Certified Solution Architect – Professional

Contents

[**Section 3: Designing Solutions for High Availability and Business Continuity** 1](#_Toc35683824)

[**Section 4: Design for operational complexity and cost control** 5](#_Toc35683825)

[**Section 5: AWS Services and Strategies for Deployment and Operation/Monitoring Management** 9](#_Toc35683826)

**12/31/2019**

# **Section 3: Designing Solutions for High Availability and Business Continuity**

* Business continuity and Disaster Recovery
* Resiliency and Fault Tolerant
* Redundancy and high availability
* Cost optimization
* Performance
* Security
* Monitoring
* Saleable and Elasticity
* Ease of deployment
* Migration and Hybrid architectures

Fault Tolerant: Ability of a system/application/infrastructure to be operational even if one or more system fails is call fault tolerant. A fault tolerant system should be the one which can recover/failover with minimal or no human interventions on a even of any failure.

AWS recommended to create a library of own AMI, with own standard & best practice. Ensure the AMI is up to date and stable for production usages.

Floating IP (elastic IP) = the IP that can be shift from the primary instance to secondary on an event of any failure.

Create regular EBS snapshots – EBS are region specific, to be used in another region we need to copy the snapshot to the desired region before creating the EBS volume from that snapshot. EBS are region specific and EBS volume are AZ specific.

Multiple site architecture: In this approach the goal is to have two or more independent copy of each application stack into two or more availability zone (site). Every application tier need to have redundant copy , in case of failure the traffic can be route to the alternative availability zone (site) – this can further be made more effective by introducing elastic load balance which can automatically route traffic between AZ, OR having a route53 health check created to route traffic between different region, alternatively one can also use elastic ip for routing traffic, so that when the instance fails the elastic ip can be remapped to another running instance in same/different AZ. Implement Autoscaling to address flatulating production load.

**01/01/2020**

Disaster Recover – Recovery plan from a system failure to resume business continuity without impacting business operation.

|  |  |
| --- | --- |
| **Benefit of having AWS as alternative site** | **Counter Argument** |
| Don’t need to negotiate contract with other provide in another region. | Avoiding vendor locking – have a multi-vendor/multi-cloud landscape. |
| One can use the same underline AWS technology across region | Need to have trained staff in more than one cloud technology – will be cost intensive to maintain multi talent pool. |
| One can use same tool/AWS build artifacts/API across region | One need to maintain different tool/build artifacts/APIs for different cloud. |

**Recovery Time Objective (RTO) –** Time taken after a disruption to restore a business process to its service level agreement as define by the operational level argument (OLA).

**Recovery Point Objective (RPO)** – Acceptable amount of data loss measure in time. For example, If the disruption occurs at 12 Noon and RPO is 15 minutes, the system should be successfully able to restore all its transactions which have occur till 11:45AM.

Note: RPO and RTO is defined by the financial impact to the business when systems are unavailable.

* Disaster Recovery Plan for Commute from AMI backup aspect:
* AMIs can help in restoring failed instance quickly.
* AMIs can be used within the context of the autoscaling to scale out DR site on an event of primary site failure.
* AWS recommended to maintain a library of preconfigure AMIs, including the application stack, so that on an event of a disaster these can be used to create new (EC2) instances to replace the old/failed once.
* Keep the AMIs available on DR sites.
* Disaster Recovery Plan for Storage from S3 Fault tolerance aspect [FOR OBJECT STORAGE]:
* Can be used as primary object storage.
* Highly redundant, with 99.99999999999 (11 9’s) redundancy rate. Objects stored within S3 buckets are automatically backed up in multiple facilities within the region.
* Additional security can be provided within S3 bucket for ensure high degree of data retention through MFA for delete operation, versioning, cross region replication – for auto backup in DR site, bucket level policies & object level policies.
* One can also use S3 Glacier storage class for archiving/storing data at very LOW COST, however RTO needs to be set to meet the recovery time as object archive in S3 Glacier takes longer time (usually 3-5 hours).
* Disaster Recovery Plan for Storage from EBS Fault tolerance aspect [FOR BLOCK STORAGE]:
* Point-in-Time snapshot can be created to backed up EBS content into S3 bucket.
* Snapshot created can be used to create a new EBS volume & then connect it the EC2 instance to replace failed EBS volume.
* EBS volume data are stored in different system within the Availability Zone (AZ), this provide protection from single system failure within AZ.
* Disaster Recovery Plan for Storage from Storage Gateway prospective
* Storage gateway provides an easy means to backup/store on-premises data on to AWS cloud. Customer can download VMware image from AWS console, and install a VM instance from the same on premises which will provide NFS/iSCSI interface for backup/storage.
* Storage Gateway – File Gateway (NFS interfaces) [OBJECT STORE]: files are asynchronously backed up into S3 bucket, there is a one-to-one mapping between on premises files and S3 objects. User can ALSO access the S3 objects(files) from the S3 bucket. S3 sub resources like lifecycle policy, versioning etc. can be implemented on the uploaded file(objects).
* Storage Gateway – Volume Gateway - Cached | Storage Mode (iSCSI interface) [BLOCK STORE]: Unlike File gateway, in case of the volume gateway, on premises files are stored into S3 bucket as BLOCK store. There are two possible operating modes – *Cached Mode* where frequently use files are cached on-premises while its asynchronously backed up on S3 bucket as volume store, in *Storage Mode* there is NO local file store everything is backed up on S3 as EBS snapshot. One can’t access the files stored in S3 directly – to access the files one need to create EBS volume from form the snapshot (*in case of cache mode, first snapshot needs to create from the volume store prior to creating EBS volume*).
* Storage Gateway – Tape Gateway: it provides an iSCSI VTL (Virtual Tape Library) interface to backed up on premises files into virtual tape data store within S3 bucket or can be achieve into S3 Glacier. It consists of virtual media changer, virtual tape drive, and virtual tapes.
* Disaster Recovery Plan for Storage from AWS Import/Export prospective:
* If large amount of data needs to migrate (backed up) quickly into AWS one can use AWS import/export feature – solution like snowball, snowball edge and snowmobile can be leverage for the same.
* Rule of thumb – “*if the data takes more than week to transfer over the available connection (VPN/Direct Connection/Open internet), then better use AWS import export instead*”.
* Disaster Recovery Plan for Storage from VM Import/Export prospective:
* Ease means to transfer VM image (hypervisor base virtual image) from/to AWS.
* This can be helpful in creating DR site at AWS or on premises.
* NOTE: VM Export is ONLY AVAILABLE to those instances which are initially brought in using VM import feature. AMI based EC2 instance cannot be exported using AWS Export feature.
* Disaster Recovery Plan for Storage from RDS Fault Tolerance aspect:
* RDS Multi-AZ: Primary-to-Secondary synchronous data replication within a region. On an event of any failure the primary DB instance DNS entry will be automatically swapped with secondary DB instance. Help in archiving zero downtime during system patching/upgrade.
* Read Replicas: For read intensive applications, a read-only DB instance can be provide with asynchronous data replication from the primary instance. On an event of any failure read replicas can be promoted
* Automated Backup: Automated backup in conjunctions with transaction logs to help recover failed DB instance with RPO up to 5 min.
* Manual Snapshot: this are necessary for backing up a DB instance OR restoring a DB instance in another region.
* Can be synced from on premises database to RDS and vise-versa – failover from the on-premises to AWS OR AWS to on-premises is possible.
* RDS instance size can be upgraded to a bigger size BUT can’t be downgraded to a smaller size.
* For DynamoDB one can copy the data into a S3 bucket and using data-pipeline it can be copied to another region DynamoDB. Also, one can use cross-region replication of the DynamoDB to replicate the data into another region DynamoDB table. For Sync operation (preparation phase), the dynamo DB can be created with lower read-write capacity, during recovery phase the capacity can be altered to meet the production demand.
* For redshift database: During the preparation phase one can create snapshot of the database in to S3 within same/different region, during the recovery phase redshift cluster can be (re)created from the copied snapshot.

**01/04/2020**

* Disaster Recovery Plan for Networking, on an event of a disaster the ability to quickly shift the network settings from the production site to the DR site is very much required.
* Route53 AWS managed service helps in quickly change and restore network connectivity (routing) on an event of a failure.
  + - Route53 is an AWS managed, highly available and highly scalable global service.
    - Route53 automatic Health-check configuration can route incoming traffic to healthy site.
* Using Elastic IP (floating IP): these are the static IPs that can be configured to the system, on an event of any disaster these IPs can be swap with the DR instances to quickly resume the production workload.
* Elastic Load balancing:
  + - It can load balance the traffic within a given region
    - Elastic IP addresses can be allocated to the ELBs, which make the transition simple on an event of any disaster. (DNS can configure to pass the load to the elastic IPs, which can load balance the traffic among healthy instances).
* Amazon VPC: On an event of a disaster Amazon VPC can help in extending the network topology to the cloud. This is mostly applicable in extending the on-premises application to the cloud during disaster.
* Amazon Direct Connect: This helps in connecting the AWS cloud network to the on-premises network by providing a low latency, high bandwidth dedicated physical connection.
* Disaster preparation phase: Before disaster strikes, the stage where the disaster/backup plans are made are called disaster preparation phase.
* Disaster recovery phase: After disaster strikes, the steps that needs to be taken to recover from the disaster is call disaster recovery phase.
* Different Strategies for Disaster Recovery are: These strategies can be mix and match to find a right solution that meets the RPO and RTO expectations.
* **Backup & Restore** 
  + - **Backup solution:** Snapshot | Object (file copy) | Volume copy.
    - **Storage solution:**  S3|Glacier | Tiring solution (according to the usages storage class will be selected).
    - **Data transfer solution:** Over internet using custom solution| AWS File Gateway | AWS Volume Gateway – Stored/Cached mode | AWS Tape Gateway | AWS import/Export | AWS VPN Connect |AWS Direct Connect
    - **Key Consideration:** 
      * Select appropriate means: tools/services for backup/storage/data transfer solutions.
      * Define retention policy – till when a backed-up objects/files/snapshots needs to be stored? How soon they need to be made available? Can they be archived?
      * Ensure adequate security measures are applied to secure backed up contents.
      * Plan regular game-day/dry run to ensure the data are correctly backed up and can be restore correctly when needed.
* **Pilot Light:** DR strategy where a minimal version of the cloud is always running on the DR site. During preparation phase: The core component will be identified and a minimal version of the solution will be running on the DR site and will be in Sync with the production. During restore phase the other components on the solution can be quickly provisioned and the production traffic will be routed to the DR site. RDS site may be upgraded to meet the production traffic.
  + - **Key Consideration** 
      * Backed up AMI can be use to provision the remaining component.
      * Running Instance size should be upgraded to meet the production demand.
      * Add resilience to the DR Site: Plan for fallout instance failure.

**01/06/2020**

* **Warm Standby:** Scale down version of the fully functional version of the application will be running on the DR site. On an event of any disaster, production traffic will be automatically route to DR site and the application will be automatically scaled up to meet production requirement.
  + - **Key Consideration**
      * Increase the size of the EC2 instance fleet (horizontal scaling).
      * Increate the sized of the EC2 instance (vertical scaling).
      * Implement Route53 health checks to automatically route production traffic to DR site, on an event of a disaster. OR manually change the DNS entries of the Route53 to route production traffic to DR site on an event of a disaster.
* **Multisite:** In case of multi-site (active-active) DR strategy, a fully functional redundant site will be running in parallel to the production site. ON an event of any failure production traffic will be severed by the DR site.
  + - **Key Consideration**
      * Configure weighted or equivalent routing mechanism to distribute production load between production site and DR site. (manually overwrite the Route53 DNS configuration to route all traffic to DR site)
      * Based on the RPO, synchronous or asynchronous replication need to be incorporate.
* Data Replication: For data replication following key things needs to be keep in mind.
* Distance between the sites – larger the distance more latency & jitter.
* Availability of the bandwidth – How two sites are connected? Type of connection VPN/Direct Connection etc.
* Data rate required by the application – it should be always lesser than that of the available bandwidth.
* Replication methodology: serial or parallel connection for replication.
* Synchronous (*150 ms latency*) or Asynchronous replication: Synchronous replication data are automatically update; Asynchronous replications are eventual consistent.
* Self-Healing: The following services are self-healing services.
  + SQS (Simple Query Service) can help in decoupling the applications which can help in automatic reprocessing of the failed instances without any human interventions
  + CloudWatch with EC2 instances autoscaling, which can automatically detect un healthy instance using CloudWatch metrics and terminate unhealthy instances.
  + EC2 autoscaling instances – which can be use to replace the unhealthy instance with healthy once.
  + AWS Glacier/S3 perform integrity check and ensure that all objects are constantly availability across different S3 systems.

# **Section 4: Design for operational complexity and cost control**

**01/09/2020**

* Possible AWS Infrastructure configuration from account prospective are
  + One Account for the company, and individual account per employee. [Typically situated for small company.]
  + One Account for the company, and IAM user per employee. [Seen mostly on mid-size companies.]
  + One Account for the company, and account per company department. [Most common.]
* For above possible infrastructure configuration, one can use one of the following bill consolidation process:
  + Manage separate bill for each account [Maintain separate payer account}.
  + Have consolidated bill for each of the linked account, into a master account (payer account). [single consolidated billing.]
* Advantages of having a consolidated billing are:
  + Single bill – need not to deal with each account/department individually.
  + Ease tracking – per account level usage
  + Ease of user maintenance & standardization across all account(department) of the organization.
  + In case of consolidated billing, AWS consider all linked account as one single account (master account), where one can take cost benefit as AWS services are priced in tire/usages with increasing usage unit price reduces.
* **AWS Cost Explorer**
  + Helps in viewing AWS cost as a graph
  + Filter graphs as per – Region, Availability Zone, AWS service, API Operation, custom cost allocation tag, Amazon EC2 instance type, purchase options, usages type, usage group and more.
  + Cost can be filtered per linked account, in case of consolidated billing.
  + Cost forecasting base on historical data.
* **AWS Budget** 
  + Two budgets free per account, up to 20,000 budgets can be configured. Fee will be applicable for budgets exceeding free budgets.
  + AWS Budget helps in planning & managing service usages, service cost, and instance reservations.
  + Following things can be viewed with ease in the AWS Budget
    - Account usage to date, including how much capacity of reserved instance has been utilized.
    - Current usages against the budgeted usage or free tier limit.
    - Predicted usage/bill at the end-of-the month.
    - SNS notification can be configure which can notify when the forecast exceeds the budget cost per account/service OR when the usages exceeded the budgeted cost.

Different Types of budget

* + - **Cost Budget:** Plan how much is plan to spend on a service
    - **Usage Budget:** how much is plan to spend on a single/multiple service together.
    - **RI (Reserved Instance) Utilization Budget:** Define utilization threshold and received alerts when the utilization is below the threshold limit, this protect from under-utilization of reserved instance (RI).
    - **RI (Reserved Instance) coverage Budget:** Define coverage threshold and received alert when the number of instance hours fall below the budgeted hours. This ensures that RI instance hours are in aligned with the budgeted hours.
  + Linked account can create budgets of their, using IAM policy one of the member account (lined account) can be granted access to read/create/edit/delete budget in other linked account – this is typically done to provide full control to financial department (linked account) to manage other account/master account budget – using AWS organization one can provide/revoke access to linked account from access AWS budget.
  + AWS cost explorer helps in visualizing the costing using different graphs - if cost explorer is NOT ENABLE, it gets automatically enabled when the budget is created for the very first time.

**Best Practice Recommendation from AWS**: Its recommended NOT to run any AWS services in master account, with ONLY exception to host a single S3 bucket where all invoices/bills are store for future references.

There are two ways to organize infrastructure from Billing prospective: a) using AWS organization & consolidate billing b) having a single account with multiple VPCs for each individual group.

|  |  |
| --- | --- |
| **Consolidated Billing** | **Single Account with multiple VPCs.** |
| Easier form AWS architecture prospective | Simple billing, one bill for everything |
| Volume discount – all discount related to all linked account can be better mange when consolidated | Per account discount – easy governance. |
| Need IAM policy account accounts to view manage account bill/budget | NO IAM policy needed. |
|  | Complex policies, to restrict/allow resource level access. |
| Tag is complex, as it needs to be done across different accounts. | Tagging is simple, as it needs to be managed within a single account. |
|  | Setting VPCs get completed from costing prospective. |

Note: In case of consolidate billing by default Reversed Instances are shared within all linked accounts. This can be disable if desired (Reserved instance sharing = turn on/turn off from the billing and preference page) by the master account (payee account).

**01/15/2020**

* + AWS CloudWatch can help in creating alert based on the usages and can send alert notifications for
    - When a set threshold of a service usages is reached as per the threshold defined in the billing alerts.
    - When the usage exceeds the amount that is defined in the billing alerts.
    - When signed up, it can also send alerts/notification when the pricing is changed.
  + AWS Tags are the key value pairs which can be used for organizing and categorizing AWS resources. There are two try of tags a) system (AWS) generated and b) custom tags a tag cannot be deleted or merged. Logical grouping of the resources is useful in – projects, cost centers, development environment, application, department.
  + Resource group: Group of AWS Resources that share one or more common tags ***within a same AWS region***. Resource group can be of two types – tag base and CloudFormation stack base. Resource group can have nested group from the same AWS region. They can be used for cost exploration or for other bulk AWS operations.
* **AWS Cost Optimization** 
  + Right Sizing, selecting the right EC2 instance family and size. This needs to be done based the performance benchmarking (based on workload utilization, CPU, RAM, networking , storage size, and I/O – which storage classes to be selected , should be selected busting throughput or provision throughput).
  + Continuous monitoring & tagging
  + Leverage AWS service for cost optimizing by finding right (new) services.
  + Cost explaining for cost optimizing
  + EC2 instance purchasing option – OnDemand EC2 instance are most expensive. Don’t use it forever, use it for a short span of time (during initial phases) before shifting to the Reserved instance. (One of the reasons for choosing OnDemand instances is when the instance type is new/high demand and availability cannot be guaranteed within a specific AZ). **[RI applicable for the Availability Zone OR a particular Region – base on the choice instance will be reserved]. [One can also have scheduled reserved instances – where instance will be reserved for a specific scheduled.]**
  + **Dedicate Host:** Where a specific host is reserved for the client, NO other client instance will be running on that host. This helps in saving by bringing your existing per-socket licenses, per-core licenses or per-VM licenses.
  + **Dedicated Instance**: Pay per hour for the instance that are running on single tannate hardware.
  + **Capacity Reservation:** reserve capacity for your EC2 instance in a specific availability zone for any duration.
  + **Reserved Instance Attribute:** 
    - **Instance Type:** EC2 instance type for example m4. large – instance family/size.
    - **Scope:** The availability of the instance scope – within region or within zones (RI limits - **20 max zonal reserved instances and 20 MAX for each of the availability zone.**)
    - **Tenancy:** it’s a single tenancy (dedicated) or multiple tenancy.
    - **Platform:** Windows or Lunix

**Offering Class:** There are two offering classes for Reserved Instance, a) standard and b) convertible. Standard instance can ONLY be modified but CAN’NT be exchange for another Reserved Instance (RI) with different RI attributes. In case of convertible RI, they can be exchange for other RI instance with different RI attributes, like standard offering classes convertible offering classes can also be modified. Standard offing class instance can be sold in market place BUT convertible offering class instance can’t be sold in AWS market-place.

**Once a reserved instance is purchase, it cannot be returned – it can only be modified, exchange, sell.**

* + **AWS Spot Block Instances:** Spot instance which can be blocked for a definite duration (1-6 hours). The pricing is dependent upon the request duration and the availability of the capacity. Instance get automatically terminated at the end the duration or can be manually terminated. NOTE: using RequestSpotFleet API function one can launch a large number of EC2 sport instance at same time. To run RequestSpotFleet one needs to provide Target Capacity, Maximum Bid price, Launch Specification(AMI Id, VPC, subnets or availability zones, security groups, block device mappings, user data, and so forth) , IAM Fleet Role
  + **Elasticity**
    - Implement autoscaling – horizontal scaling of the instance when needed scale out and when NOT needed scale out.
    - Non-production workload – start when needed and shutdown (*deprovision)* when not needed.
    - Go Serverless where possible – pay ONLY for the used computation.
    - Use managed services instead of EC2 workloads.
* **AWS Organization:** AWS organization is a service for managing accounts at scale. **It’s a global service, can be reached from US EAST url.** 
  + **AWS OU are used for centrally managed accounts, automated creation of the AWS accounts, OUs, and Hierarchy AND can be used to enforce Service Control Policies.** (*AWS OU does not manage or control cross account permission; it’s been done by the IAM policies*). AWS OU can override permission that are otherwise granted by the IAM policies.
  + The changes made in organization unit have eventual consistency.
  + **AWS Organization Unit is a container of other Organizational Unit (OU) or AWS accounts.**
  + **OU Policy** – are the rule/policy to control AWS services. If the **Service Control Policy** is attached to the root of the OU then the policy is applicable to all the OU. If the **Service Control Policy** is applied ONLY to the AWS account then that is applied only to that particular account.
  + **Feature of AWS OU**
    - **Create Account – OU can be used to create member accounts**
    - **Add new accounts – OU can invite new member account to it**
    - **Can apply service control policies to the member accounts**
    - **Consolidated single billing for all the member AWS accounts**
    - **Manage Hierarchical grouping of the AWS account as per – budgetary needs, security needs, compliance needs. [ONE AWS OU can have up to 5 nested OUs]**
    - **Manage AWS Service / individual API action for the member accounts, once applied this will override the IAM policy access provided by the member account administrator.**
  + **Mode of operation for AWS Organization** 
    - **Consolidate billing** *(default feature).*
    - **All features** *(additional feature like service control policies are applied in addition to the consolidate billing) – Once a member account accepts to join a OU request sent by the Master Account by default all OU features will be enabled for the member account. This is primarily done to implement restriction applied from the master account using service control policies. Existing consolidate billing member, can be send fresh invitation to join for full feature mode.*

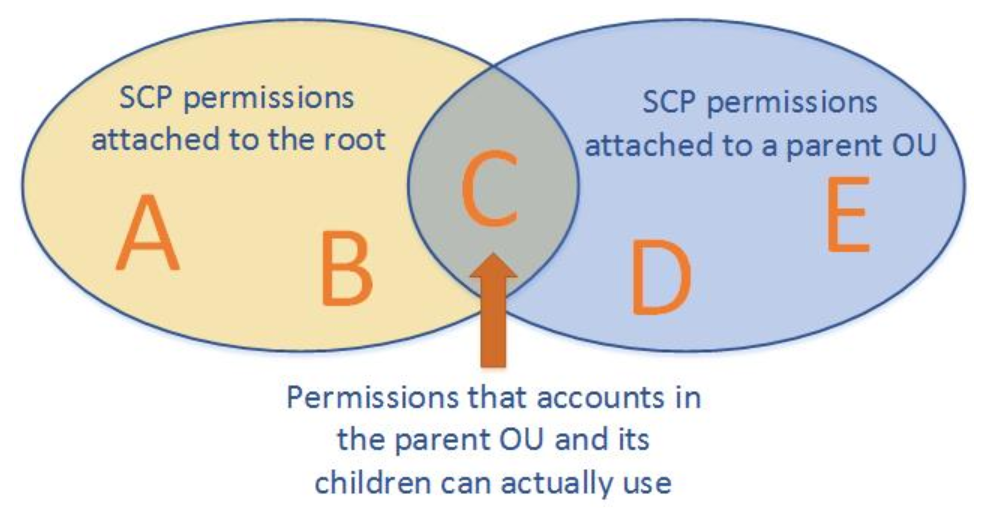
**Service Control Policies (SCP)– can prevent member account from accessing specific AWS services or even can block uses from leaving from the OU. Service Control Policy defines the services that can be used within the member account, its left to the IAM to provide required access to access AWS services that are granted by the Service Control Policy for that particular account.**

**The master account of the organization is not affected by any SCPs that are attached either to it OR any root OR any OU master account is part of. [Master account is immunized by the SCP] AWS organization attaches an AWS managed SCP FullAWSAccess to all the root, OUs, and account which ensure building of the organization. In order to restrict access to an OU or an account replace FullAWSAccess policy with a custom polices with limited permissions.**

**There are two possible ways to manage member account AWS services**

**- using SCP whitelisting – in this SCP defines the services that can be access within the member OUs/Accounts and rest other services are explicitly gets denies.**

**- using SCP blacklisting (default) - in this SCP defines the services that are denied within the member OUs/Accounts and rest other services are implicitly allowed.**

****

* + **AWS Organization Integration with other services** 
    - **AWS CloudTrail:** CloudTrail from different AWS member account can be maintained in the master account, this will ensure that all trails are centrally logged.
    - **AWS CloudWatch:** CloudWatch can be configured to store all metrics from all member account into one single account. This will enable sharing of the event across organization.
    - **AWS Config:** Organization wide compliance status can be maintained, by sharing AWS Config across organization status.
    - **Artifacts:** All compliance report can be made available across organizations, anyone from the AWS organization can view the report when needed.
    - **AWS Firewall Manager:** Centrally all WAF configuration can be maintained from one single AWS account across AWS organization.
    - **AW Directory Services:** Integration of the AWS directory services, allows seamless sharing of the directory service across multiple accounts and VPCs within those account across AWS Regions.
    - **AWS License Manager:** The integration allows for cross-account discovery of computing resources throughout the AWS organizations to ensure licenses compliance of the resources in the organization.
    - **AWS RAM (resource access manager):** This helps in sharing AWS resources cross account. **(example sharing subnet, transit gateway etc).**
    - **AWS Catalogue:** Integrating AWS catalogue can help the organization to maintain a single portfolio of services hosted access the organization.
    - **AWS Single Sign** on across different account within an AWS organization.
  + **AWS Service Linked-Role (**AWSServiceRoleForOrganizations.**):** *This is an AWS managed role (i.e. one can’t edit or add policies to this role),* which helps the master account to enable trusted services in member accounts. When a new account is created in the organization OR a member account accepts an invitation to join the organization this role gets provision in the member’s account. ONLY AWS organization can assume this role. This role is NOT NEEDED if the organization is operated ONLY for consolidate billing the role can be deleted, later if all feature (organization all feature mode) needs to be enable the role needs to be restored back. Once organization all feature mode is enabled, then the AWS Service linked role can’t be deleted. AWS Service Linked role are excepted from AWS Organization Service Control Policies (AWS OU SCP). AWS OU SCP never effected AWS Service Linked Role.
* **AWS Organization best practices** 
  + Use AWS cloud trail to monitor and log API Calls in the mater data S3 bucket.
  + Don’t add AWS resources to AWS Master Account, except for those which are absolute necessary.
  + Follow Least Privilege Principle.
  + Provide AWS OU SCP at OU level then doing it at the account level. Policies applied at the OU level, automatically inherited by the accounts so one didn’t have to add policies to each individual account within the OU.
  + Avoid have SCP at the root level, this will bar all the accounts to avail the service.
  + Use whitelisting OR blacklisting, DON’T mix whitelisting and blacklisting – as it creates confusion during debugging.

# **Section 5: AWS Services and Strategies for Deployment and Operation/Monitoring Management**

**01/17/2020**

* **CloudFormation** 
  + Infrastructure as cloud – create /update/delete/version control “***stack***”, collection of AWS resources combines together in a single template.
  + Simplify Infrastructure management
  + Quick replicated of an Infrastructure in different region
  + CloudFormation will rollback stack creation, if any of the AWS failed to provision.
  + “***Change Set***” will hold ONLY the modified delta of the CloudFormation template, one can review the change and decide upon applying it OR rejecting it.
  + ***Deletion policy*** will help in retaining specific resource/information when a specific template is deleted. E.g. when deleted an EBS volume one might take a snapshot to retain its data, similarly for RDS instance before deleting an RDS instance one may like to take a snapshot.
* **Cloud Formation Template Anatomy**

|  |  |
| --- | --- |
| **CloudFormation Template Header** | **Description** |
| AWSTemplateFormatVersion (Optional) | The version of the CloudFormation Template usually defined by the version date. |
| Description (Optional) | Short description of the template, that define the stack. |
| Metadata (Optional) | Metadata of the template which can be added to describe the template. |
| Parameters (Optional) | Value that are passed to the template are called Parameters. One can refer the Parameter section from the resource and the output section.  Sample Parameter  "Parameters" : {  "InstanceType" : {  "Description" : "Amazon EC2 instance type",  "Type" : "String",  "Default" : "m4.large",  "AllowedValues" : [ "t1.micro", "t2.micro", "t2.small", "hs1.8xlarge", "cr1.8xlarge", "cc2.8xlarge", "cg1.4xlarge"]  } |
| Mappings (Optional) | Custom define Key/Value mapping which can be use to specific conditional parameters. It works like a lookup table  One can easily map the value matching to the key using intrinsic function fn:FindInMap in both resource and output sections. |
| Condition (optional) | Condition that control whether certain resources are created OR whether certain resource properties are assigned a value during stack creation or update.  For example: for test environment multi-AZ setup is not desired, this can be achieved through CloudFormation condition element. |
| Transform (optional) | This is to use for serverless applications (aka Lambda based application), to specify the version of the serverless application module (SAM). On specifying Transforms element, one can use AWS SAM syntaxs to declare resources in the template. |
| **Resource** (**required**) | This is required to specifies the stack resources and their properties for example – for specifying Amazon EC2 instance, s3 bucket. |
| Outputs (optional) | Describes the values that are returned whenever one views the stack properties. When one declares an output for an S3 bucket name, on calling aws cloudformation describe-stack CLI command it displays the name of the S3 bucket. |

* **Intrinsic functions:** This are the built-in functions which helps in managing the stack. Intrinsic functions can be used following places within AWS cloud formation template – Metadata, Resource and Output sections. It can also be used in update policy attributes. One can also use intrinsic function in conditionally create stack resources.

|  |  |
| --- | --- |
| **Intrinsic Functions** | **Description** |
| Fn::GetAtt | Fn::GetAtt returns the attribute of a resource in the template.Fn::GetAtt:[*logicalNameOfResource*, *attributeName*] |
| Fn::FindInMap | Fn::FindInMap returns the corresponding value mapped to the keys in a two level map that is defined under the AWS cloud formation mapping sections.  {"Fn::FindInMap" : [ "*MapName*", "*TopLevelKey*", "*SecondLevelKey*"]} |
| Fn::GetAZs | Fn::GetAZs returns an array of listOFAvailabilityZone for a given region  { "Fn::GetAZs" : "*region*" } |
| Fn::ImportValue | Fn::ImportValue returns the value of an output exported by another stack. This is typically used to create a cross-stack reference.  Things to keep in mind regarding the cross-stack referencing   * Cross-stack reference cannot be created across region (within region ONLY). * For each AWS account Export Name must be unique within a region. * One can’t delete a stack if it’s been referred by another stack. * One can’t modified/remove an output value that is reference by another stack. |
| Ref | The intrinsic function ref returns the value of the specific parameters or resources.  {"Ref": "*logicalName*"} |

* To further control resources within a stack one can add additional relationship and behavior of the resources specified in AWS CloudFormation template using following CloudFormation template attributes.
* **CreationPollicy**: TO delay the resource creation until other resource is created successfully (unit CloudFormation Template received specific number of success signal) or based on set timeout. Currently ONLY ***AWS::AutoScaling::AutoScalingGroup***, ***AWS::EC2:Instance*** and ***AWS::CloudFormation:WaitCondition*** supports CreationPolicy. One can use cnf-signal helper script to send signal back, notifying about the progress of the stack creation. (cnf-signal are helped script available***/opt/aws/bin/cfn-signal***)
* **DeletionPolicy**: If you want to retain (preserved) or create a snapshot (backed up) for resources before deleting the stack. This is also applicable to the stack update operation that leads to resource been deleted from the stack. (*Note: this is NOT APPLICABLE to the resources whose physical instance is replaced during stack update operation. For example, where AWS resource property is updated and cloudFormation template replace the resource with a new one.*)

**Deletion Policy Attributes**

**- Delete:** if delete attribute of the deletion policy is attached to any resource then resource will be deleted once the AWS CloudFormation stack is deleted. By default, if there is no deletion policy attached, then resource will be deleted from the stack. *However: Default deletion policy for RDS (AWS::RDS::DBCluster) is snapshot. For successfully deleting a S3 bucket, all the content within the S3 bucket needs to be deleted.*

**- Retain:** if retain attribute of deletion policy is attached to a resources AWS will NOT delete the resource(s), when AWS cloudFormation template stack is deleted.

**- Snapshot** snapshot attribute of the deletion policy can be attached to the AWS resources which supports snapshots like EBS volume, ElasticCache , RDS , Redshift etc. AWS cloudFormation creates a snapshot of the resource before deletion the resources.

* **DependOn Attribute**: Specifies the resources creation depends on creation of the other resources.
* **MetaData Attribute**: TO add more metadata to the resources.
* **UpdatePolicy Attribute**: How the resource updates need to be carried out.
* **UpdateReplacePolicy Attribute**:

**Nested Stack:** Moving the repeatedly using AWS resources within a main template into a separate stack within the main template and refer is where the common AWS resources are needed.

**Cross-stack-reference:** In order to improve the maintainability and ownership of the stack, standard infrastructure designs are grouped under single stack and been reference by other stack using cross-stack-references. For example, VPC networking related components can be placed under network-stack and the web-application stack can simply reference the resources from the output of the network stack which is maintained by the networking team.

**Stack-Policy:** The CloudFormation stack policy is a JSON document that defines what can be updated as part of a stack update operation. To set or update the policy, the IAM users or roles must first have the ability to call the cloudformation:SetStackPolicy action.

**AWS CloudFormation Best Practice**

**Planning and organizing**

1. Organize Your Stacks by Lifecycle and Ownership
2. Use Cross-Stack References to Export Shared Resources
3. Use IAM to Control Access
4. Reuse Templates to Replicate Stacks in Multiple Environments
5. Verify Quotas for All Resource Types
6. Use Nested Stacks to Reuse Common Template Patterns

**Creating templates**

1. Do Not Embed Credentials in Your Templates
2. Use AWS-Specific Parameter Types
3. Use Parameter Constraints
4. Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances – use CloudFront helper script cfn-init to install software and deploy application.

|  |
| --- |
| **Cloud Developer Kit (CDK):** is an open source framework to model and provision your cloud application resources through CloudFormation template. In CDK everything is a construct, developers can design, compose and share their custom resources that incorporate unique requirement. CDK can be model either in java or .net. |

* **Cloud Formation Best Practice – Planning & Organization** 
  + Organize Your Stacks By Lifecycle and Ownership
  + Use Cross-Stack References to Export Shared Resources
  + Use IAM to Control Access
  + Reuse Templates to Replicate Stacks in Multiple Environments
  + Verify Quotas for All Resource Types
  + Use Nested Stacks to Reuse Common Template Patterns
* **Cloud Formation Best Practice – Creating Template** 
  + Do Not Embed Credentials in Your Templates
  + Use AWS-Specific Parameter Types
  + Use Parameter Constraints
  + Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances
  + Use the Latest Helper Scripts
  + Validate Templates Before Using Them
* **Cloud Formation Best Practice – Managing Stacks** 
  + Manage All Stack Resources Through AWS CloudFormation
  + Create Change Sets Before Updating Your Stacks
  + Use Stack Policies (*Stack policy helps in preventing resources from unintentional update of the resources*)
  + Use AWS CloudTrail to Log AWS CloudFormation Calls
  + Use Code Reviews and Revision Controls to Manage Your Templates
  + Update Your Amazon EC2 Linux Instances Regularly
* **Cloud Formation Best Practice – Securing Stack** 
  + **Limiting Access to CloudFormation Stacks with IAM – Limit the IAM user access to the CloudFormation API. *Who can do what?***
  + **Use IAM conditions for CloudFormation to limit the creation and update of the AWS resources using CloudFormation using** 
    - **cloudformation:TemplateURL** – using this policy one can ensure that all create/update/delete stack call are made by the user using a specific CloudFormation Template.
    - **cloudformation:ResourceTypes** – Using resourceType parameter one can enforce that what type of AWS resources are getting created/updated/deleted
    - **cloudformation:StackPolicyURL**– The CloudFormation stack policy is a JSON document that defines what can be updated as part of a stack update operation. To set or update the policy, your IAM users or roles must first have the ability to call the cloudformation:SetStackPolicy action. Using StackPolicyURL parameter one can enforce the stack policy that needs to be used

**PENDING – OPSWORKs Note be added later.**

**03/21/2020**

* **ElasticBeanStack:** AWS Elastic Bean Stack is an eassy to use service for deploying scalable web application and services development with supported programming languages on familiar servers like Apache, Nginx, Passenger, Docker, tomcat and IIS.

|  |  |  |
| --- | --- | --- |
| **CloudFormation** | **AWS Ops Work** | **Elastic BeanStalk** |
| Where the target is the create a low-level infrastructure across different AWS region. | Where the target to deploy application in AWS infrastructure with more control on the deployment using chef /puppet base scripts. | Where the target is to deploy application into selected servers with minimum maintenance possible and visibility to application health monitoring. |
|  | While onboarding docker image into Opswork, one need to define granular details to onboard the docker image to create Elastic Container Service. | While onboarding docker image into Elastic BeanStalk with minimal configuration required. |
|  | There is NO direct integration of Ops Work with other AWS services like S3 RDS etc. However, one can use service role to access other services from Ops Work. | There is a direct integration of Elastic BeanStalk with other AWS services like S2, RDS etc . |

* ElasticBeanStalk provides
  + Supporting of major programming language like java/go/ruby/.net/php/node.js/python.
  + Supporting different platform for running major programming language.
  + Supporting web containers like Tomcat / Passenger/ Docker etc.
  + Can also support custom build platform for hosting not supporting languages/platforms.
  + It also provisions the resources needed to run the application like EC2, load balancer etc.
* How to deploy an application on elastic bean stack.
  + Develop the application
  + Upload the application bundle, like .jar .war file etc.
  + Provide information about the application
  + Elastic Bean Stack automatically launch the application and provision the resources needed for the hosting of the application.
  + Once the environment is provision and the application is successfully launched on the provision resource, elastic bean stack help in monitoring the application and also deploy a new version of the application if needed. Application monitoring metric and environment status can be monitored through AWS management console or through AWS command line interfaces.

Note: Elastic Bean Stalk does not provide data persistence, within EBS. Once the EBS is stopped the data persistence within its resources will be lost. It’s ADVISABLE NOT TO host RDS instance within EBS instance as the RDS data as well as the snapshot will be lost once the environment is stop. One can host the RDS instance outside of the EBS and refer the same using the CName within the application or store the files within S3 buckets.

* For Dev environment, one can consider using the elastic bean stock as an alternative for test database, but recommended for PROD environment.
* AWS EBS resources can be accessible directly, however it’s advisable not to alter their configuration directly as it may become unusable.
* One can rebuild an environment which has been terminated with 42 days, once terminated environment is rebuild it creates a new resource with the same name, ID and configuration – however data stored earlier will be lost.
* **Elastic BeanStalk Application**: Application server as a container for the environment that run the web-application and the version of the source code, saved configuration, logs and other artifacts that one creates for using Elastic Bean Stack. (In case of Elastic Bean Stack, Application is conceptually compare to a folder.)
* Deleting an application is Elastic BeanStack Application, results in deleting the application source code, logs , configuration and all the version of that application.
* **Application Version**: Application Version is a part of an application which refers to a labeled iteration of deployable code for the web application. Application version points to a unique labeled deployable package of the source code store under S3 location. One can deploy an application version to an running environment Or create a fresh application version of the code and deploy the same to the environment.
* **Environment**: With respect to Elastic BeanStack an environment is a version that deployed onto AWS resources. An environment can deploy ONLY ONE version at a time, if multiple different version of the code need to deployed side-by-side it needs multiple environment.
* **Environment Tier**: there are two types of environment tier – *webserver environment* & *worker environment*. Webserver environment are the once that servers HTTP request, while worker environment are the once that servers pulling task from the Amazon SNS (Simple Notification Queue) for processing.
* **Environment Configuration:** Collection of setting for the underline AWS resource for hosting EBS Application, if the configuration changes – new settings are applied on the underline AWS resources or deploys (change) the underline AWS resources with the new settings.
* **Configuration Template:** Template for creating environment configuration for (EBS) elastic beanstack.
* **Elastic BeanStack –** web container is a component of the webserver that runs Java Servlets / JSP. Apache webserver is HTTP server that manages static pages, caching, redirection while tomcats supports webapplication features.
* **Elastic BeanStalk Host Manager (HM) –** agent deployed to EC2 instance by the Elastic BeanStalk for deployment and monitoring of the application. Following are the task that are performed by an EBS HM (host manager).
  + Deploy the application into EBS – EC2 resources.
  + Aggregate Event and Metrics for retrieval via console /API/ Command Line Interface.
  + Generate Instance level events.
  + Monitor Application Level Log file for critical errors. (this feature is not available on OpsWorks).
  + Monitor Application Server.
  + Patching Instance component.
  + Rotating Application log and publishing it directly to configure S3 bucket.
* **Elastic BeanStalk Custom Platform or Custom AMI** – If supported default pre-build platform does not supports the customer requirement customer can opt for using Elastic BeanStack Custom Platform feature where one can build their desired platform from scratch and use it within EBS for hosting custom application.

In order to start with the building custom platform, one needs to starts with building AMI from a supported operation system, then create an Elastic Bean Stalk platform using Packer an open source tool for creating machine images for many platform including AMI which can be used within EC2 instances.

When to use custom platform?

- for hosting legacy application which are NOT supported by the Elastic BeanStalk platforms.

- to reduce the startup time of the application, as Elastic BeanStalk needs to run multiple scripts to build the desired environment for deploying the application, ALL these configuration (scripts) can be move to the AMI instead of running it during the installation process they by reducing the startup time of the application.

**03/22/2020**

* **Elastic BeanStalk – for docker container base applications:** A docker is a standardized unit of software development, containing everything needed to run including library binary files, system tools, code and runtime. The main benefit of using docker container is the ability to quickly and easily spin lightweight repeatable environments using minimal codding.
* One of the possible ways to onboard a custom application into elastic beanstalk is to create a docker image and use the docker image to build the Elastic Beanstalk application.
* Environment Variable define within the Elastic BeanStalk are passed directly to the container image.
* Elastic BeanStalk infrastructure automatically takes care of the capacity provisioning, load balancing , auto scaling and application health monitoring.
* If the docker container with the Elastic BeanStalk crashed / killed, Elastic BeanStalk restart the docker container automatically.
* Docker Platform configuration available within Elastic BeanStalk
  + **Single container Configuration**: It can be use to deploy Docker image describe in docker file or Dockerrun.aws.json (optional) definition and source code EC2 instances running in an Elastic Beanstalk environment.
  + **Multiple container Configuration**: Use Elastic Container Service (ECS) to coordinate deployment of multiple docker containers to Amazon ECS cluster in Elastic BeanStalk environment. Instance in the environment each run the same set of containers, which is define in dockerrun.aws.json file (in this case dockerrun.aws.json is mandatory).
  + **Pre-Configure docker platform configuration** **with supported language**: There are many pre-configure docker platform that can be use to run popular combination software stacks like java with glassfish etc. This option does not provide means to configure the software that application runs on. However, to customized the preconfigure docker platform to install additional software that the application needs, one can add a docker file to the application root folder.
* Elastic BeanStalk integration with other AWS Service
  + By default, Elastic BeanStalk creates s3 bucket in the same region where EBS is running, this bucket will be used by EBS for storing object required for proper operation of the application. The buckets are NOT encrypted, but the same time secure as it does not have public access to it.
  + Elastic BeanStalk can be integrated with Elastic File System (EFS), it can be mount to instance across availability zone, which can use it like local storage drive without changing the code.
  + Elastic BeanStalk can be integrated with RDS, EBS instance can refer the RDS using CName value so that in case of multi-az RDS instance once the primary DB instance fails and it get switch over the secondary instance one don’t have change the db end-point.

NOTE: Don’t use the EBS generated security group to all access to the RDS instance, as it will create a dependency on the RDS security group, which will prevent it from deleting when the EBS instance is terminated. (All other EBS resources will be deleted on termination, but NOT the security group). Solution to this problem is to crate a new security group (outside EBS) allowing DB communication with the security group and attach it to both EBS EC2 instance and to the RDS instance, this will help in removing the dependency).

* + Elastic BeanStalk can access dynamo DB (created within or outside the Elastic BeanStalk).
* Monitoring in ElasticBeanStalk – default ElasticBeanStalk is monitored by Cloud Watch. Elastic BeanStalk can also be customized to send custom metrics to CloudWatch. CloudWatch then can be used for monitoring OR creating alerts to implement decisions or automatic notification based on the define rules.
* Elastic BeanStalk application logs are automatically send to CloudWatch, which can be configured to also forward to an S3 location or archived at S3 location. Each of the EC2 instance of the Elastic BeanStalk has an CloudWatch Log agent installed which helps in publishing metrics datapoints [custom datapoints] to CloudWatch to each configure Log Groups. [Each Log Group applies its own filter pattern to determine what log streams events to send CloudWatch as data points – each logstreams can be configured to have their own retention, monitoring, access control settings].
* Additionally, enhance monitoring can be enable for Elastic BeanStalk EC2 instance. Each of the AMI for Elastic BeanStalk comes with health agent installed on it which helps in supporting monitoring of the enhance health.
* Elastic BeanStalk can export log data from different log group to be exported to S3 location, a same S3 location be used to store logs from multiple different log group (source). In order to separate the log data , different prefix can be attached to each of the log groups. [additionally lifecycle policies can be created on the S3 bucket to archive the data into S3 Glacier Storage class.]
* Logging for docker container within Elastic BeanStalk : One needs to configure application to write logs into a specific folder and then those logs can be configured to send to S3 bucket.
* CloudTrail can be enabled to Elastic BeanStalk API calls – every call made to Elastic BeanStalk can be made to record in cloudTrails logs which can then be send to S3 bucket.
* Best-Fit to use Elastic BeanStalk
  + Quickly Deploy application for prototyping or for testing.
  + To deploy application to AWS with no or minimal knowledge of AWS infrastructure.
  + Migrating custom web application to AWS infrastructure.
    - Create a docker file including all application dependencies.
    - Create a docker image from the docker file
    - Upload the docker image to public or private docker image repository.
    - Deploy the docker image using Elastic BeanStalk as single docker image or multiple docker image.
  + To be used in software development project by the software developers.
* When NOT to use Elastic BeanStalk
  + When team has expertise on build secure/scalable AWS infrastructure, and needs more control on the AWS infrastructure.
  + Team needs to recreate the AWS infrastructure quickly across different AWS regions.