43.13:32163
root@ip-172-31-43-13:/home/ubuntu# curl http://172.31.43.13:32163
<!doctype html><html lang="en"><head><title>HTTP Status 404 - Not Found</title><s

```

Figure 17: Snapshot of the Kubernetes 'getURL' command and its outcome.

use `exec` command: `kubectl exec [-it] pod name bash`

Services: It is a way to expose application ports to the outside world. Command:

```

See 'kubectl exec --help' for usage.
root@ip-172-31-43-13:/home/ubuntu# kubectl exec -it tomcat-deployment-754dd4f6fc-hpmhh bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
root@tomcat-deployment-754dd4f6fc-hpmhh:/usr/local/tomcat# whoami
root
root@tomcat-deployment-754dd4f6fc-hpmhh:/usr/local/tomcat# top
top - 12:31:57 up 20 days, 4:19, 0 users, load average: 0.06, 0.07, 0.07
Tasks: 3 total, 1 running, 2 sleeping, 0 stopped, 0 zombie

```

Figure 18: Snapshot of the Kubernetes 'exec' command and its output.

`kubectl expose deployment tomcat-deployment --type=NodePort`

Kubectl RUN

It is a command used to run a particular image in a cluster.

`kubectl run name --image =imageName`

4.5 Method 3: Using AWS ECS for Container Orchestration

Amazon Elastic Container Services (ECS) is a container orchestration technology. ECS lets you manage container-based applications using simple API calls. ECS allows you to manage containers, scale them in the AWS cloud environment. Users can scale their clusters based on the requirements. Users can run ECS clusters across multiple availability zones within a region. Users can use other AWS services like Amazon VPC (Virtual Private Cloud) easily with ECS i.e. users can create their cluster within an existing VPC or can create a new one.

Task Definition: To run an application with ECS, the user must create a task definition. It is a simple text file of type JSON or YAML which defines your deployment. It is similar to deployment in Kubernetes. In the following task, the definition creates a webserver with an “nginx” container image and AWS VPC network mode.

```
{
  "family": "webserver",
  "containerDefinitions": [
    {
      "name": "web",
      "image": "nginx",
      "memory": "100",
      "cpu": "99"
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "memory": "512",
  "cpu": "256",
}
```

Figure 19: Snapshot of AWS task definition.

Task: The instantiation of a task definition is called a task within a cluster. One can define many tasks within a single cluster.

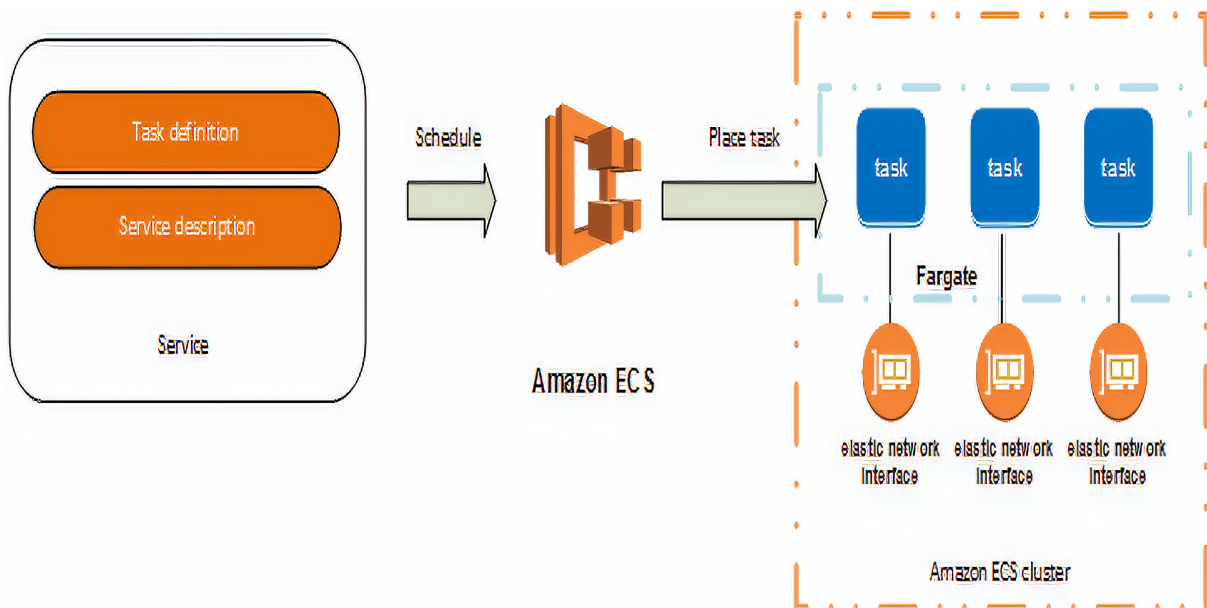


Figure 20: AWS ECS components [7].

Scheduler: The scheduler schedules your tasks inside an ECS cluster. ECS Service scheduler provides task placement strategies for applications. In addition to that, users can manually define task placement strategies for ECS clusters. ECS service scheduler monitors the scheduling strategy and takes appropriate action if the strategy is not being followed. There are two scheduling strategies:

- **Replica:** In this strategy, the scheduler deploys tasks all over the cluster in multiple containers across different regions and isolated locations in them known as availability zones.
- **Daemon:** In this strategy, the scheduler deploys tasks in exactly one container.

AWS ECS services are integrated with other AWS services as listed below.

AWS IAM: Manages authentication and authorization of other AWS resources.

EC2 Auto scaling: It enables you to automatically scale in or out tasks based on schedules and current health check status.

Elastic load balancing: To achieve high tolerance, AWS uses load balancers which distribute load across your infrastructure.

Amazon Elastic Container Registry ECR: It is Amazon's repository for container images. It is similar to the Docker hub.

AWS CloudFormation: It allows users and system administrators to manage and configure different AWS resources.

The following are steps to deploy an ECS service:

Step 1: Define a container and its tasks. We will choose an existing template of the Tomcat web server container running the tomcat image with 2GB memory and 1 virtual CPU.

Container definition Edit

Choose an image for your container below to get started quickly or define the container image to use.

sample-app
image : httpd:2.4
memory : 0.5GB (512)
cpu : 0.25 vCPU (256)

nginx
image : nginx:latest
memory : 0.5GB (512)
cpu : 0.25 vCPU (256)

tomcat-webserver
image : tomcat
memory : 2GB (2048)
cpu : 1 vCPU (1024)

custom Configure
image : --
memory : --
cpu : --

Task definition Edit

A task definition is a blueprint for your application, and describes one or more containers through attributes. Some attributes are configured at the task level but the majority of attributes are configured per container.

Task definition name	first-run-task-definition	?
Network mode	awsvpc	?
Task execution role	Create new	?
Compatibilities	FARGATE	?

Figure 21: Snapshot of step 1 in the ECS service creation process .

Step 2: Define a service. We will choose one task as the desired number of tasks, and choose the automatically created security group. Users can add application load balancers if necessary for their application in this step.

Step 1: Container and Task
Step 2: Service
Step 3: Cluster
Step 4: Review

Diagram of ECS objects and how they relate

Define your service Edit

A service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster.

Service name tomcat-webserver-service

Number of desired tasks 1

Security group Automatically create new
A security group is created to allow all public traffic to your service only on the container port specified. You can further configure security groups and network access outside of this wizard.

Load balancer type ☒ None ☐ Application Load Balancer

*Required Cancel Previous Next

Figure 22: Snapshot of ECS service creation step 2.

Step 3: Configure your cluster by specifying VPC ID and subnet ID. Users can also choose to create new VPC IDs and new subnets automatically.

Step 1: Container and Task
Step 2: Service
Step 3: Cluster
Step 4: Review

Diagram of ECS objects and how they relate

Configure your cluster

The infrastructure in a Fargate cluster is fully managed by AWS. Your containers run without you managing and configuring individual Amazon EC2 instances.

To see key differences between Fargate and standard ECS clusters, see the [Amazon ECS documentation](#).

Cluster name

Cluster names are unique per account per region. Up to 255 letters (uppercase and lowercase), numbers, and hyphens are allowed.

VPC ID Automatically create new ⓘ

Subnets Automatically create new ⓘ

*Required Cancel Previous Next

Figure 23: Snapshot of ECS service creation step 3 .

Step 4: Review all the configurations specified in the previous steps and deploy an ECS service.

Step 1: Container and Task
Step 2: Service
Step 3: Cluster
Step 4: Review

Diagram of ECS objects and how they relate

Configure your cluster

The infrastructure in a Fargate cluster is fully managed by AWS. Your containers run without you managing and configuring individual Amazon EC2 instances.

To see key differences between Fargate and standard ECS clusters, see the [Amazon ECS documentation](#).

Cluster name

Cluster names are unique per account per region. Up to 255 letters (uppercase and lowercase), numbers, and hyphens are allowed.

VPC ID Automatically create new ⓘ

Subnets Automatically create new ⓘ

*Required Cancel Previous Next

Figure 24: Snapshot of ECS service creation step 4 .

4.6 Method 4: Serverless Deployment with AWS Lambda

Serverless is a cloud architecture that allows you to focus only on your code. All other infrastructure will be managed by AWS. Users can build and run applications in a notably short period using a serverless architecture. The serverless architecture eliminates infrastructure tasks such as server provisioning, replication, load balancing, database maintenance, operating system, and coordination and maintenance of all these components.

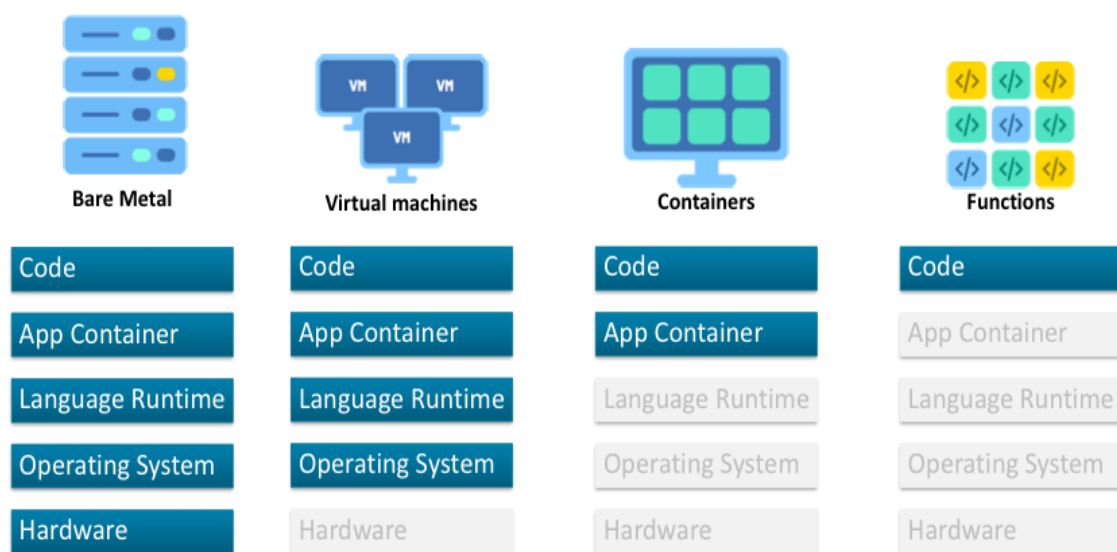


Figure 25: Software abstraction level.[8]

The software has evolved for several years, increasing its focus on code. The above figure shows evolution. **Physical machines** provide good performance yet additionally have limitations like cost and scalability. The **virtualization** era provided the decoupling of the workload from bare metal physical machines. **Containers** eliminates the need for the guest OS and allows you to focus more on code. **Serverless** provides most abstraction and needs developers to only focus on code.

AWS Lambda

Lambda is the AWS cloud computing service that lets you directly write functions without the provisioning of servers. Lambda does all the computing and scaling. Users need to pay for only compute time for which lambda functions run.

Following are two ways to trigger AWS Lambda:

- Event-driven: Where AWS Lambda runs your code in response to an event. E.g. changes to the data in your Amazon S3 bucket
- A compute lambda service is a response to an HTTP request using Amazon API Gateway or API call made using API SDK

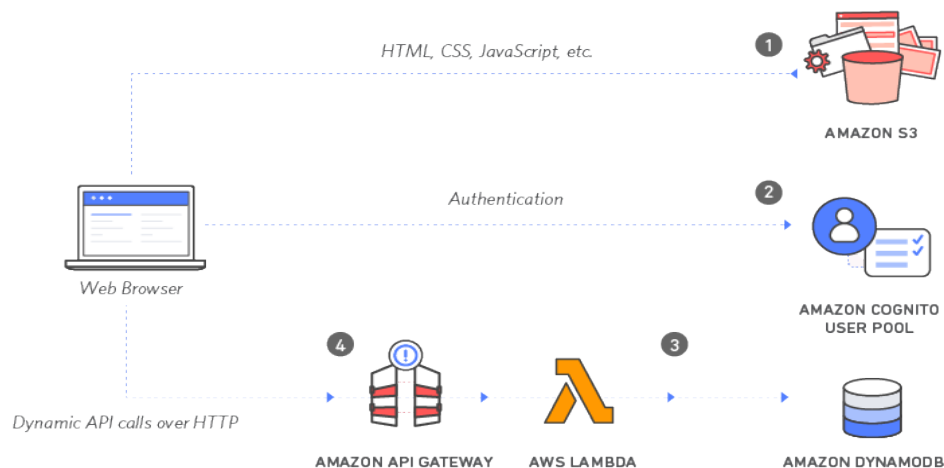


Figure 26: Serverless architecture with AWS Lambda.[9]

The above figure shows an example of serverless architecture with AWS Lambda. It has the following infrastructure components:

Amazon S3: provides storage service for all static content of an application including HTML, CSS, Javascript, and image files which are loaded in the user's browser.

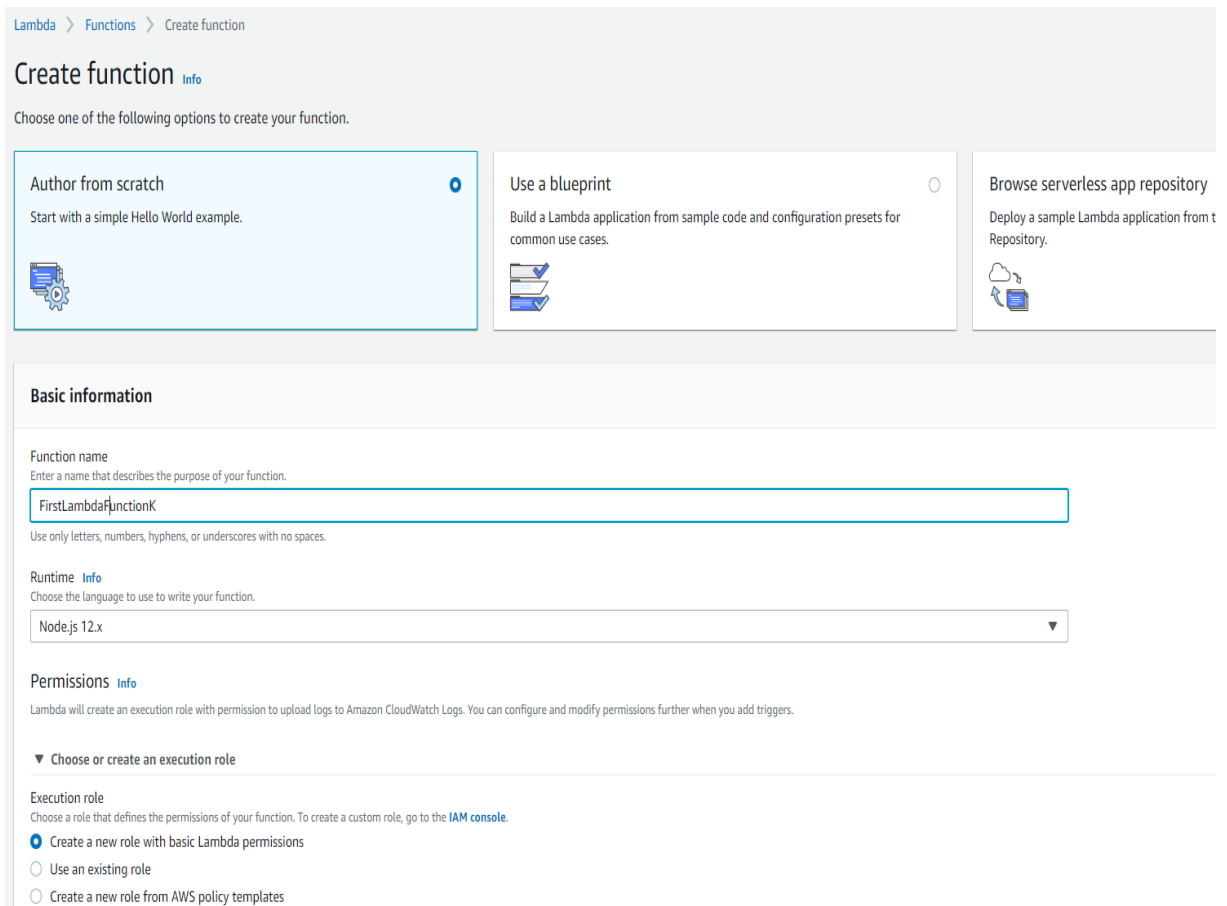
Amazon Cognito: Provides user management and authorization to secure your back-end application's API. It provides a JWT (JSON Web Token) which can be included in your HTTP request for authorization purposes.

Database: Amazon's DynamoDB provides a serverless backend.

Lambda function and API Gateway: User sends a request and receives a response from public backend API built using AWS lambda compute function and API gateway.

Following are the steps to create a serverless deployment:

Step 1: Create a Lambda function. Users can create a Lambda function from scratch or can use an existing blueprint. Using the serverless app repository is another convenient option.



The screenshot shows the 'Create function' page in the AWS Lambda console. At the top, there's a breadcrumb trail: 'Lambda > Functions > Create function'. The main heading is 'Create function' with an 'Info' link. Below this, a prompt says 'Choose one of the following options to create your function.' There are three options: 'Author from scratch' (selected with a radio button), 'Use a blueprint', and 'Browse serverless app repository'. The 'Author from scratch' option includes a description 'Start with a simple Hello World example.' and a gear icon. Below these options is the 'Basic information' section. It has a 'Function name' field with the text 'FirstLambdaFunctionK' and a note 'Enter a name that describes the purpose of your function.' Below the field is a note: 'Use only letters, numbers, hyphens, or underscores with no spaces.' The 'Runtime' section has a dropdown menu set to 'Node.js 12.x' with a note 'Choose the language to use to write your function.' The 'Permissions' section has a note 'Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.' Below this is a section titled 'Choose or create an execution role' with a dropdown arrow. Underneath, the 'Execution role' section has a note 'Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.' and three radio button options: 'Create a new role with basic Lambda permissions' (selected), 'Use an existing role', and 'Create a new role from AWS policy templates'.

Figure 27: Snapshot of serverless deployment step 1.

Step 2: Customize the lambda function code as per your application's requirement. AWS Lambda supports a wide range of programming languages like Go, Node.js, JAVA, etc.

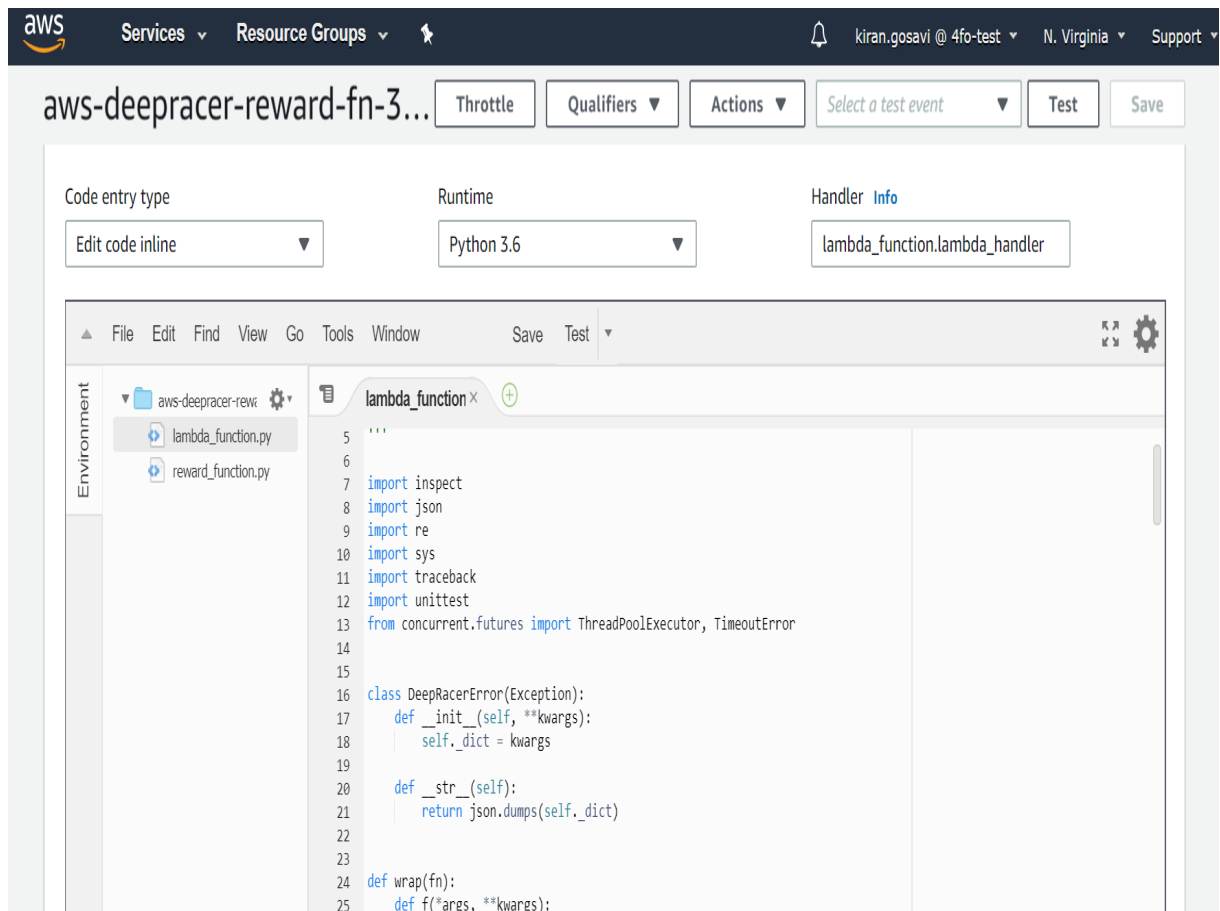


Figure 28: Snapshot of serverless deployment step 2 .

Step 3: Test your implementation

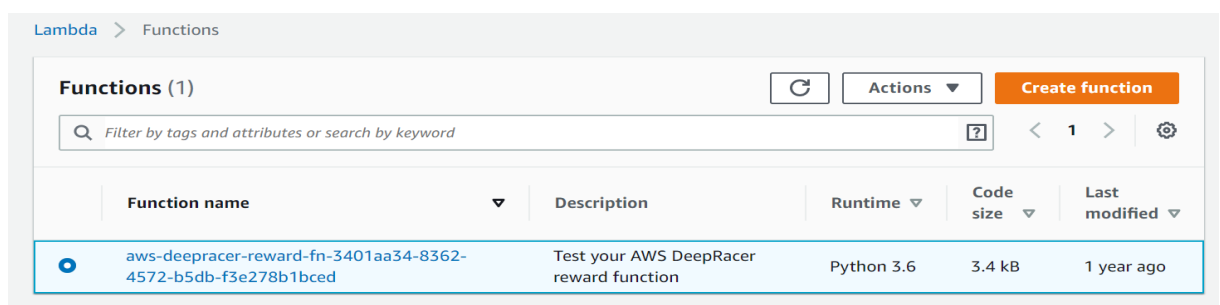


Figure 29: Snapshot of serverless deployment step 3.

5 Comparison of Deployment Methods

We have studied several deployment methods in the AWS cloud. In **method 1**, we deployed Node.js server on an EC2 instance. Method 1 deployment strategy was monitored and maintained by third-party open-source application named PM2. We have seen how containerization technology has revolutionized the DevOps field. The Kubernetes is a popular container orchestration technique. We studied Kubernetes and deployed the Tomcat web server directly on an EC2 instance in **method 2**. In addition to Google's container orchestration deployment, Amazon also has their own container orchestration technology called ECS. In **method 3**, we used ECS to deploy tasks inside a cluster. Amazon provides complete management, maintenance and coordination of infrastructure in the Serverless architecture. We created a lambda function to use serverless architecture in **method 4**.

In method 1, the **level of task automation in the process of deployment** is comparatively low. Users are required to manually set up the EC2 machine, and are required to set up application code on EC2 instances. Deployment monitoring and maintenance is not automated. In method 2, level of task automation is better than method 1, as Kubernetes manages and coordinates all the containers making it easy to deploy an application for a developer. With ECS, task automation is better than method 2 and 1. AWS serverless deployment provides the highest level of task automation amongst these 4 methods and allows developers to focus completely and solely on application code.

Scalability is an important aspect in the case of web application deployment, where

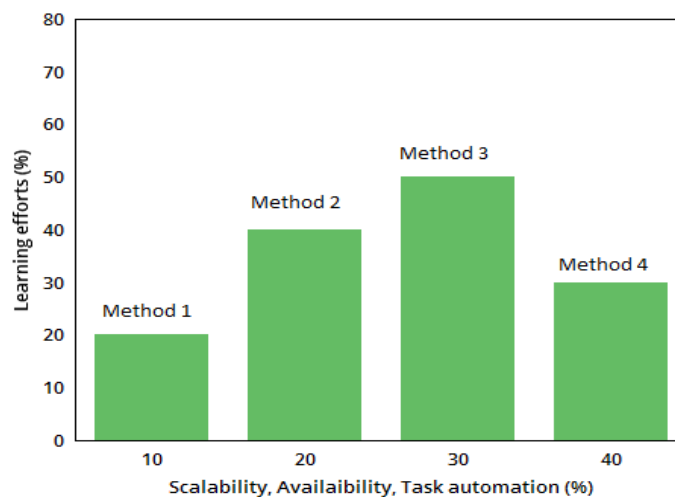


Figure 30: Comparison of AWS deployment methods.

the infrastructure needs to scale in or out depending on the number of users using the application. In method 1, scaling an application is comparatively difficult as developers might need to add other infrastructure components such as load balancers, EC2 instances manually. In method 2, scalability is managed with the help of Kubernetes. The Kubernetes can scale the deployment with addition of pods and load balancers automatically.

In method 3, scalability is managed by using other AWS services such as load balancers with additional EC2 computing instances. Scalability is fully automated in method 4. AWS automatically balances load if there is additional traffic.

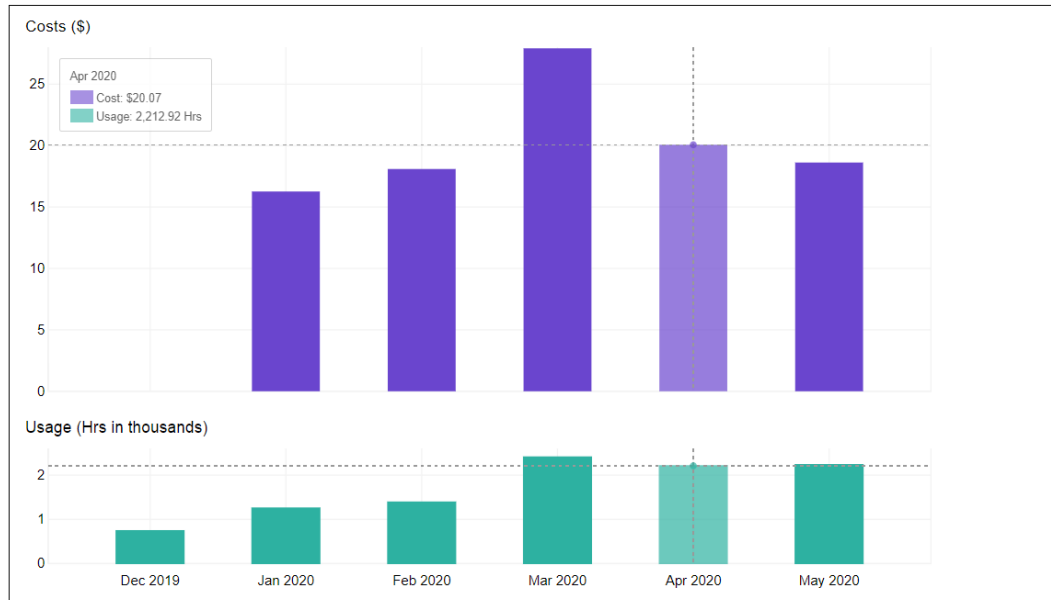


Figure 31: Snapshot of EC2 monthly computing cost (4FO).

AWS pricing for each deployment method is estimated based on various factors such as computing power, memory, AWS service cost, type of storage, region, etc. In addition to that, AWS has the policy to save money when users reserve, which indicates users will be charged 75 percent less than the equivalent on-demand price for the same capacity resources. As a result, all deployment methods discussed in this document may hold a distinct cost for various organizations, considering factors such as their purchase type i.e. on-demand or reservation, amount of usage, and regional cost. The significant part of the total cost is associated with computing using AWS services such as EC2 and Lambda. Figure 31 shows the EC2 cost analysis for the past 6 months in the 4FO. The EC2 cost in the figure is based on a reserved instance type. The 4FO takes advantage of AWS services at discounted prices since they reserve instances and pay upfront.

EVALUATED METRIC	Deployment Method			
	Deployment of Node.js on EC2 instance	Deployment of Tomcat web server Kubernetes cluster, which runs directly on EC2 instance	Deployment of Tomcat web server on a container. Tasks and services are created inside a cluster of ECS	Deployment using serverless architecture lambda functions
AUTOMATED TASKS OF APPLICATION DEPLOYMENT	Setting up a physical machine, OS system installation, drivers installation.	Management of containers, configuration, coordination of containers.	Management, coordination, configuration of containers i.e. container orchestration, Monitoring infrastructure (using AWS services like Cloud Watch).	Infrastructure creation, configuration and management. User is required to focus solely on code.
SCALING	Auto scaling is absent. Users manually need to add another EC2 instance and necessary scaling components.	Kubernetes handle auto scaling. It changes current state to desired state and scales if required.	Auto scaling is handled by ECS. It works integrated with other AWS services like ELBs to achieve that.	Serverless architecture automatically scales as per the application need.
AVAILABILITY	Availability is moderate because in case of a single EC2 instance failure, the complete application will go down.	Kubernetes distributes load across multiple worker nodes and achieves high availability.	ECS schedules tasks on multiple containers and achieves high availability.	AWS serverless architecture provides high availability using ELBs etc. other services.
LEARNING CURVE	Learning curve is lower than method 2, 3 and 4 as it requires dealing with only EC2 instances.	Learning curve is higher than method 1 because of complex Kubernetes container orchestration technology.	ECS learning curve is higher than method 2 as it requires prior knowledge of containerization technology and all other AWS services needed for integration	Learning curve is moderate (i.e. lower than method 3 & 4). however, higher than method 1.

Table 1: Comparison of AWS deployment methods

6 Summary

Amazon provides a variety of cloud services on a demand basis in the field of storage, analytics, machine learning, computing, management tools, and IoT (Internet of Things). In addition to that, we have learned that AWS web services IAM, EC2, Cloud Formation, S3, Route 53, ECS, AWS Lambda, etc. when used in a combination can form scalable and robust infrastructure. Enterprises, small businesses or startups need to respond quickly to their new and constantly changing business requirements, which can be fulfilled by AWS. Users are only charged for the time they are using the infrastructure, making infrastructure cost-efficient and convenient to build. AWS deployment methods provide building blocks for any complex architecture which is crucial and beneficial for the product development of an organization.

Amongst the different deployment strategies that we have seen, every strategy has a different set of benefits and limitations. Perks of using method 1 are inexpensive charges of the single EC2 instance and it is best suited for an application with insignificant traffic and data. Scalability, replication, and load balancing are achieved bare minimum in method 1. Method 2 has great support for complete automated configuration, management, coordination of many containers because of Kubernetes container orchestration technology. In addition to that, method 2 is highly available and robust. However, container orchestration is a new, evolving domain, and thus requires a comprehensive learning curve. Methods 3 and 4 have scalability, robustness, automated load balancing capacity. Method 4 offers a cool feature that allows developers to focus solely on application code, eliminating any need to manage and maintain infrastructure. This reduces the time to develop an application as it removes the developer's obligation to manage or configure any infrastructure components. However, complex computation like big data processing is commonly not handled by AWS lambda. AWS Lambda has a concurrent process execution limit of 1000 and can handle function storage up-to-75 GB [14].

Determining a suitable deployment strategy depends on factors such as type of application, traffic load, or the number of users using an application, development timeline, and cost. For applications with a confined number of users and thus insignificant traffic, organizations can choose method 1: deployment on the EC2 server. Kubernetes deployment strategy is a good option in case the container orchestration is demanded, which is directly deployed on an EC2 instance. Amazon provides a container orchestration strategy called ECS which can be used conveniently in integration with other AWS services. If an organization needs to focus solely on code to develop an application within a short period, instead of managing and coordinating infrastructure, then AWS serverless Lambda is a great choice. Organizations require to consider all the above factors before finalizing the deployment strategy of an application.

7 Future Work

We have seen AWS lambda serverless deployment in method 4. Serverless architecture is the future of deployments, which allow users to shift additional operations to AWS cloud. It minimizes the infrastructure worry of a developer and enables him to focus more on building and running applications without worrying about the servers. AWS Fargate is the latest AWS serverless web service. Fargate is a serverless compute engine for container orchestration. Fargate scales and manages infrastructure automatically which runs containers inside. Exploring AWS Fargate deployment strategy can be the future work for this document.

8 Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-references to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

References

- [1] *4FriendsOnly.com Internet Technologies 4FO*
<https://www.4fo.de/en/>
- [2] *Amazon Web Services (AWS)*
<https://hackernoon.com/top-5-amazon-web-services-or-aws-courses-to-learn-online-free-and-best-of-lot-d94e192054b7t>
- [3] *AWS service count*
https://en.wikipedia.org/wiki/Amazon_Web_Services
- [4] *AWS's place in cloud solution market*
<https://www.statista.com/chart/19134/cloud-vendor-use/>
- [5] *Evolution of Containerization technology*
<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>
- [6] *Kubernetes cluster example.*
<http://www.programmersought.com/article/9110533177/>
- [7] *AWS ECS components*
<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>
- [8] *Software abstraction level.*
<https://blogs.oracle.com/developers/functions-as-a-service:-evolution,-use-cases,-and-getting-started>
- [9] *Serverless architecture with AWS Lambda.*
<https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/>
- [10] *Kubernetes as k8s*
Garrison, Justin (December 19, 2016). "Why Kubernetes is Abbreviated k8s". Medium.
- [11] *Kubernetes as gold standard*
<https://blog.newrelic.com/engineering/container-orchestration-explained/>
- [12] *Working of CloudFormation API.*
<https://aws.amazon.com/blogs/devops/customers-cloudformation-and-custom-resource/>
- [13] *Kubernetes Basics*
<https://kubernetes.io/docs/tutorials/kubernetes-basics/>
- [14] *AWS Lambda limit*
<https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>
- [15] *Accessing AWS S3 Storage.*
<https://aws.amazon.com/blogs/database/use-amazon-s3-to-store-a-single-amazon-elasticsearch-index/>