22.4.2021

Deployment Plan

E-Commerce Service

Sabina Mammadova
PRODUCTION & OPERATIONS MANAGEMENT | ADA UNIVERSITY

Contents

AWS Infrastructure

Let us see the process flow when the customer does an operation on the website.

At first, the request will encounter AWS CloudFront in which the service speeds up the secure delivery of both static and dynamic content of the website to the customers around the world. Moreover, it offers a multi-tier cache by default ensuring diminished latency and load on the origin servers when the object is not cached in the edge location. Next, it will trigger highly available and scalable cloud DNS web service – Route53 which allows to route the customers to the infrastructure both inside and outside of AWS. Since the website is utilized worldwide, it is appropriate to route the requests of the users to the nodes that are geographically close to them to provide the lowest latency. The next station for the request is load balancing where the request will be handled by the worker node which has the least rate of business since the traffic is routed btw zones in which cross-zone load balancing feature is adopted. At the last step, the request is handled by the AWS EKS cluster NODEGROUPs and the response is sent back to the client.

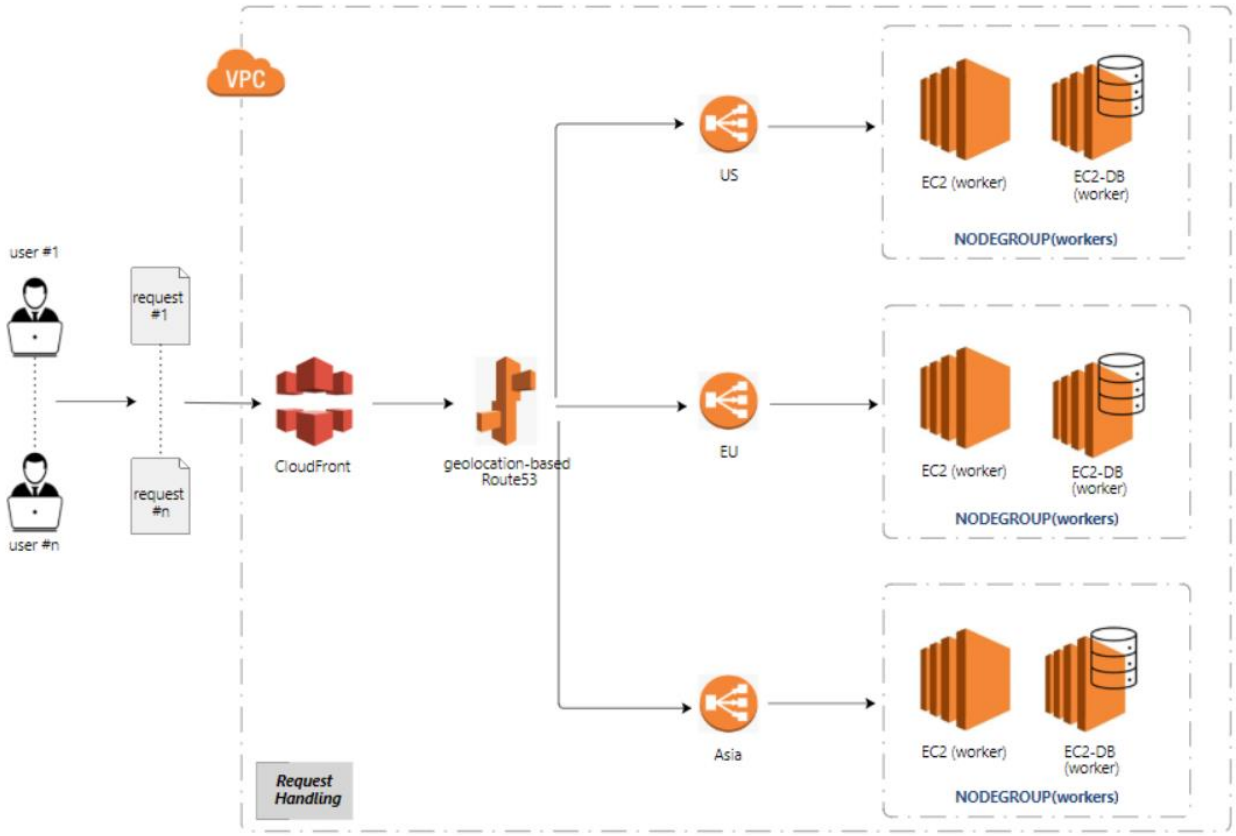

Figure 1.

It is highly recommended, even required for especially this business built on e-commerce service, to recognize its customers' preferences and offer the respective products to increase customer satisfaction, acquisition, and retention. However, it is not easy to control and get valuable insights out of large chunks of continuous data. Moreover, it is also crucial to monitor and manage the overall performance, health checks of running applications. Below VPC, which

will be connected to the VPC in Figure 1, is prepared for BI Analysts and IT Managers who need this authority to collect metrics for the insights or get an operational visibility.

The process is the following:

AWS Cloudwatch collects monitoring and operational data in the form of logs, metrics, and events in near real-time. AWS Kinesis Firehose is automatically scaling service used to reliable load streaming data into stores/analytics services, yet it needs AWS Cloudwatch for the monitoring although the pricing for the logs is ~ 6 times lower for 1TB of logs/month with 90 days retention. Next, the data is pushed to AWS S3 because it is archived in Cloudwatch for 2 weeks, after which it is removed and processed in parallel across Hadoop distributed cluster of virtual servers on AWS EMR for data analysis step. After massive computation, the processed data is put into AWS Redshift which is a cloud data warehousing service to query enormous amount of structured/semi-structured data using standard SQL. AWS S3 + AWS Athena can be considered as an alternative to the Redshift where Athena works directly on top of S3 (working as read-only service from an S3 perspective) and brings advantages in terms of portability and cost, whereas Redshift requires a cluster for data extracts before querying and puts its uniqueness in terms of performance, scale and handling large sets. At the end, to easily create and publish interactive ML-powered BI dashboards for insights, scalable, cloud-based, serverless AWS Quicksight service can be used.

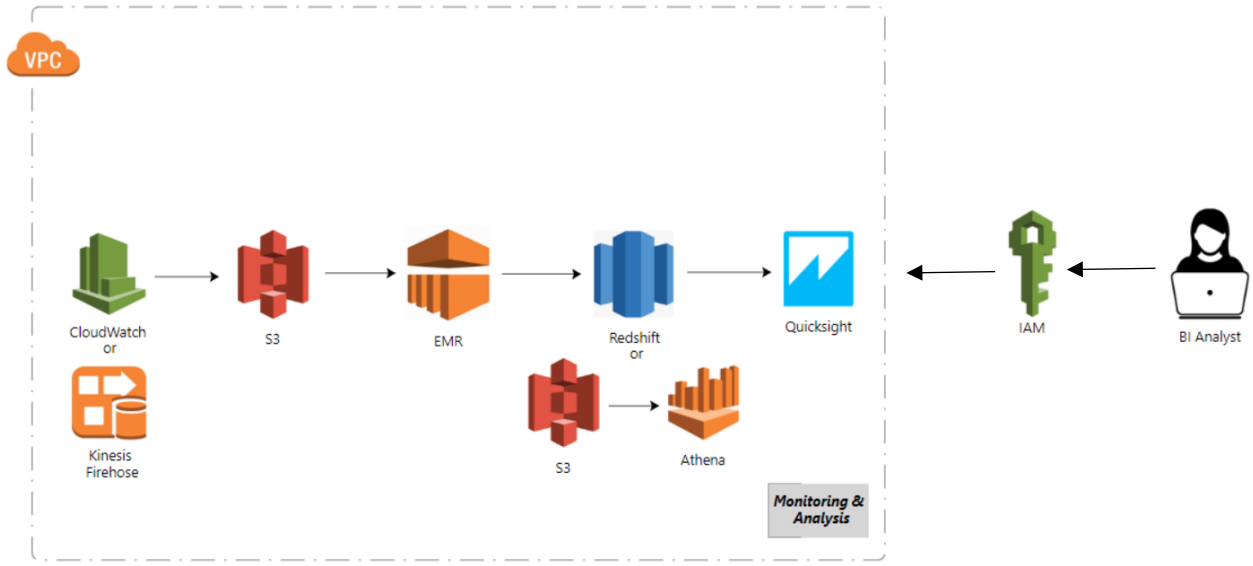

Figure 2.

Production Environment

Let us analyze the production environment.

In general, AWS EKS container service is used to host the whole system. EKS Cluster is created in which master nodes serves the role of Control Plane (scheduling and orchestration) and AWS provisions K8s master nodes in the background. Since it is managed by AWS, it will replicate the master nodes across multiple Availability Zones on the region ensuring high availability. Master

nodes include *Etcd* storage which includes the current configuration of K8s cluster (if it is lost, then the cluster is gone) and it is replicated as well. To assure the cluster does not run out of compute resources, *region specific auto scaling groups* is used to spread the resources across multiple AZs making the system durable to zone specific maintenance such that it simplifies the troubleshooting scenarios because less computing instances will be debugged.

The actual computing environment consists of worker nodes – EC2 instances. The infrastructure of the worker nodes is semi-managed by EKS with NODEGROUP meaning EC2 instances are grouped where Nodegroup creates, deletes instances, gets required processes like container runtime, K8s worker processes by itself but the configuration of the instance should be done by an administrator or developer if privileged access to control plane exists. The communication between master and worker nodes is established via K8s Processes installed on worker nodes along with docker agent. The storage environment also includes installed database agents on EC2 instances provisioned by Portworx – a distributed block storage solution that deeply integrates with Kubernetes to run and manage any stateful service in cloud/on-premises. By doing this, the separation of concerns between the processing and the storage is fulfilled by Kubernetes and Portworx respectively.

What happens when the developer deploys a containerized image to the cluster?

He/she uses programmatic access to cloud by AWS CLI and either downloads Kubectl – command-line tool to run commands for Kubernetes clusters – or if downloaded, uses *kubectl* command to deploy the containerized application code to the cluster. Besides, the developer downloads (can be with spec generator with various configuration options to get the URL for installation)/uses portworx which is fully supported by kubectl. Next, he/she creates:

- StorageClass file to define the type of storage resources like the size, number of replicas, e.g., assume it is set to 3 – if the application gets the storage of this type, portworx ensures 3 copies of the volume), I/O priority, snapshot schedule;
- PVC (Persistent Volume Claim) file to request PV resources from cluster administrators without having specific knowledge of the underlying storage infrastructure;
- DB_Config (DB-PVC) file to set the configuration details for the database and include PVC name to associate the configuration with the claim.

The claim will be accepted if the total PV capacity set by admin has not been exceeded and the creation is done dynamically by the Portworx. The size of the volume can be set to any number and expanded based on the needs of the application and StorageClass should be declared in the PVC file to make the storage visible to the worker node.

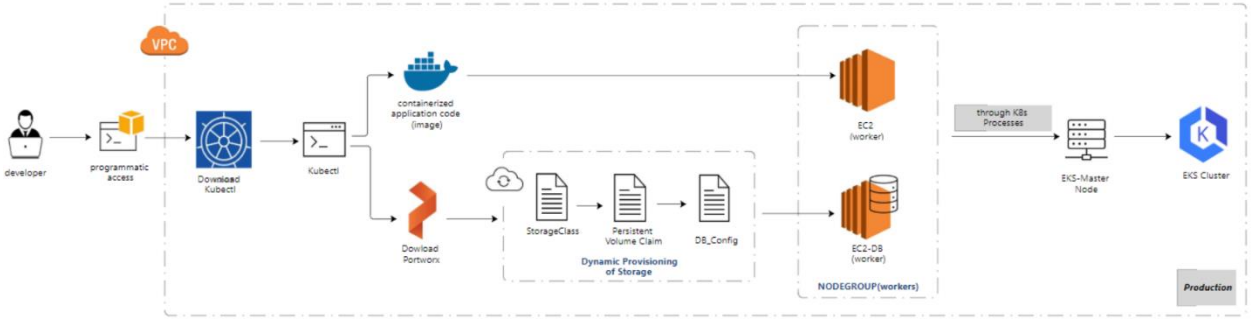

Figure 3.

At the end, to smoothly run the containerized applications in the cluster, they need certain sensitive data/credentials like passwords, keys, SSL certificates. Kubernetes Secrets is the mechanism to store, distribute (multiple volumes can use the same secret), protect the secret information, and write by the applications. So, the only thing is to create a secret or use the existing one in the namespace for the further work.

Deployment Plan

With respect to continuous delivery so to speak *ideology*, "Blue-Green" deployment strategy will be followed which is the perfect match for Kubernetes orchestration platform. When the developer wants to update the application running in the cluster, he/she does not touch it but marks as blue and deploys the new version of that application to the cluster and marks it as green. At first, the developer tests the new version in the production environment without any downtime experience by the users since they do not see/feel the change, if it works as expected, the old version will be discarded to free the resources and the user traffic will be routed to the new version. In case of technical issue occurrence on the new version, the service will switch back to the old version. With the Codefresh and Argo Rollouts, more advanced delivery methods can be applied to support the deployment process.

Since the workload and the storage are separated, Portworx handles the up-to-date, synchronous replication of the volume by creating a topology-aware replica on some other node in the cluster in a different AZ with respect to the availabilities and boundaries of the AWS. There is no need to recreate and populate a new database with the data in the existing volume in another zone which is both error prone and time consuming leading to application downtime. If an unintended deletion, DB migration/update failure or network partition which might make the worker node no longer available happens, Kubernetes will automatically reschedule the pod which runs the DB in the cluster to the node that has the copy of the data. So, the update on the volumes is operated in isolation from the computing environment leading to straightforward and backed-up deployment.

It is also important to have DR (Disaster Recovery) plan in another region along with the local production environment. If the local environment is lost, then DR is lost too. Thus, Portworx has a feature called Cloud Snap which enables to take the snapshot of one or more individual container volumes from larger volumes, push to any storage service, e.g., S3, and pull down to another environment. So, DR site will be available at the location with a network latency but at least it will be guaranteed to have the working site on hand.