**AWS Whitepaper Notes**

Contents

[**AWS Security – Introductions** 1](#_Toc36387638)

[**AWS Security – Overview Process** 1](#_Toc36387639)

[**AWS Security – Lambda service overview** 3](#_Toc36387640)

[**AWS Best Practices - Security** 3](#_Toc36387641)

[**AWS Best Practices - Cloud Data Migration** 3](#_Toc36387642)

[**AWS Best Practices - Microservices** 4](#_Toc36387643)

[**Disaster Recover – Backup & Strategy** 9](#_Toc36387644)

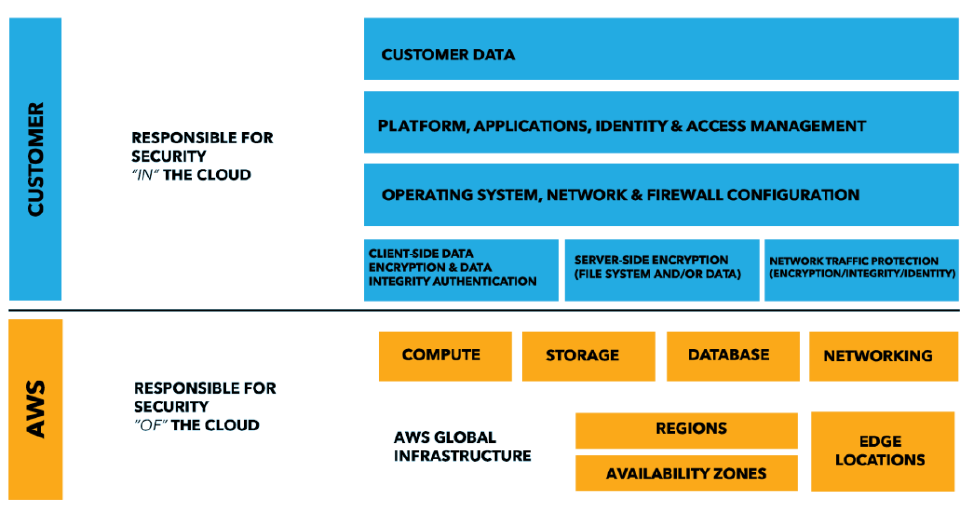
[**AWS Security Incident Reporting Guide** 12](#_Toc36387645)

# **AWS Security – Introductions**

To be added later.

# **AWS Security – Overview Process**

AWS Share Responsibility Model: Anything that put in the cloud customer is responsible for it, security of the cloud is the responsibility of the AWS. The amount of security varies between service and the sensitivity of the data. For all service basic security features are common like – SSL/TSL for data transmission, user activity monitoring etc. AWS takes care of the AWS cloud infrastructure and the security for its managed services like lambda functions, Amazon RDS, etc. For Managed services AWS takes care of the operating system patching, database patching, firewall configuration, and disaster recovery.



For customer, if they are availing IaaS service like EC2 instance then customers are responsible for

- Management of the guest operating systems (including update and security patching)

- Installation of the application utility and services

- configuration of AWS provided security firewall called as security group.

- customer is involved in the same task as they are managing their own servers irrespective of where it’s been installed.

For customer, if they are availing SaaS (AWS Managed services)

- AWS handles most of the responsibilities on behalf of the customers it takes care of the launching of the AWS manage services instances, patching of the guest operating system or databases, replicating of the database.

- Customer is responsible for managing user access and permissions for proper segregation of duties.

**AWS Security Compliance:** For AWS compliance responsibilities are also shared between customer and the AWS. AWS platform is aligned to the following security compliance standards

* SOC 1/SSAE 16/ISAE 3402 (formerly SAS 70)
* SOC 2
* SOC 3
* FISMA, DIACAP, and FedRAMP
* DOD CSM Levels 1-5
* PCI DSS Level 1
* ISO 9001 / ISO 27001
* ITAR
* FIPS 140-2
* MTCS Level 3

Along with the above list, AWS is also compliant of the following industry-specific compliance standards

* Criminal Justice Information Services (CJIS)
* Cloud Security Alliance (CSA)
* Family Educational Rights and Privacy Act (FERPA)
* Health Insurance Portability and Accountability Act (HIPAA)
* Motion Picture Association of America (MPAA)

Along with the above mentioned global and industry specific standards, AWS also provides wide range of whitepaper, reports, certifications, and other third-party attestation which can be download from the AWS site.

**Physical Security and Environmental Security:** AWS provide adequate security to protect Physical AWS datacenters – ONLY in need basis with minimum of two-factor authentications. AWS following protection for its own datacenters

1. **Fire Detection and suspensions**
2. **Power** – AWS always mating a two redundant power supplies**.**
3. **Climate and Temperature control.**
4. **Management**
5. **Storage device decommissioning:** Once a hardware (storage device) reach its end of life, AWS usages NIST (800-880) guidelines “Guidelines for Medial Sanitization” for decommissioning of the storage devices.

**Business Continuity Management: Page # 11**

# **AWS Security – Lambda service overview**

# **AWS Best Practices - Security**

# **AWS Best Practices - Cloud Data Migration**

<https://d1.awsstatic.com/whitepapers/Storage/An_Overview_of_AWS_Cloud_Data_Migration_Services.pdf>

There are different services to transfer data to/from on premises to AWS cloud. Each services offers different solution and various levels of speed, security, cost and performance.

Challenges related to cloud migration

**Security / Data Sensitivity**:

AWS Direct Connect: Secure dedicated line / IAM controlled access to the Direct Connect console/ Can be integrated with AWS Cloud trail to capture all the API calls made by the customer or on behalf of the customer.

AWS Import/Export Snow: IAM access control for accessing AWS Import/Export console. Integrated with KMS encryption for providing data encryption at rest. Use industry standard TPM (Trusted Platform Module) to prevent an unauthorized modification of the hardware, firmware, or the software to physically secure the AWS Snowball device.

Storage Gateway: Storage Gateway console secure by IAM access, Data transmitted through storage gateway are encrypted using SSL/TSL, Data stored withing the Storage gateway are encrypted using AES 256 encryption. Authentication between a storage gateways and iSCSI initiator can be authenticated using Challenge-Handshake-Authentication Protocol (CHAP).

Amazon S3 Transfer Accelerator: IAM can be used to secure access to S3 bucket, Bucket level/ Object Level access can be granted to individual IAM user or to another AWS account using access policies. Data stored withing S3 bucket can be encrypted using S3-SSE, SSE-KMS, SSE-Customer Key. Or using client-side encryption. Data transmitted to/from S3 are encrypted using SSL/TSL encryption. Additionally, multi-factor authentication can implement to provide additional security.

AWS Kinesis Firehose: IAM is use to restrict access to AWS Kinesis Firehose. Data transmitted through AWS Kinesis Firehose are encrypted using SSL/TSL encryption.

**Cloud Migration Tool: (both managed and unmanaged migration tools offered by AWS for lift-and-shift data to/from on premises).**

Below formula helps in calculating the number of days needed to transfer data of given size based on the availability of the.

|  |
| --- |
| *Number of Days = (Total Bytes)/(Megabits per second \** 125 *\** 1000 *\* Network Utilization \** 60 seconds \* 60 minutes \* 24 hours*)* |

**Page # 6**

# **AWS Best Practices - Microservices**

Microservice architecture is not a completely a new approach of software development instead it’s a combination of various successful and proven concepts such as:

- Agile Application Development

- Service Oriented Architecture

- API first approach.

- Continuous Integration and Continuous Delivery.

- 12 factor app, design pattern.

Microservice can be build using AWS Serverless technology, which along with various other benefits also leverage the benefits of the serverless framework like

- No infrastructure to provision or managed

- Automatically scale by unit of consumption.

- “*Pay for Value*” billing model.

- “Build in availability and fault tolerance”

Unlike traditional approaches, where monolithic application is built as layers of service. In case of microservices architecture, the functionalities are spitted into cohesive verticals, align as per their functional domain.

Microservice UI are mostly developed as single page JavaScript application hosted from Amazon S3 services, while static content is made available through Content Delivery Network (CDN) to reduce latency. To reduce chattiness and improve latency additionally caching mechanism can be built to cache frequently accessible items.

In AWS platform Microservice implementation are most done through AWS Lambda or through docker container with AWS Farget. AWS Lambda function helps quickly develop API to cater business requirements without worry much about the server provisioning. Another common approach is to host docker containers over AWS Farget. To reduce complexity of cluster management of the AWS Farget , one can leverage AWS ECS (Elastic Container Service) or AWS EKS (Elastic Kubernetes Service) where with help of a single API invocation one can launch or stop a Docker enable application and at the same time leverage some of the existing AWS services like ELB (Elastic Load Balancer) , EBS (Elastic Block Store), or IAM (Identity and access management).

AWS Farget can automatically support scaling of resources by launching thousands of containers to cater the incoming load. It also supports container placement strategy and termination task.

AWS EKS – is a managed service which run latest version of Kubernetes Service and all available plugins & tooling available from the Kubernetes communities, at the same time supports IAM feature to securing the application. Application running on any standard Kubernetes environment is fully compatible with the application running on AWS EKS.

Amazon Elastic Container Repository helps in storing docker image use in AWS ECS or in AWS EKS without any need for provisioning and hosting servers for container repository.

AWS Private Link helps in establishing connection withing the VCP and Private Link Supported AWS services (S3/Dynamo DB), AWS services hosted in separate AWS account and supported Marketplace partnered services without any need for Direct Connect, Internet Gateway, NAT devices, Public IPs. The traffic pass through AWS Private Link remains within AWS network & does not leave the AWS network. Private link is a great way to isolate microservices where each microservices can be hosted over a separate VPCs, scale is as per the need of the customer and make it available through AWS Private Link directly or through AWS Marketplace.

There are many options for data storage for microservice – AWS RDS for structural data storage, for high throughput application AWS DynamoDB can be used which is a NOSQL database. There are multiple options available for caching where frequently accessible data can be store to improve performance. DAX (DynamoDB Accelerator, provide caching in between the application and the DynamoDB.

Dynamo DB provide serverless offering, where read-write capacity need not to be guess in advance. Alternatively, one can also use the on-demand Dynamo DB.

**API Implementation**

Architecting, deploying, monitoring, continuously improving and maintaining an API is a tedious task. AWS API Gateway helps in hosting APIs without any need for provisioning backend servers for hosting APIs. AWS Lambda function can be integrated with the API gateway to develop microservice APIs, without any need for provisioning /hosting / scaling backend server resources.

Deploying microservice using docker image over AWS Farget without worrying about the underline infrastructure. In additionally to this for data storage one can use serverless AWS DynamoDB and AWS Aroura DB.

**Deploying AWS Lambda function**

For deploying AWS Lambda function, one can leverage AWS SAM (Serverless Application Module) which provides an easy way to define AWS Lambda function and other serverless resources. AWS SAM also provide SAM local which can be use to develop, analyzed and test Lambda function locally before deploying them into AWS Cloud infrastructure.

**Distributed System**

There are multiple challenges for a distributed system, like service discovery, data consistency, asynchronous communication, distributed monitoring and logging.

***Service Discovery:***

One of the most challenging tasks of a distributed system is service discovery. Along with the service discovery one also needs to needs to decide how to store service metadata, how to identify healthy services etc. AWS provided following option for service discoveries:

**DNS Base Service Discovery:**

Amazon ECS can be integrated with the Route53 base service discovery service (Route53 Auto Naming API) which helps discovery of services. Service name are automatically mapped to the Route53 DNS records – client service can use the service name to resolve server endpoint using DNS queries. One can also specify health check condition in the service task definition so that ECS returns ONLY the healthy service endpoints in response to the DNS query.

Additionally, one can also leverage AWS CloudMap capabilities that extend Route53 Auto Naming feature and provide API base service discovery mechanism with faster change propagation.

For services hosted in AWS EKS one can leverage unified service discovery for services hosted in AWS EKS.

Alternatively, Leveraging Third Party Software for service discovery is also possible in AWS – third party software like HashiCorp Consul, etcd or Netflix Eureka can be hosted within AWS infrastructure. AWS Quick Start supports launching of flexible, scalable HashiCorp Consul automatically.

**App Mesh:** To ease out the communicating between larger number of services, one can use AWS APP Mesh which creates an additional layer for handling inter-service communication. AWS Service Mesh handles the service discovery transparently, application developer need not to add additional application codding for service communication. AWS Service Mesh standardized the communication, adding end-to-end visibility to the service while increasing the availability of the service. APP Mesh can be integrated with existing or new microservices running on AWS Fargate, Amazon ECS, Amazon EKS or on self-managed Kubernetes on AWS.

**Distributed data storage**

Unlike in the case of monolithic application where application data are generally stored in a relational database with a common data model for all application layers. In case of the microservices there is a need for distributed data storage – each microservices should have their own data persistence layer. However, distributed data management pose few challenges.

In case of distributed data management system, among Consistency, Availability and Partition ONLY two can be achieved at a given time (CAP Theorem). Also, in case of a microservice the business transactions span across microservice making it difficult to implement AICD transaction, instead one needs to use multiple local transaction within microservices to redo a already processed transactions , in case the business transaction fails multiple local transaction needs to be perform to undo the successful transactions (SAGA pattern). *AWS Step Function* helps in SAGA patterns with ease.

For building centralized reference data, AWS lambda functions with schedule Cloud Watch Events can be used to clean up and manage duplication.

In context of microservice architecture, event sourcing enables decoupling of the services (different services of a same application) using publish/subscribe pattern where same event can be feed to different services with different data mode. This can be used in conjunction with the CQRS (Command Query Responsibility Segregation) where read workload is decoupled from the write workload in order to optimized performance, scalability and security. Amazon Kinesis Data Stream can be use to build centralized event store for enabling event sourcing within microservice running on different AWS platforms.

**Asynchronous communication & Lightweight Messaging**

In case of traditional monolithic application, the communication within different components is straight-forward, whereas in case of microservices communication within different components of a same application needs to pass through the network.

**REST-Based communication**

Most cases, a REST Based HTTP/S communication is preferred to communicate between different components of a microservice application. The REST architectural style relies on stateless communication, uniform interfaces, and standard methods. API Gateway can used as a front door for the backend APIs. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

**Asynchronous Messaging and Event Processing:**

In this type of approach, the communication between two microservice is achieve through a queue. Benefit of this type of communication is, there is NO NEED for service discovery and the services are loosely coupled preventing fan out failure to the downstream systems while making retries easy. AWS offers multiple services like Amazon SNS, Amazon SQS and Amazon MQ to implement asynchronous messaging.

**Orchestration and State Management:**

AWS offers AWS Step Functions service, which helps in developing an orchestration layer for distributed system. Step Functions provides a state machine that hides the complexities of service orchestration, such as error handling and serialization/parallelization. This also helps in scaling and make change quickly without any need to change the logic of underline microservices.

Step Functions is part of the AWS serverless platform and supports orchestration of Lambda functions as well as applications based on compute resources, such as Amazon EC2 and Amazon ECS, and additional services like Amazon SageMaker and AWS Glue.

**Distributed monitoring**:

In case of distributed system, monitoring each individual component is critical at the same time difficult to achieve because of the distributed nature of the system.

**Monitoring**

Amazon CloudWatch to collect and track metrics, centralize and monitor log files, set alarms, and automatically react to changes in your AWS environment. CloudWatch can monitor AWS resources such as EC2 instances, DynamoDB tables, and RDS DB instances, as well as custom metrics generated by the applications and the services or any log files that applications generate.

Another popular option–especially for Amazon EKS–is to use *Prometheus*. Prometheus is an open-source monitoring and alerting toolkit that is often used in combination with *Grafana* to visualize the collected metrics. Many Kubernetes components store metrics at /metrics and Prometheus can scrape these metrics at a regular interval.

**Centralized Logging**

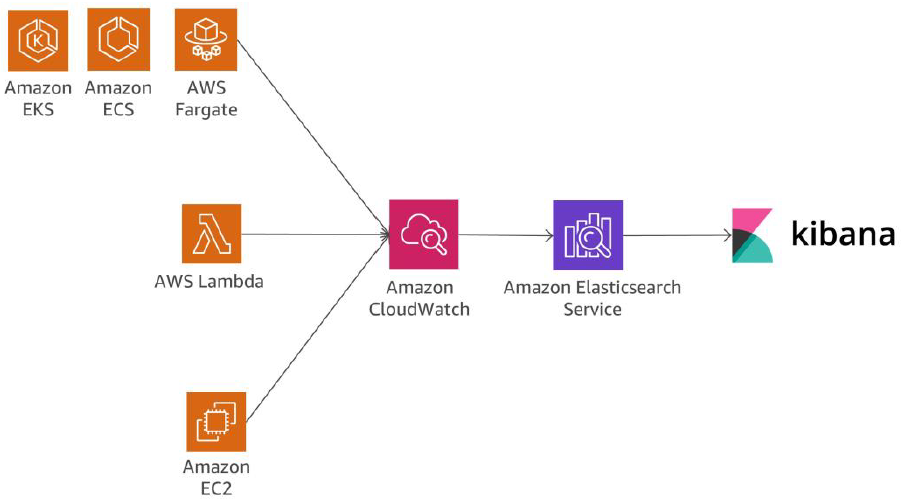
For serverless application its important to log in a centralized location. AWS CloudWatch provides an easy way to log centrally, by default lambda logs are store in AWS CloudWatch. AWS ECS provides support for awslogs log driver to route container log to AWS CloudWatch. For Amazon EKS, it is necessary to run FluentD which forwards logs from the individual instances in the cluster to a centralized logging CloudWatch Logs where they are combined for higher-level reporting using Elasticsearch and Kibana.

**Distributed Trace**

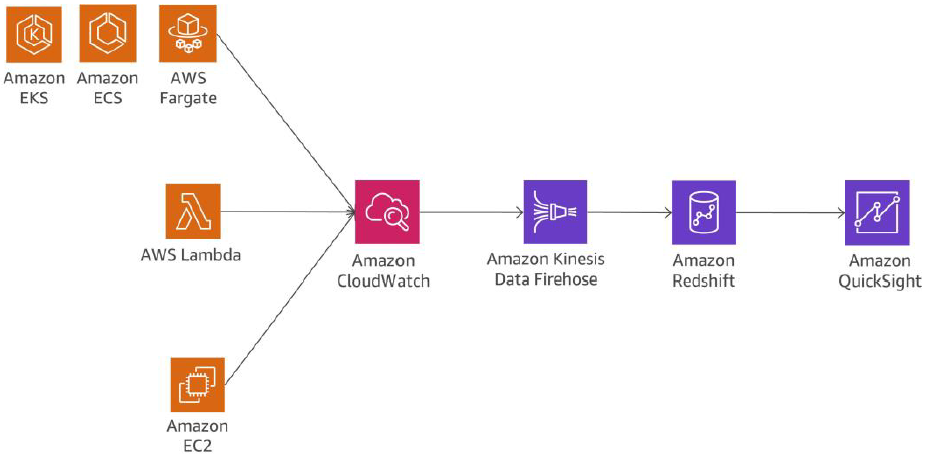
In case of microservice application as there are multiple services so its equally important to track the end-to-end logs for a business flow across all microservices. AWS X-Ray provides a single correlation ID which unique identifiers and attached to all requests and messages related to a specific event chain. The trace ID is added to HTTP requests in specific tracing headers named X-Amzn-Trace-Id when the request hits the first X-Ray-integrated service. With the help of correlation ID & trace ID, it’s possible to track end-to-end business flow.

**Log Analysis**

Searching, analyzing, and visualizing log data is an important aspect of understanding distributed systems. CloudWatch provide an excellent support for searching, analyzing, and visualizing log instantaneously. Alternatively, one can also use Amazon Elastic Search (ES) together with Kibana. Amazon ES can be used for full-text search, structured search, analytics, and all three in combination. Kibana is an open source data visualization plugin for Amazon ES that seamlessly integrates with it.



Another alternative solution for log analysis can be – Amazon Redshift together with Quick Sight. It’s also possible to stream logs into Amazon Redshift through Amazon Kinesis Data Firehose.



**Chattiness**

Due to the nature of the microservices, there are multiple component interacting with multiple other components. To keep the component interaction effective REST over HTTP/S is preferred due to lightweight nature, however extremely large volume can be problem. Implementing right catching can reduce the chattiness and latency, however a right balance needs to be set in order to balance the good caching rate and timeless/consistency of the data. AWS provided multiple caching option through AWS Elastic Cache; API gateway also provides in build cache layer to reduce the load on the backend service.

**Auditing**

Due to nature of the microservice, it difficult to track all users’ changes performed on each of the microservices. AWS Cloud Trail provided an auditing action on the services, any actions performed by the users are recorded on S3 bucket which can be later evaluated for auditing. Multiple microservices running on different AWS account can be configured to aggregate CloudTrail logs into a single S3 bucket for consolidated auditing.

**Resource Inventory and Change Management**

It important to monitor the changes and ensure that the changes are not violating the organization policies, in case of any policy violation appropriate alert needs to be raised to the concern teams to the address the event. For Resource Inventory and change management AWS Cloud Config provide an appropriate functionality which can be leveraged to track and mange infrastructure changes and immediately (automatically) react to the non-compliant changes.

**Conclusion**

Microservices architecture is a distributed design approach intended to overcome the limitations of traditional monolithic architectures. Microservices help to scale applications and organizations while improving cycle times. However, they also come with a couple of challenges that might add additional architectural complexity and operational burden.

AWS offers a large portfolio of managed services that can help product teams build microservices architectures and minimize architectural and operational complexity. This whitepaper guides you through the relevant AWS services and how to implement typical patterns, such as service discovery or event sourcing, natively with AWS services

**AWS Best Practices - Continuous Integration and Continuous Delivery**

# **Disaster Recover – Backup & Strategy**

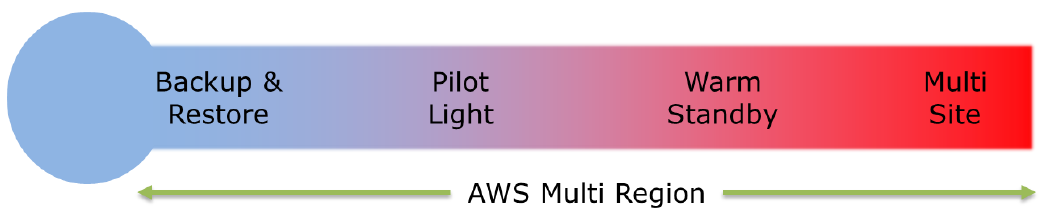
* RTO (Recovery Time Objective): The time it takes after a disruption to restore a business process to its service level, as defined by the operational level agreement (OLA). Time it takes to fully recover from a disaster (system failure)
* RPO (Recovery Point Objective): The acceptable amount of data loss measured in time. Point in time to which the data can be recovered.
* For defining designing backup strategy for backing up and restoring, identity each of the potential failure points and their business impacts, security and regulatory requirements, retention polices, RTOs and RPOs.
* Backup and Recovery strategy needs to be designed for
* File level recovery
* Volume level recovery
* Application level recovery
* Image level recovery
* Recovery strategy for different scenarios
* **Cloud Native Infrastructure** where the entire architecture is comprised of AWS services, then the ready built in feature of the AWS can be leveraged for the backup and restoration strategy. In this scenario – for file base storage leverage S3 bucket and for Volume backup one can leverage EBS snapshots stored in S3 bucket (even replicated to another region for high degree of fault tolerance). For cost optimization, incremental snapshots can be created where the first snapshot will have most of the data backed up and remains one will store the incremental changes.

Volume level Backup

* + Temporally unmounting the disk/volume to ensure consistent backup.
  + Flashing out the buffer memory in case of RAID setup
  + Using agent-based backup solution
  + Creating replica of the Primary volume (while doing this, one need to ensure that single large volume should be sufficient to address the maximum size required).

Database Backup

* + For backing up of the databases running on the EC2 instances – it can be done by creating backup of the data files using native methods /tools like EBS snapshots
  + For Large Databases which are built on RAID setup – one can offload the performance impact of primary db instance by taking backup from read replicas instead of primary db. The read replicas can have similar RAID configuration as that of the primary db OR can be set to consolidate all RAID volume into a single volume (provided required size of EBS volumes are available).
  + FOR RDS INSTANCES – manual and automated backup are available
  + AUTOMATED BACKUP FOR RDS INSTANCE: this can configure to take full daily backup at the define window set during db instance creation. Using automated backup snapshot in conjunction with the transaction logs, one can recover from any failure up-to 5 min in past. Automated backup can be retained up to 35 days.
  + MANUAL BACKUP (DB snapshot) FOR RDS INSTANCE: this can done to create a point in time backup which can be used to restore the DB or recreate a new DB instance with different endpoint.
* Image Backup
  + AMI can be created to store image which can be used to quickly provision/restore instance during any failure.
* AWS quota Elastic IP address is 5 Elastic IP addresses per account per region. Elastic IP address are regional constructs – they are confined to a single region ONLY.
* S3 durability 99.9999999999 11 9’s
* Essentials infrastructure components that needs to be consider for Backup & Recovery Strategy
* **Region:** Select multiple region, each of which are geographically separate global region, to ensure availability of one or more region during disaster event.
* **Storage:** Durable storage.
  + **S3 bucket:** Store content redundantly over multiple availability zone with high durability of 99.9999999999 11 9’s. MFA & versioning to avoid accidental delete. On premises data can be backed up into S3 through – Direct Connection, through AWS import/export portable devices, through internet. On an event of DR uploaded data can be quickly retrieved or can be used to mapped to a new instance.
  + **Glacier:** For archival storage.
  + **EBS volume:** Point in time snapshot can be created, independent of the instance lifecycle which can be stored in S3 to achieve high durability of the EC2 instance data.
  + **AWS import/export:** this feature can be used to import/export large amount of data to/from AWS using portable devices.
  + **AWS storage gateway:** This can be used to automatically backup on-premises data into AWS cloud using iSCSI protocol. One of the following variants can be used based on the need
    - **AWS Storage Gateway –** cached volumes
    - **AWS Storage Gateway –** store volumes
    - **AWS Storage Gateway –** Tape volumes
* **Compute:** Durable compute
  + **AWS EC2 instance:** compute instance can be created from store AMI (AWS machine image)
  + **AWS VM import/export:** this can be used to migrate & run on premises VM into AWS.
* **Networking:** In an event of disaster, network settings can be changed quickly to route production traffic from failed instance to DR instances. The following AWS components can help to quickly switch production traffic.
  + **Route53:** Highly available, AWS managed DNS service.
  + **Elastic IP address** – it can programmatically remap from the failed instance to new instance. Software licenses which are allocated to the MAC address , instead of remapping the elastic IP address Elastic NIC can be remapped.
  + **Elastic Load balancer** – it can automatically route the production traffic to healthy instance within the region. (*it’s a regional service, can’t route traffic outside the region*).
  + **AWS VPC:** Fully controllable data center within AWS region**.**
  + **AWS Direct Connect:**
* Database:
  + **AWS RDS –** supports, manual and automatic snapshots, multi-AZ configuration to prevent/recover from the failure.
  + **Dynamo DB –** fully managed service from AWS
  + **Redshift –** support manual and automatic snapshot which can be use to restore the failed instance within same or different AWS region.
* **Deployment Orchestration:** On an event of any failure, deployment orchestration quickly helps in restore failed instances using deployment automation and post startup installation and configuration.
  + **AWS Cloudformation:** Enable to provision AWS resources in orderly and predictable fashion. Entire configuration can be stored in a single file which can also be versioned.
  + **AWS Elastic BeanStalk:** Easy way to deploy and scale application on AWS, it’s a fully managed service on an event of any failure detection elastic beanstalk replace the underline services with healthy instance.
  + **AWS OpsWorks –** Chef base infrastructure as a code service which can be used in conjunction with AWS CloudFormation to automatically provision new stack in an event of an failure and replace the host in the newly created stack.
* Different Type of Disaster Recovery Strategy



* **Backup & Restore:** There is NO running DR site, primary site data are backed up on a remote site (*un-affected by the primary side failure*). On an event of a failure the backed-up data will be used for restoring the instances.

|  |  |
| --- | --- |
| **Preparation Phase** | **Recovery Phase** |
|  |  |

* **Pilot Light:** Minimum version of the environment is always running on the DR site. On an even of any failure, remain system can be recreated with ease and primary site can be restored.

|  |  |
| --- | --- |
| **Preparation Phase** | **Recovery Phase** |
|  |  |

* **Warm Standby:** A scale down version of the environment will be running on the DR site. In an event of any failure the DR system will be scale out to cater production traffic.

|  |  |
| --- | --- |
| **Preparation Phase -** | **Recovery Phase** |
| 1. Set up EC2 instances to mirror data 2. Create and maintain AMI for scaling up of the new instances. 3. Run AWS resources with minimal footprint. 4. Patch and update stand-by instance to keep in sync with the production environment. | 1. Scale up EC2 instance (horizontal scaling) or use a large EC2 instance (vertical scaling). 2. Manually change the DNS entry in the Route53 or use Route53 health check to route traffic to the new instance(s). 3. Scale DB layer to handle the scale up load. |

* **Active-Active:** In case of multi-site active-active configuration, fully functional production ready instances will be running in parallel on the DR site. On an event of a failure the production traffic will be route to the alternative site. **Lowest RTO, highest DR maintenance cost.**

|  |  |
| --- | --- |
| **Preparation Phase** | **Recovery Phase** |
| 1. Set the DR site equivalent to the production site. 2. Set the Route53 weight to send no/small traffic to the DR site while sending full/major traffic to the | 1. Manually change-over to the DR site 2. Use autoscaling to right size the instance fleets |

* Key factors to be consider while replicating the data
* Distance between the sites – larger the distance more latency and jitter
* Availability of the bandwidth –
* Data rate required by the application – should be lower than the available bandwidth.
* Replication technology – Synchronous Replication / Asynchronous Replication Should be done in parallel to optimize replication.
* Fallback Strategy – Once the primary site is back, the traffic needs to be routed back to the primary site from DR site. Based on the different DR strategy the step will differ

|  |  |
| --- | --- |
| **For Backup and restore Strategy** | **For Pilot light / Warm Standby / Active-Active** |
| 1. Freeze the data change on the DR site. 2. Take backup of the DR site 3. Restore Primary site using the DR site 4. Unfreeze the data change on the DR site, ensure DR site is getting backed up by the primary site data. | 1. Use reverse mirror of the data from the DR site to the primary site. 2. One Primary site is restored completely. 3. Route the traffic back to the primary site |

* Disaster readiness
  + Testing – schedule regular game-days. Ensure that game-day scenario is aligned to the real disaster scenarios.
  + Effective Monitoring and Alert
  + Keep Backup always on.
  + Evaluate licensing agreement – ensure that correct licensing option is selected.

# 

# **AWS Security Incident Reporting Guide**

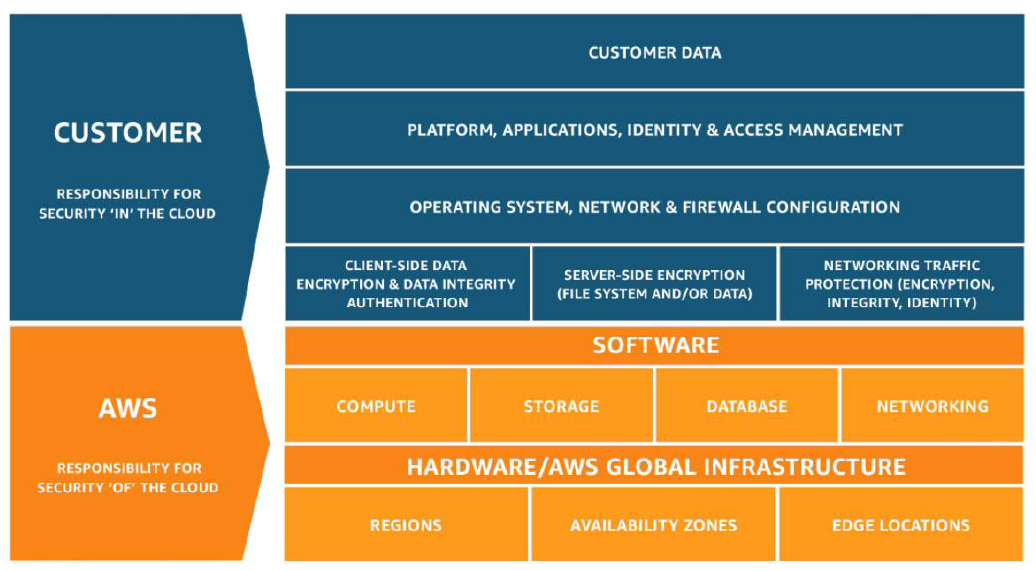
From AWS CAF (Cloud Adaptation Framework) Security prospective it consists of following four components:

* **Directive Control**: Established governance, risk and compliance model within the working environment.
* **Preventive Control**: Protect workload and mitigate threat and vulnerability.
* **Detective Control**: Provide visibility and transparency over the operations.
* **Responsive Control**: Drive remediation of potential deviation from the security baseline.

Foundation of Incident reporting are **EDUCATE**, **PREPARE**, **SIMULATE** and **ITERATE**.

**EDUCATE**

Share responsibility model.



**Design goals for cloud response**

* Establish response Objective
* Response using cloud
* Know what you have and what you need
* Use redeployment mechanism
* Automate where possible
* Chose scalable solution
* Learn and improve your process

**Cloud Security Incident domains**

There are three domains within customer responsibility where security incident can happen – ***service domain, infrastructure domain*** and ***application domain***.

**Indicator of cloud security events**

* Log & Monitoring
* Billing Activity
* Threat Intelligence
* Partner Tool – AWS Partner Network (APN) – Security Partner Solution, Security Solution in AWS Marketplace.
* AWS Outreach – AWS response to Abuse and Compromise sections.
* One-Time contact: there should be a well define / ticketing solution available which employees can use to reach out to the security when they see any abnormality related to security.

**Understanding Cloud Capability:** Understand the various services that AWS offers to detect/response to a security incident.

**Data Privacy:** Even AWS can’t see customer data.

**AWS Response to Abuse and Compromised**: AWS team proactively monitors AWS account for any suspicious and malicious activities, on such event they report and shutdowns un authorized activities running on AWS. Majority of the abuse can be categorised as

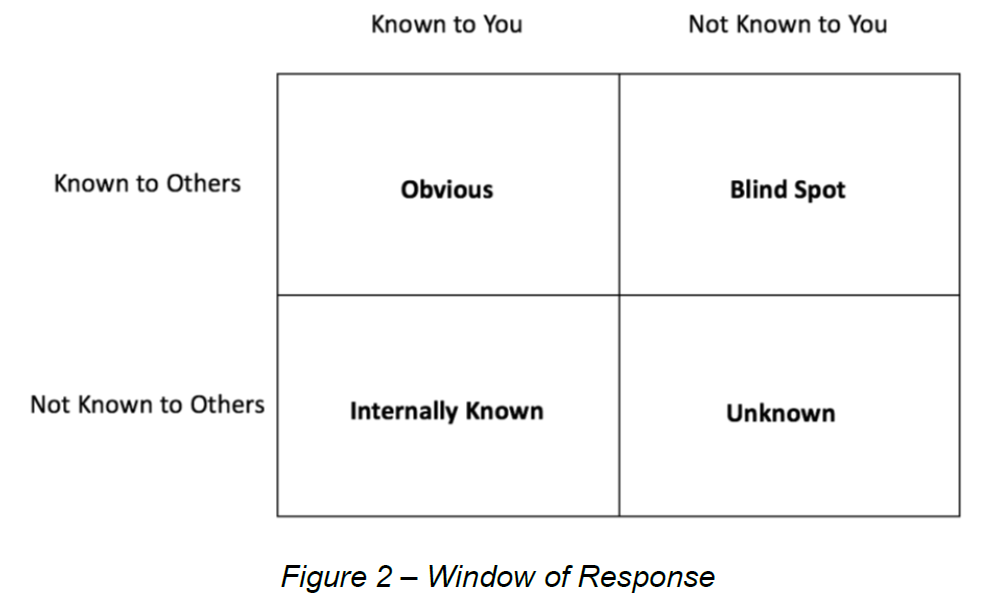
* **Compromised source**: an unpatched EC2 instance which can infect and become a botnet agent.
* **Unintentional Abuse**: Overly aggressively web crawling, this might be categorized as denial-of-services by some websites.
* **Secondary Abuse**: End users storing malicious files in hosted S3 bucket
* **False Complain**: Many a time, internet users may report a normal activity as abusive activities.

**LEARN.**

Automate the process such that, it gives humans has more time to focus and increasing security measure and spend time in corelating events, practicing simulations, device new response procedure, perform research, develop new skills, test and build new tools.

* Define Role and responsibilities
* Provide Trainings
* Define Response Mechanism
* Create a Receptive and Adaptive security culture
* Predicting response – partnered with others /share knowledge

IN 1955, Joseph Luft and Harrington Ingham created the Johari Window (window of response), a simple graph to represent the knowledge of the partner and the internal tribe. Through it was not intend to be use for security risk, the same can be easily extend for security risk.



For **Unknown** quarter, one can follow the below method to reduce the security risk.

**Defence security assertion**: Define security assertion, make it easily searchable. Start from early cloud days, then starting it late.

**Education, communication and research**: Create a cloud security expert in your team or leverage external experts, to scrutinized your environment. Create a feedback look, between the security expert and the engineering team.

**Reduce Attack Surface**: Improve defect to reduce the attack surface for unknown threats.

**Threat Intelligence**: Subscribe to continuous feed of current and relevant security threat, risk and indicators from around the world.

**Alerts**: Generate notification alert for all unusual malicious activities.

**Machine Learning**: Leverage Machine Learning to find complex abnormalities of the organization or specific persona. AWS Macie and AWS Gard Duty can be used for the same. Extend the business-centric data-lakes architecture to create security-centric data lakes, store all kinds of security information into security data lake, and leverage AWS services to derived patter from complex abnormalities.

**PREPARE.**

Prepare access to AWS Account: ensure that security response team have access to the environment where security incident was reported. Place a mechanism in place for the security teams to get access quickly. As its common practice to have multiple AWS account linked to a master account (payee account) it might be required security team to have cross account access - ensure that security team have cross account access. This can be leverage using ***service control policies***.

**In-direct access**: Security team, assist the application team/account owner to apply remediation on an event of incident.

**Direct access:** Application team/account owner deploys IAM roles for the security teams to assume such roles during incident event for applying remediation.

**Alternative Access:** Security incident responder can login into a secluded/secure account to investigate and remediate threat instead of having direct or indirect access to the actual environment.

**Automated Access:** Instead of provisioning access for the incident responder, create role specifically for the automation resources like (EC2 instance and Lambda). When incident occurs automation resources can assume such roles and act on implementing remediation. One can use AWS System Manager Run Command to run administrative tasks remotely & securely on any EC2 instance where AWS System Manager Agent is installed.

**Managed Service Access:** AWS account managed by the trusted third-party partners, to manage/implement/remediate on an event of an incident.

**Prepare Process:** Once the access is provisioned, there should be clear process defined which security incident reporting team can follow in order to investigate and remediate an incident.

**Decision Tree:** Sometimes, different actions need to be implemented based on the incident event.

**Use Alternative Account**: Security incident remediation team, may need to investigate the threat in a separate isolated account. AWS Organization can be used to create a separate forensic environment to analysed the threats. Auto infrastructure automation to create investigating environment mimicking the actual environment in the alternative account.

**View or Copy Data:** Security responders should have view access to the security logs. Appropriate IAM permission should be in place for the responders to copy point-in-time logs into investigation S3 bucket from the production bucket, in order to analyse the incident. Data can be store in S3 storage or can be archive under S3 Glacier for long term retention. One can also protect the data using S3 Glacier Vault Lock, where one can easily apply compliance base rule for long term retention.

**Share EBS Snapshot for incident investigation** – if the snapshot to encrypted make sure cross account access to the CMK (customer managed key) is provided along with the permission for copying the EBS snapshot.

**CloudWatch Logs and VPC flow logs** are store centrally. User Kinesis to process the logs from different AWS account into a single AWS account. While storing the data ensure the storage is immutable to protect the data integrity.

**Launch Resource Near the event:** Incident occurred on premises can also be investigate on the cloud environment, there are better accessibility to service to investigate & response to an incident. It may be beneficial to have a long term separate isolated AWS account for investigation, and for long term storage and legal usages.

**Isolated Resources:** There can be a need to create an isolate the resources (system) to perform forensic investigation. Best practice for launching a forensic investigation instance, Create AMIs and store the AMI or CloudFormation template so that it can be quickly provision when needed, this will also helps in standardising the forensic workstations

1. Chose relevant AMI (windows or LUNIX), for launching forensic investigation workstation.
2. Launch EC2 instance based on the AMI
3. Harden the OS, remove un wanted software packages – configure relevant auditing and Logging mechanism.
4. Install open source / private toolkit software required for investigation
5. Stop EC2 instance, create a AMI from the EC2 instance once its stopped.
6. Weekly/monthly build EC2 instance from the AMI and apply patches to the installed software/platforms.

**Cloud Provide Support**

* **AWS Support** – best practices document, whitepaper, AWS documentations, support forums etc. Chose appropriate support plans.
* **DDoS Support** (Denial of Service OR Distributed Denial of service). AWS provided AWS Shield for DoS (Standard or Advance) support. DoS Standard is free for all customers which includes standards known technique – comprehensive availability protection against well-known infrastructure attacks. User also enrol for advance DDoS protection.

**Simulate**

**Security Incident response simulation** (SIRS) helps in identifying

1. **Validate Readiness**
2. **Develop Confidence – learn from simulation and training staffs.**
3. **Follow Compliance and contractual obligations**
4. **Being agile – incremental improvement with leaser focus.**
5. **Become faster and improving tools**
6. **Refine communications and accelerations**
7. **Developing comfort with rare and unexpected scenarios**

**Simulation Steps:**

**Page#32**

Simulated examples:

**Iterate**

Create a feedback loop, to know what is working AND what is not working. One can create new procedure or update the existing procedure based on the feedback received.

Runbook

Runbook is an organization procedure, which consist of a task or series of tasks which need to be refer when there is an incident occurred. Keep re iterating the tasks to improve the core logic.

Automation

Once the core logic is defined in the runbook, one should look forward towards the automate the task(s). For automation comprehensive AWS APIs can be used

Page#36