Disassociating a Monolithic Application Into Microservices without any Downtime

Dhivin Kumar P, Vignesh S

Department of Computer Science & IT, Jain University, Bangalore, Bengaluru, Karnataka, India

ABSTRACT

As the code architecture of an application develops to face newer updates, it is difficult to update and maintain. Because of this, a Traditional Monolith Architecture is hard to scale, along these lines to present new systems, highlights and advances is a perplexing issue which thusly constrains the zone of development. In this research article with the assistance few Amazon Web Services (AWS) technologies, a straightforward solid application in a Docker Container is deployed, and the same application is conveyed as microservices, at that point change traffic to the microservices with no down time. When building up a server-side application, it is important to begin with a Modular or layered approach comprising of few components. Despite the fact that there are few advantages of a Monolithic Application Architecture, it additionally houses some disadvantages. To overcome the downsides, Microservices Application Architecture was presented. Every Microservice is an application which frames an assortment of littler, interconnected administrations which has its own business rationale alongside different connectors. The Microservice architecture fundamentally impacts the connection between the application and the database. Rather than offering a solitary database pattern to different administrations, each assistance has its own database outline. These Microservices can be composed with the assistance of various structures and programming dialects and these Microservices can be sent autonomously as a solitary service or a cluster of services. AWS technologies used are as follows: Amazon EC2, Amazon ECS, Amazon Elastic Container Registry, Amazon VPC, AWS Cloud Formation, Elastic Load Balancing.

KEYWORDS: *Monolithic Application, Microservice, AWS (Amazon Web Service), AWS EC2 (AWS Elastic Cloud Compute), ECS (Elastic Container Service), Amazon VPC (Virtual Private Cloud), Amazon Elastic Container Registry, AWS Cloud Formation, Elastic Load Balancing*

How to cite this paper: Dhivin Kumar P | Vignesh S "Disassociating a Monolithic Application Into Microservices without any Downtime" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-4 | Issue-3, April 2020, pp.881-885, URL: www.ijtsrd.com/papers/ijtsrd30704.pdf

IJTSRD30704

I. INTRODUCTION

Inside a microservices design, every application segment runs as its own service and communicates with different services by means of characterized API. Microservices are worked around business capacities, and each service performs a solitary function. Microservices can be composed utilizing various structures and programming languages, and can be deployed independently as a solitary service, or as a cluster of services. At first a monolithic node.js application is deployed to a Docker Container, followed by decoupling the application into microservices with no down time. The node.js application hosts a straightforward message board with threads and messages between clients. Eventually running a straightforward monolith application in a Docker Container, finally deploying the same application as microservices and change traffic to the microservices with no zero downtime.

Independent Scaling:

At the point when features are broken into microservices, the measure of framework and number of cases utilized by every microservice class can be scaled up or down freely. This makes it simpler to gauge the expense of a specific feature, recognize highlights that should be improved first, just as keep execution dependable for different feature in the event that one specific component is running wild on its asset needs.

Isolation for Security:

In a monolith application on the off chance that one element of the application has a security rupture, for instance a powerlessness that permits remote code execution, at that point one must expect that an aggressor could have accessed each other element of the framework also. This can be risky if, for instance, the transfer include has a security issue which winds up trading off your database with client passwords. Isolating highlights into microservices utilizing Amazon ECS will allow in tying down access to AWS assets by giving each service its own IAM role. At the point when microservice best practices are followed, the outcome is that if an aggressor bargains one help, they just access the assets of that administration, and can't on a level plane access different asset from different administrations without breaking into those services too.

Deployment Velocity:

Microservices bring down the disadvantages or risks during development, which can empower a group to assemble quicker. In a monolithic application, adding features can conceivably affect each other feature that the monolithic application contains. A developer should cautiously consider the effect of any code they include and guarantee that they don't break anything. Then again, an appropriate microservice architecture has new code for another component going into another service. Developers can be certain that any code they compose will really not have the option to affect the current code at all except if they unequivocally compose an association between two microservices.

A variety of microservice technologies have evolved over the past two years.[1] The growing user interest and widespread adoption of Docker and container technology have forced legacy vendors to deliver their at least first container products but it needs to be seen in long run how these technologies can smoothly integrate and meet the technical requirements of legacy systems.[3] The application developers and administrators can run same application on laptops, VMs or cloud by automating deployment inside containers.[4] The container-based virtualization does not use complete virtual machines and hence no overhead of running a completely installed operating system.[2]

Speed:
Deploying a Container with another arrival of code should be possible without critical arrangement overhead. Operational performance is improved, in light of the fact that code worked in a container on a developer's machine can be effortlessly moved to a test server by basically moving the container. At assemble time, this container can be connected to different containers required to run the application stack. This is when compared to traditional Cloud applicants. [14]

Dependency Control: A Docker container image is a point in time catch of an application's code and conditions. This permits a developer organization to make a standard pipeline for the application life cycle.

Resource Efficiency: Containers encourage improved asset productivity by permitting various heterogeneous procedures to run on a solitary framework. Asset effectiveness is a characteristic consequence of the segregation and designation systems that containers use. Containers can be limited to expend certain measures of a host's CPU and memory. By understanding what assets, a container needs and what resources are accessible from the basic host server, it is possible to measure the right resources used with littler hosts or increment the thickness of procedures running on a solitary huge host, expanding accessibility and streamlining asset utilization of cloud services. [15]

II. FEATURES OF MICROSERVICES
There are few benefits of the Microservice architecture, such as: [16] [17]

- ➢ It handles the issue of intricacy by deteriorating application into a lot of sensible services which are a lot quicker to create, and a lot more obvious and keep up.
- ➢ It empowers each service to be developed autonomously by a group that is centered around that administration.
- ➢ It decreases hindrance of embracing new advancements since the engineers are allowed to pick whatever advances makes sense for their administration and not limited to the decisions made toward the beginning of the undertaking.
- ➢ Microservice design empowers every microservice to be conveyed autonomously. Accordingly, it makes continuous development feasible for complex applications.
- ➢ Microservice architecture empowers each service to be scaled autonomously.

III. SERVICES FOCUSED
Visual Studio Code joins the effortlessness of a source code proofreader with strong developing tools, as IntelliSense coding and troubleshooting. VS Code likewise coordinates with manufacture and scripting instruments to perform normal assignments making ordinary work processes quicker. VS Code has support for Git so one can work with source control without leaving the supervisor including seeing pending changes differences. VS Code consolidates web advances, for example, JavaScript and Node.js with the speed and adaptability of local applications.[5]

Amazon Elastic Compute Cloud (Amazon EC2) is a web administration that gives secure, resizable computation capacity in the cloud. It is intended to make web-scale distributed cloud computing simpler for developers. Amazon EC2's straightforward web administration interface will permit in acquiring and designing capacity with negligible friction. It also permits with unlimited authority of computing resources and allows to run on Amazon's demonstrated processing condition. Amazon EC2 decreases the time required to get and boot new server occasions to minutes, permitting you to rapidly scale limit, both here and there, as your registering necessities change. Amazon EC2 changes the financial matters of processing by permitting you to pay just for limit that you really use. Amazon EC2 gives engineers the instruments to assemble disappointment versatile applications and disconnect them from basic disappointment situations. [6]

Amazon Virtual Private Cloud (Amazon VPC) is utilized to arrangement a consistently disconnected area of the AWS Cloud where we can dispatch AWS assets in a virtual system that you characterize. You have full oversight over your virtual systems administration condition, including determination of your own IP address run, making of subnets, and design of course tables and system portals. Amazon VPC gives propelled security highlights, for example, security gatherings and system get to control records, to empower inbound and outbound sifting at the case level and subnet level. Secretly interface with SaaS arrangements bolstered by AWS PrivateLink. Utilizing Amazon VPC, the procedure of traffic reflecting to catch and mirror organize traffic for Amazon EC2 cases is conceivable. [7]

AWS CloudFormation gives a typical language to portray and plan for all the framework assets in the cloud environment. CloudFormation permits allows in utilizing the programming dialects or a basic book document to display and arrangement, in a computerized and secure way, all the assets required for the applications over all districts and records. AWS CloudFormation permits us to show the whole foundation with either a content record or programming dialects. AWS CloudFormation helps in provisioning the assets in a protected, repeatable way, permitting to fabricate the framework and applications, without the requirement for manual interferences or custom contents. [8]

Amazon Elastic Container Service (Amazon ECS) is a profoundly versatile, superior compartment arrangement administration that bolsters Docker containers and permits in handily run and scale containerized applications on AWS. Amazon ECS dispenses with the requirement to introduce and work with the container orchestration programs, administer and scale a bunch of virtual machines, or calendar holders on those virtual machines. Containers can be sent and administer without the need to arrangement or deal with the servers. Amazon ECS is profoundly incorporated with the AWS Services. [9]

Amazon Elastic Container Registry (ECR) is a completely managed Docker container vault that makes it simple for developers to store, admin, and convey Docker Container Images. Amazon ECR is coordinated with Amazon Elastic Container Service (ECS), improving your advancement to creation work process. Amazon ECR has the pictures in an exceptionally accessible and adaptable engineering, which permits the clients to dependably send holders for their applications. Incorporation with AWS Identity and Access Management (IAM) gives asset level control of every store. [10]

Elastic Load Balancing naturally disseminates approaching application traffic over different targets, for example, Amazon EC2 Instances, IP addresses or even containers and Lambda Functions. It can deal with the changing heap of the application traffic in a solitary Availability Zone or over various Availability Zones. ELB offers three sorts of burden balancers which highlight high accessibility, programmed scaling and hearty security, that is expected to make the applications issue lenient. [11]

Amazon Elastic Block Store (EBS) is a simple to utilize, elite square stockpiling administration intended for use with Amazon Elastic Compute Cloud (EC2) for both throughput and exchange escalated remaining tasks at hand at any scale. An expansive scope of outstanding burdens, for example, social and non-social databases, undertaking applications, containerized applications, enormous information investigation motors, record frameworks, and media work processes are generally sent on Amazon EBS. [12]

Amazon Route 53 is an exceptionally accessible and versatile cloud Domain Name System (DNS) web administration. It is intended to give designers and organizations a very solid and savvy approach to course end clients to Internet applications by interpreting area names that the PCs use to interface with one another. Amazon Route 53 successfully associates client solicitations to framework running in AWS –, for example, Amazon EC2 occurrences, Elastic Load Balancing load balancers, or Amazon S3 pails – and can likewise be utilized to course clients to foundation outside of AWS. Amazon Route 53 is completely agreeable with IPv6 too. Amazon Route 53 can be utilized to arrange DNS wellbeing checks to course traffic to solid endpoints or to autonomously screen the strength of the application and its endpoints. Amazon Route 53 Traffic Flow makes it simple for you to oversee traffic all around through an assortment of steering types, including Latency Based Routing, Geo DNS, Geoproximity, and Weighted Round Robin—which can all be joined with DNS Failover so as to empower an assortment of low-inertness, issue lenient models. [13]

IV. PROBLEM IDENTIFICATION

The traditional methodology of implementing a monolith application has an impediment in size and intricacy. Application is excessively huge and complex to completely comprehend and made changes quick and effectively. Monolith applications has a hindrance to embracing new innovations. Since changes in systems or dialects will influence a whole application it is incredibly costly in both time and cost. The size of the application can hinder the beginning up time. It is necessary to redeploy the whole application on each update.

Effect of a change is generally not very surely known which prompts in performing broad manual testing. Nonstop development and deployment are troublesome. Monolith applications can likewise be hard proportional when various modules have clashing asset necessities. Another issue with solid applications is unwavering quality. Bug in any module (for example memory spill) can conceivably cut down the whole procedure. Also, since all cases of the application are indistinguishable, that bug will affect the accessibility of the whole application.

V. PROPOSED SYSTEM

In the proposed architecture, at first build a container image picture for monolithic node.js application that will be pushed to Amazon Elastic Container Registry in the AWS Platform. Container will permit to effectively package an application's code, setups, and configurations into simple to utilize building obstructs that convey ecological consistency, operational effectiveness, engineer efficiency, and rendition control. Containers can help guarantee that applications send rapidly, dependably, and reliably paying little mind to organization condition. In the venture organizer, the envelopes for framework and administrations. Foundation holds the AWS Cloud Formation framework setup code to use in the following stage. Requests between tiers are automatically load- balanced, spreading requests evenly between processes in a tier. [18]

The organizer administrations contains the code that frames the node.js application, including the database db.json, the server server.js, package.json, and the application docker file. With straightforward API calls, it is possible to dispatch and stop Docker-empowered applications, inquiry the total condition of your bunch, and access numerous commonplace highlights like security gatherings, Elastic Load Balancing, EBS volumes, and IAM jobs. One use Amazon ECS to plan the arrangement of compartments over the cluster dependent on the asset needs and accessibility necessities. Break the node.js application into a few interconnected administrations and push each assistance's picture to an Amazon ECR archive. Node.js application is conveyed as a lot of interconnected administrations behind an Application Load Balancer (ALB). At that point, utilize the ALB to consistently move traffic from the stone monument to the microservices.

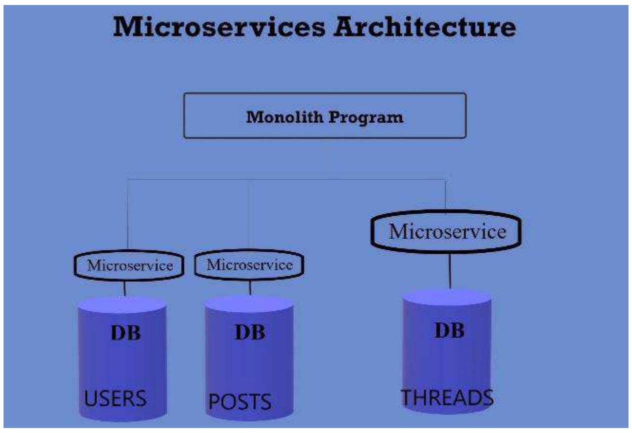
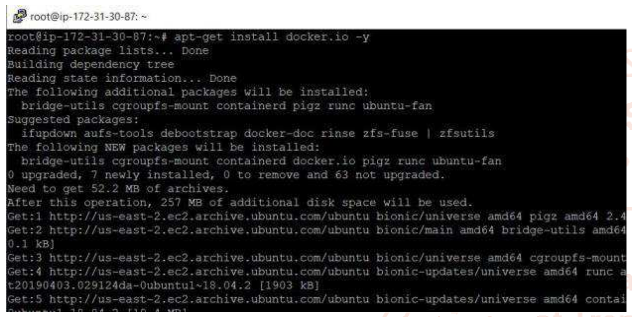


Fig1. Microservices architecture

VI. IMPLEMENTATION METHODOLOGY

Containerize the Monolith application:

Build the Container image for the monolith node.js application that needs to be pushed into Amazon Elastic Container Registry. Install Docker and ensure its running condition by using the command '*Docker –version'.*



```
root@ip-172-31-30-87: ~
root@ip-172-31-30-87:~# apt-get install docker.io -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils cgroupfs-mount containerd pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools debootstrap docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils cgroupfs-mount containerd docker.io pigz runc ubuntu-fan
0 upgraded, 7 newly installed, 0 to remove and 63 not upgraded.
Need to get 52.2 MB of archives.
After this operation, 257 MB of additional disk space will be used.
Get:1 http://us-east-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 pigz amd64 2.4
Get:2 http://us-east-2.ec2.archive.ubuntu.com/ubuntu bionic/main amd64 bridge-utils amd64
0.1 kB]
Get:3 http://us-east-2.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 cgroupfs-mount
Get:4 http://us-east-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 runc a
t20190403.029124da-0ubuntu1~18.04.2 [1903 kB]
Get:5 http://us-east-2.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 contai
```

Fig2 – Docker Installation

Use Visual Studio Code as the Text Editor for scripting. This needs to be implemented in the local environment (Host Computer). Infrastructure holds the AWS Cloud Formation infrastructure configuration code to use in the next step. The folder services contains the code that forms the node.js application, including the database db.json, the server server.js, package.json, and the application dockerfile. Create a repository and select a task definition (here EC2) in the AWS portal. Build the image using the 'docker build -t api' command in the terminal. Followed by tagging and pushing the image into the Amazon Container Registry.

Push the Monolith:

Use Amazon Elastic Container Service (Amazon ECS) to launch a managed cluster of EC2 compute instances and deploy the containerized image as a container running on the cluster.

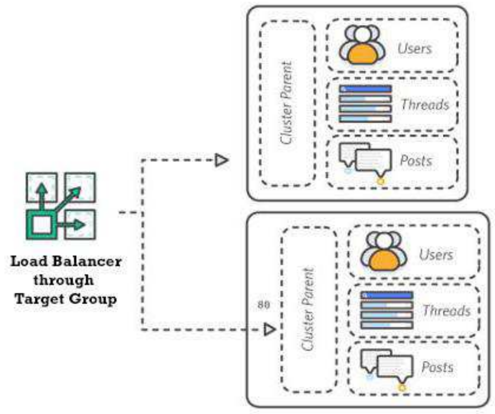


Fig 3 – Request & Response via microservices

A. Client: The client will make a request over port number 80 to the load balancer.
B. Load Balancer: The load balancer will distribute the requests across all available ports.
C. Target Groups: Instances are then registered in the target group of the application respectively.
D. Container Ports: Each container will run a single application process which will bind the node.js cluster to the port 80 within its namespace.
E. Containerized node.js Monolith: The node.js cluster is responsible to distribute traffic to the sub-ordinates within the monolithic application. Though the architecture is containerized, it still remains to be a monolith one because each container has all the same features of the rest of the containers.

The Application Load Balancer (ALB) allows the service of accepting incoming traffic. The ALB will automatically route traffic to container instances running in the cluster using them as the target group. The node.js application will enable in routing the traffic to each worker based on the URL.

Breaking of Monolith Application:

Now to break the node.js application into multiple interconnected services and push each image of the service to an Amazon ECR repository.

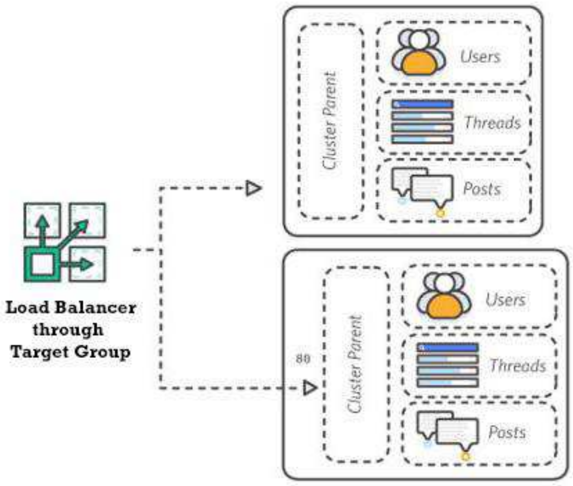


Fig 4 – Architecture via Amazon ECS and Amazon ELB

For deploying the application as three microservices, it is important provision three different repositories (one for each service) in Amazon ECR. Run *aws ecr get- login –no-include-email –region [region].* In the terminal, run *docker build -t [service-name] ./[service-name]*

Example: "*docker build -t posts ./posts*." Once build completes, tag the image so it is possible to push it to the repository: *docker tag [name-service]:latest [account-id].dkr.ecr.[region].amazonaws.com/[service- name]:v1* example: *docker tag post:latest [acc-id].dkr.ecr.us-west-2.amazonaws.com/posts:v1*. <Followed by>Running the docker push to push your image to ECR: *docker push [account- id].dkr.ecr.[region-name].amazonaws.com/[service-name]:v1*

The node.js application is deployed as a set of interconnected services behind an Application Load Balancer (ALB). Then, use the ALB to seamlessly shift traffic from the monolith to the microservices. Deploy three new microservices onto the same cluster and write Task Definitions for each service. all the services (monolith and three microservices) are running behind the same load balancer. To make the transition from monolith to microservices, start routing traffic to the microservices and stop routing traffic to your monolith. Deploy the three services onto the cluster and Switch over the traffic to the Microservices.

CONCLUSION

Operational speed is improved, in light of the fact that code worked in a container on a developer's host/local machine and can be moved to a test server effortlessly by essentially moving the Container service. Build the Docker container image for the monolith node.js application and push it to Amazon Elastic Container Registry. Use Amazon Elastic Container Service (Amazon ECS) to start the managed cluster of EC2 compute instances and convey the deployed image as a container running on the cluster. The assignment definition reveals to Amazon ECS how to send the application containerizes over the cluster. The node.js application courses traffic to every independent services dependent on the URL and sends the node.js application as a lot of interconnected administrations behind an Application Load Balancer. At that point, utilizing the ALB to flawlessly move traffic from the monolithic application to the microservices was a success. Update the default rule to advance to drop-traffic. Turn down the Monolith and now, traffic is streaming to the deployed microservices and it is possible to turn down the Monolith administration. ALB courses traffic dependent on the request of the URL.

The scope of this system is to build a reliable Application services managed from a remote location (AWS Platform). Though there are Microservices being embedded in the modern application, they are vulnerable to downtime and multi-threaded management. To handle this, using microservices in Cloud Platform to install any given application and manage them remotely is an advancement to the applications than the ones which are deployed at a local host machine.

References:-

[1] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, 2015, pp.116, 113–115.

[2] container-based virtualization (operating system-level virtualization) http://searchservervirtualization.techtarget.com/definition/containerbased-virtualization-operating-system-level-virtualization

[3] Amazon's Container Strategy, Examinedhttp:// www.informationweek.com/cloud/infrastructure- as-aservice/amazonsy container-strategy-examined/a/did/1317515?itc=edit_in_body_cross

[4] https://code.visualstudio.com/docs/editor/whyvscode

[5] https://aws.amazon.com/ec2/

[6] https://aws.amazon.com/vpc/

[7] https://aws.amazon.com/cloudformation/

[8] https://aws.amazon.com/ecs/

[9] https://aws.amazon.com/ecr/

[10] https://www.docker.com/resources/what- container

[11] https://aws.amazon.com/elasticloadbalancing/

[12] https://aws.amazon.com/ebs/

[13] https://aws.amazon.com/route53/

[14] S. Kanev, J. Darago, K. Hazelwood, P. Ranganathan, et al., "Profiling a warehouse-scale computer," in Proc. 42nd Annu. Int. Symp. Comput. Archit., 2015, pp. 158–169.

[15] M. Ferdman, A. Adileh, et al., "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in Proc. 17th Int. Conf. Architectural Support Programming Languages Operating Syst, 2012, pp. 37-48.

[16] M. Schwarzkopf, A. Konwinski, M. Abd-El- Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys)*, Prague, Czech Republic, 2013, pp. 351– 364.

[17] Zimmermann O. Microservices Tenets: Agile Approach to Service Development and Deployment. Comput Sci Res Dev. 2016; 32(3):301–10

[18] W. John, J. Halén, X. Cai, C. Fu, T. Holmberg, V. Katardjiev, M. Sedaghat, P. Sköldström, D. Turull., "Making cloud easy: design considerations and first components of a distributed operating system for cloud," {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 18).