EKS | High Availability | DR | AZ

|  |  |
| --- | --- |
| Author | Pradeep/Byron |
| Issue Date | 14th May 2024 |
| Version | 1 |
| Distribution list | Infomagnus-AWS Team |

Contents

[Introduction 2](#_Toc166762131)

[Executive Summary 3](#_Toc166762132)

[Current implementation and Best Practices 3](#_Toc166762133)

[Disaster Recovery 3](#_Toc166762134)

[EKS | High Availability (HA)Design 10](#_Toc166762135)

[EKS | Availability Zones configuration (AZ) 13](#_Toc166762136)

[EKS | Incorporating additional services (RDS, MSK, S3) 13](#_Toc166762137)

[Appendices 14](#_Toc166762138)

# Introduction

This document outlines the design and implementation of a highly available, disaster-resilient Amazon Elastic Kubernetes Service (EKS) cluster using multiple Availability Zones (AZ). It aims to provide insights into the architecture, implementation steps, best practices, and considerations necessary to ensure the reliability and continuity of services for the client PDS (Pacific dental services)

# Executive Summary

Amazon EKS provides a managed Kubernetes service, making it easier to run Kubernetes on AWS without needing to install and operate your own Kubernetes control plane. This document details the high availability, disaster recovery, and multi-AZ configuration to ensure minimal downtime and data loss in case of failures. The proposed architecture will leverage AWS's capabilities to deliver a robust, cost-effective solution that will be leveraged by several of the services managed by PDS.

# Current implementation and Best Practices

## Disaster Recovery

**The current implementation for disaster recovery of an EKS cluster is:**

* No on-cluster cross-region replication.
* Relying on cluster-level backups (Vallero backups).
  + (Question: What are the backup frequencies? Where are they stored and for how long? How often are these backups incremental and how often full?)
* In worst-case scenarios, delete the cluster and restore everything, including specific namespaces and applications.
* Infrastructure is coded in Terraform.
  + The terraform is not automated in a EKS cluster pipeline.
  + Written instructions for creating the EKS cluster.
  + (Question: How does Terraform State manage the ‘drift’ created by deleting the cluster? Does it ignore the drift, or does the recreation of the cluster require starting a new State?)
* There is no explicit testing of Disaster Recovery.
* No documented Recovery Point Objective (RPO) or Recovery Time Objective (RTO) for the cluster.
  + No RTO or RPO defined for resident applications.
* (Question: What alerts the team when a cluster fails? Outside of monitoring, is there a mechanism that sends an alert on cluster failure to a chat channel or actively monitored email box?)

A computer screen shot of a computer network

Description automatically generated

**Best Practices**

**1.Performing Root Cause Analysis (RCA) When a Cluster Fails**

**Why It's Important: Understanding the underlying reasons for cluster failures is crucial for preventing future incidents, improving system reliability, and minimizing downtime. A thorough RCA helps identify the root cause and provides insights for implementing corrective actions.**

**Best Practices:**

* + **Log Aggregation and Monitoring:**
    - **Tool Utilization: Use tools like Amazon CloudWatch, Elasticsearch, Fluentd, and Kibana (EFK stack) to aggregate and analyze logs. This helps in correlating events and identifying the root cause.**
    - **Centralized Logging: Implement centralized logging for all components within your EKS cluster. This makes it easier to search, filter, and analyze logs from a single interface.**
  + **Metrics and Alerts:**
    - **Prometheus and Grafana: Use Prometheus to collect metrics and Grafana to visualize them. Set up alerts for critical metrics to get real-time notifications about potential issues.**
    - **AWS CloudWatch Alarms: Configure CloudWatch Alarms to monitor the health of your EKS cluster and trigger notifications based on predefined thresholds.**
  + **Distributed Tracing:**
    - **AWS X-Ray: Integrate AWS X-Ray to trace requests and understand the performance of your applications. This helps in pinpointing latency issues and errors in microservices architectures.**
  + **Incident Response Plan:**
    - **Runbooks: Develop detailed runbooks that outline steps to be taken during a cluster failure, including how to collect logs, analyze metrics, and identify the root cause.**
    - **Automated Incident Management: Use AWS Systems Manager to automate incident management tasks and streamline the RCA process.**
  + **Post-Mortem Analysis:**
    - **Documentation: Document each incident with a detailed post-mortem report that includes the root cause, impact analysis, and steps taken for resolution.**
    - **Lessons Learned: Conduct regular reviews of incidents and update your processes and runbooks based on lessons learned to prevent future occurrences.**

**2. Implement Cross-Region Replication**

**Why It's Important:** Relying solely on cluster-level backups can lead to significant downtime and potential data loss during catastrophic events. Cross-region replication ensures that critical data and services are available in another region, minimizing downtime and improving disaster recovery capabilities.

**Best Practices:**

* **Data Replication:** Use AWS services like S3 Cross-Region Replication and AWS Backup for cross-region replication of critical data.

**Cloud Agnostic model:**

* 1. Rook (with Ceph): Rook is a cloud-native storage orchestrator for Kubernetes that uses Ceph to provide scalable and resilient storage. Ceph supports cross-region replication through its multi-site features.
  2. **Velero:** provides backup and restore, disaster recovery, and migration capabilities for Kubernetes clusters.
  3. Longhorn: Longhorn is a distributed block storage system for Kubernetes that supports cross-cluster disaster recovery
  4. MinIO:MinIO is a high-performance, Kubernetes-native object storage system. It supports cross-region replication through its multi-site replication feature.
* **Relational Database Service (RDS):**
  + **Multi-AZ Deployments:** RDS supports Multi-AZ (Availability Zone) deployments, which automatically replicate data synchronously to a standby instance in a different AZ within the same region.
  + **Read Replicas:** Create Read Replicas in different regions to replicate data asynchronously. This is useful for offloading read traffic and for disaster recovery.
  + **Aurora Global Database:** For Aurora (a MySQL and PostgreSQL-compatible relational database), use Aurora Global Database to replicate data across multiple regions with low latency. This ensures high availability and supports disaster recovery.
* **Streaming Data:**
  + **MSK (Managed Streaming for Kafka):** Use MSK to replicate Kafka topics across regions using MirrorMaker. This is ideal for streaming data applications that require high availability and disaster recovery.
* Custom Solutions:
  + **ETL Pipelines:** Implement ETL (Extract, Transform, Load) pipelines using services like AWS Glue, Apache NiFi, or Airflow to replicate data between regions. This is useful for more complex data transformation and migration scenarios.
  + **Data Warehousing:** Use Redshift’s cross-region snapshots to replicate and restore data warehouses across regions.

* **Multi-Region Deployment:** Deploy critical components of your EKS cluster across multiple regions to enhance resilience and availability. This approach enhances resilience, availability, and disaster recovery capabilities by ensuring that if one region fails, the services can continue running in another region
* **Benefits:**
  + **Resilience:** By distributing components across regions, you can minimize the impact of regional failures, such as natural disasters or outages, ensuring continuous service availability.
  + **Low Latency:** Deploying services closer to users in different regions can reduce latency and improve user experience by serving requests from the nearest region.
  + **Disaster Recovery:** Multi-region deployment provides a robust disaster recovery strategy, allowing for seamless failover to another region in case of a catastrophic event.
  + **Compliance:** Some industries require data to be stored in specific geographical locations to comply with regulations. Multi-region deployment can help meet these requirements.
* **Best Practices:**
  + **Global Load Balancers:** Use global load balancers like AWS Global Accelerator or Route 53 with latency-based routing to direct traffic to the nearest healthy region.
  + **Data Replication:** Ensure data is replicated across regions using tools like Ceph, Velero, Longhorn, or MinIO for storage, and RDS Read Replicas or Aurora Global Database for relational databases.
  + **Service Discovery:** Implement service discovery mechanisms that can operate across regions, ensuring that services can find each other regardless of their deployment location.
  + **Consistent Configurations:** Maintain consistent configurations and policies across regions to simplify management and reduce the risk of discrepancies.
  + **Automated Failover:** Set up automated failover mechanisms to detect regional failures and route traffic to healthy regions without manual intervention.
  + **Testing and Drills:** Regularly test your multi-region deployment and disaster recovery plans to ensure they work as expected and that your team is prepared for real incidents.
* **Automated Replication:**

1. Consistency: Automated replication processes ensure that data is synchronized across regions in real-time or near-real-time, minimizing discrepancies between regions. This consistency is crucial for applications that require up-to-date information, ensuring all users have access to the latest data regardless of their location.
2. **Reliability:** Automated replication reduces the risk of human error by eliminating the need for manual intervention in the replication process. This ensures that replication tasks are executed consistently and reliably, enhancing overall system reliability and reducing the likelihood of data inconsistencies or loss.
3. **Scalability:** Automated replication can easily scale with your data growth. As your application generates more data, the automated processes ensure that new data is continuously replicated across regions without additional manual effort. This scalability is essential for handling high data throughput and maintaining performance.
4. **Disaster Recovery:** In the event of a disaster, automated replication ensures that a recent and consistent copy of your data is available in a secondary region. This enables quick recovery and minimizes downtime, which is critical for maintaining service availability and meeting recovery time objectives (RTO).
5. **Operational Efficiency:** Automating the replication process reduces operational overhead by minimizing the need for manual management and intervention. This allows your operations team to focus on other critical tasks and improves overall efficiency.
6. **Error Handling and Monitoring:** Automated replication solutions often include built-in error handling and monitoring capabilities. These features ensure that any issues in the replication process are detected and resolved quickly, maintaining data consistency and reliability across regions

**3. Enhance Backup and Restore Procedures**

**Why It's Important:** Regular and automated backups reduce the risk of data loss and provide a reliable method for restoring data and services.

**Best Practices:**

* **Frequent Backups:** Increase the frequency of backups from weekly to daily or more frequently for critical data.
* **Automated Backups:** Use AWS Backup to automate the backup process for EBS volumes, EFS filesystems, and other critical data.
* **Backup Monitoring:** Implement monitoring and alerting for backup processes to ensure backups are completed successfully.
* **Store Database Backups in Backup Region**: In the event of primary region failure, it is imperative to have access to database backup files available outside of the failed region.
* **Test RPO and RTO targets**: To ensure that recovery is within objectives, test the database can be restored within time objectives and the any data loss is limited to recovery point objective.

**Automated Failover to Backup Region:** Automated failover significantly reduces downtime by quickly switching operations to a backup region when a primary region experiences an outage. This ensures that services remain available with minimal interruption, maintaining business continuity.Automated failover enhances the overall resilience of your infrastructure. By having a standby region ready to take over, you mitigate the risk of single points of failure and ensure that your system can withstand regional outages or disasters.With automated failover, users experience minimal disruption. The failover process is designed to be seamless, so users may not even notice that a failover has occurred. This maintains a consistent and reliable user experience

**4. Infrastructure is coded in Terraform**

**Steps to achieve best practices:**

* **Use Pipelines to automate Terraform:** Automating Terraform with pipelines ensures that infrastructure is provisioned in a consistent and repeatable manner. This reduces the risk of human error and ensures that the same configuration is applied every time, leading to more reliable and predictable infrastructure deployments.
* Automating the deployment process saves time and effort. Instead of manually running Terraform commands, pipelines can automatically execute them, freeing up resources to focus on other critical tasks
* Using pipelines integrates Terraform workflows with version control systems like Git. This allows for tracking changes to the infrastructure code, maintaining a history of modifications, and enabling rollbacks if necessary. It also ensures that infrastructure changes are peer-reviewed and tested before deployment.
* Pipelines make it easier to scale the infrastructure management process. As the infrastructure grows, pipelines can handle the increased complexity and volume of deployments, ensuring that all changes are managed efficiently
* Pipelines enable CI/CD practices for infrastructure as code (IaC). This allows for automated testing, validation, and deployment of infrastructure changes, leading to faster and more reliable delivery of updates and new features.

**Requirements for Using Pipelines to Automate Terraform**

* **Set Up Version Control:**
  + **Store your Terraform code in a version control system (VCS) like GitHub, GitLab, or Bitbucket. This allows for collaborative development, version tracking, and integration with CI/CD pipelines.**
* **Choose a CI/CD Tool:**
  + **Select a CI/CD tool that supports Terraform automation. Popular options include Jenkins, GitLab CI/CD, GitHub Actions, CircleCI, and AWS CodePipeline.**
* **Create Pipeline Configuration:**
  + **Define a pipeline configuration file (e.g., .gitlab-ci.yml for GitLab, .github/workflows for GitHub Actions) that specifies the steps for running Terraform commands. The pipeline should include stages for:**
* **Planning: Run terraform plan to generate an execution plan and identify changes.**
* **Validation: Validate the Terraform configuration with terraform validate and format the code using terraform fmt.**
* **Applying: Apply the changes with terraform apply to update the infrastructure.**
* **Set Up Secrets and Credentials:**
  + **Store sensitive information like AWS credentials, Terraform Cloud API tokens, or other necessary secrets in the CI/CD tool’s secret management system. Ensure that these secrets are securely managed and accessible by the pipeline**

1. **How do you manage DNS and traffic for high availability and failover across regions? Are there specific tools or AWS services you rely on for this?**

**Client Response**

**PDS Answer:**

* Distributing nodes across different zones.
* Distributing pods across different zones for high availability.
* Using CoreDNS with multiple replicas and monitoring.
* Pointing CoreDNS at two different domain controllers for internal DNS.

Best Practices

**1. Enhance DNS Management for High Availability**

**Why It's Important:** Ensuring high availability and reliability of DNS is critical for maintaining application uptime and seamless failover capabilities.

Multi-Region DNS: Use Amazon Route 53 for DNS management, which supports health checks and failover routing to direct traffic across multiple regions.

Auto Scaling: Ensure Core DNS replicas are automatically scaled based on the load to handle increased traffic.

**2. Implement Cross-Region Traffic Management**

**Why It's Important:** Managing traffic across multiple regions ensures high availability and optimal performance for users by routing traffic to the healthiest and closest region.

**Best Practices:**

* **Global Accelerator:** Use AWS Global Accelerator to improve the availability and performance of your applications with automatic health checks and routing.
* **Route 53 Health Checks:** Set up health checks and failover routing policies in Amazon Route 53 to ensure traffic is directed to healthy endpoints.

1. **What are the RTO and RPO for the organization? SLAs and SLOs?**

**Client Response**

**PDS Answer:**

* The organization needs to define RTO and RPO targets.
* There is a high bar they are aiming for, but specifics need to be discussed and defined through the Standards Committee for Kubernetes.

**Best Practices**

**1. Understanding RTO and RPO**

**Recovery Time Objective (RTO):** The maximum acceptable amount of time it should take to restore a service after a failure or disaster.

**Recovery Point Objective (RPO):** The maximum acceptable amount of data loss measured in time. It defines how much data the organization can afford to lose during a disaster.

**Best Practices:**

* **Assess Critical Systems:** Identify and assess critical systems and applications to determine appropriate RTO and RPO targets.
* **Business Impact Analysis:** Conduct a business impact analysis (BIA) to understand the potential impact of downtime and data loss on business operations.
* **Regular Review:** Regularly review and update RTO and RPO targets based on changes in business requirements and technological advancements.

1. **Has DR plans been tested in a live environment?**

**Client Response**

**PDS Answer:**

* DR plans have been tested by creating separate clusters and restoring the entire enterprise applications cluster onto these separate clusters.
* Some reconfiguration was necessary, but the data and functionality were successfully restored.

**Best Practices**

**1. Regularly Test DR Plans in Production-Like Environments**

**Why It's Important:** Testing DR plans in environments that closely mimic production ensures that the plans will work effectively during an actual disaster. This helps identify potential issues and allows for fine-tuning of the DR processes.

**Best Practices:**

* **Production-Like Testing:** Conduct DR tests in non-production environments that closely mimic the production setup to ensure accuracy.
* **Scheduled Tests:** Schedule regular DR tests (e.g., quarterly) to ensure that the DR plan remains effective and up-to-date.
* **Comprehensive Testing:** Include all critical components in the tests, such as data restoration, application functionality, and network configurations.

1. **What other thoughts about system failure that are not covered by DR? Less than a full system failure—graceful fail.**

**Client Response**

**PDS Answer:**

* It depends on what fails and who supports what during a failure.
* Consideration of how certain component failures impact the system.
* For user experience, fallback mechanisms to legacy systems can provide partial functionality.
* Issues with critical components like load balancers or Istio could heavily impact user experience.

**Best Practices**

**1. Implement Graceful Degradation**

**Why It's Important:** Graceful degradation ensures that when certain components fail, the system continues to function at a reduced capacity rather than completely failing. This improves user experience and maintains partial service availability.

**Best Practices:**

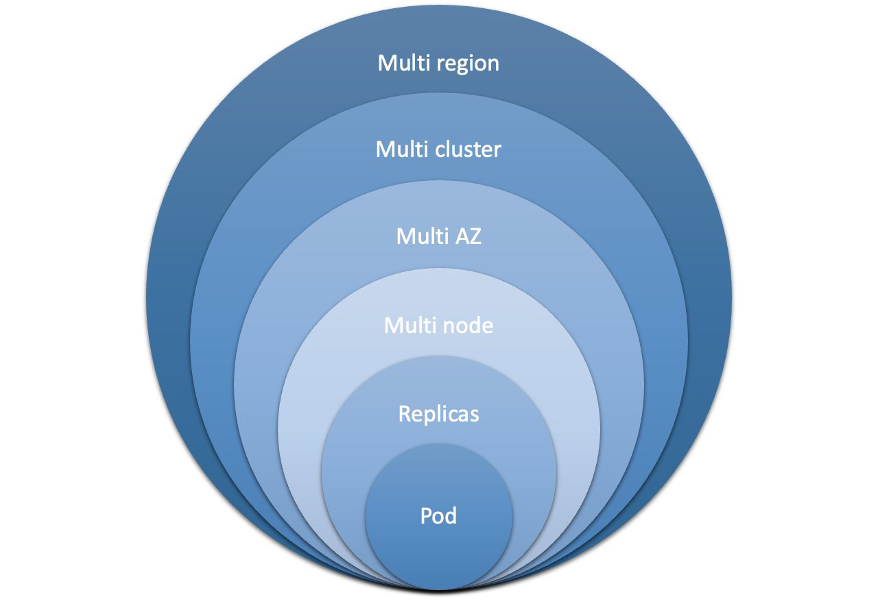
* **Feature Toggles:** Use feature toggles to enable or disable specific features dynamically based on system health.
* **Fallback Mechanisms:** Implement fallback mechanisms to provide alternative functionality or direct users to legacy systems.
* **User Communication:** Inform users about degraded functionality and expected recovery time to manage their expectations.

# EKS | High Availability (HA)Design

The primary goal of HA design is to ensure that the EKS cluster can withstand failures and continue to operate without significant downtime, meeting the desired uptime and service level agreements (SLAs).

**Architecture**

* **Multi-AZ Deployment**: Distributing resources across multiple AZs to avoid single points of failure.
* **Redundancy**: Ensuring that there are redundant instances of critical components.
* **Load Balancing**: Using Elastic Load Balancers (ELB) to distribute traffic evenly across nodes.



Kubernetes brings powerful orchestration capabilities to enhance workload resilience:

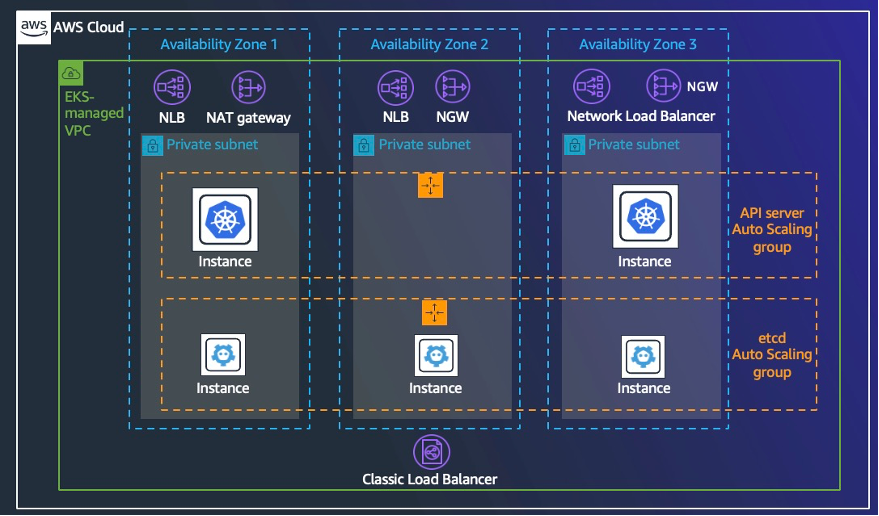
1. Replication and scaling: Kubernetes allows you to define the desired number of replicas for your workloads, ensuring that multiple instances of the application are running concurrently. If a replica fails or becomes unresponsive, Kubernetes automatically replaces it with a healthy one.
2. Health checks and self-healing: Kubernetes continuously monitors the health of individual replicas by performing health checks. If a replica fails these checks, then Kubernetes automatically terminates it and starts a new one in its place. This self-healing feature ensures that the workload remains available and resilient even in the presence of failures.
3. Fault isolation: Kubernetes allows you to define and enforce resource limits and constraints for workloads. By setting resource quotas and limits, you can prevent a single workload from consuming excessive resources, which reduces the risk of resource exhaustion and isolating failures to specific workloads rather than affecting the entire cluster.
4. Rolling updates and rollbacks: Kubernetes facilitates seamless rolling updates, which allows you to update your application without incurring downtime. By rolling out updates in a controlled manner, Kubernetes ensures that a minimum number of replicas are available and operational at all times. Additionally, Kubernetes supports rollbacks that revert to a previous version of an application if issues arise during an update.
5. Multi-domain deployments: Kubernetes supports deploying workloads across multiple failure domains. By deploying replicas across multiple nodes, AZs, and clusters, you increase the resilience of your workload against infrastructure failure at various levels.

**Generic Implementation Steps**

1. **Create a VPC with Multiple Subnets**: Ensure subnets are distributed across at least three different AZs.
2. **Configure Security Groups**: Define security groups to control traffic between nodes and external services.
3. **Deploy EKS Cluster**: Use AWS Management Console, CLI, or eksctl to create an EKS cluster across multiple AZs.

**eksctl create cluster --name my-cluster --region us-west-2 --zones us-west-2a,us-west-2b,us-west-2c**

1. **Set Up Worker Nodes**: Create Auto Scaling Groups (ASGs) for worker nodes across multiple AZs.
2. **Configure Load Balancing**: Deploy an ELB/NLB to distribute traffic to worker nodes.
3. **Implement Health Checks**: Regularly check the health of the control plane and worker nodes.



# EKS | Availability Zones configuration (AZ)

**Objectives**

Leverage multiple AZs to enhance fault tolerance and resilience of the EKS cluster.

**AZ Design Considerations**

* **Resource Distribution**: Evenly distribute resources across AZs to prevent overloading a single AZ.
* **Network Latency**: Consider latency and throughput between AZs when designing the network.

**Network Configuration**

* **VPC and Subnets**: Design a VPC with subnets in multiple AZs.
* **Security Groups and NACLs**: Define security groups and network ACLs to control traffic within the VPC.

**Implementation Steps**

1. **VPC Creation**: Create a VPC with subnets in at least three AZs.
2. **EKS Cluster Setup**: Deploy the EKS cluster across these AZs.
3. **Worker Node Configuration**: Configure worker nodes in ASGs spanning multiple AZs.
4. **Network Configuration**: Set up appropriate security groups, NACLs, and routing tables.

# EKS | Incorporating additional services (RDS, MSK, S3)

**Amazon RDS for MSSQL**

**Benefits:**

* **Multi-AZ Deployment: Automatic failover to a standby instance in another AZ in case of an outage.**
* **Automated Backups: Daily backups and point-in-time recovery.**
* **High Availability and Durability: Built-in replication and automated failover.**

**Integration Steps:**

1. **Create RDS Instance: Set up an RDS instance with Multi-AZ configuration.**
2. **Configure Security: Set up security groups and IAM roles.**
3. **Database Connections: Configure applications to connect to the RDS instance.**
4. **Automate Backups and Maintenance: Enable automated backups and maintenance tasks.**

**The following are some limitations when working with Multi-AZ deployments on RDS for SQL Server DB instances:**

**1.Cross-Region Multi-AZ isn't supported.**

**2.Stopping an RDS for SQL Server DB instance in a multi-AZ deployment isn't supported.**

**3.You can't configure the secondary DB instance to accept database read activity.**

**4.Multi-AZ with Always on Availability Groups (AGs) supports in-memory optimization.**

**5.Multi-AZ with Always On Availability Groups (AGs) doesn't support Kerberos authentication for the availability group listener. This is because the listener has no Service Principal Name (SPN).**

**6.You can't rename a database on a SQL Server DB instance that is in a SQL Server Multi-AZ deployment. If you need to rename a database on such an instance, first turn off Multi-AZ for the DB instance, then rename the database. Finally, turn Multi-AZ back on for the DB instance.**

**7.You can only restore Multi-AZ DB instances that are backed up using the full recovery model.**

**Multi-AZ deployments have a limit of 10,000 SQL Server Agent jobs.**

**If you need a higher limit, request an increase by contacting AWS Support.**

**Managed Streaming for Kafka (MSK)**

**Benefits:**

* **Scalable Messaging Platform**: Facilitates real-time data streaming and analytics.
* **High Availability**: MSK automatically replicates data across multiple AZs.

**Integration Steps:**

1. **MSK Cluster Creation**: Create an MSK cluster across multiple AZs.
2. **Configure Producers and Consumers**: Set up applications to produce and consume messages from Kafka topics.
3. **Monitoring and Maintenance**: Use AWS tools to monitor Kafka performance and health.

**Amazon S3**

**Benefits:**

* **Durable Storage**: Provides scalable storage for backups, logs, and static assets.
* **Cross-Region Replication**: Ensures data durability and availability by replicating data across regions.

**Integration Steps:**

1. **Bucket Creation**: Create S3 buckets for different use cases (e.g., backups, logs).
2. **Configure Permissions**: Set up appropriate IAM policies and bucket policies.
3. **Enable Cross-Region Replication**: Configure replication rules for critical data.

# Appendices

The proposed HA, DR, and AZ configuration for EKS clusters ensures high availability, resilience, and minimal downtime. By following the best practices and implementation steps outlined in this document, clients can achieve a robust and cost-effective Kubernetes environment on AWS.

**Glossary**

* **EKS**: Amazon Elastic Kubernetes Service
* **HA**: High Availability
* **DR**: Disaster Recovery
* **AZ**: Availability Zone
* **VPC**: Virtual Private Cloud
* **ASG**: Auto Scaling Group
* **RBAC**: Role-Based Access Control
* **IAM**: Identity and Access Management
* **KMS**: Key Management Service
* **ELB**: Elastic Load Balancer
* **NLB**: Network Load Balancer

**References**

* [Amazon EKS Documentation](https://docs.aws.amazon.com/eks/)
* [AWS Well-Architected Framework](https://aws.amazon.com/architecture/well-architected/)
* <https://docs.aws.amazon.com/eks/latest/userguide/disaster-recovery-resiliency.html>
* [Operating resilient workloads on Amazon EKS | Containers](https://aws.amazon.com/blogs/containers/operating-resilient-workloads-on-amazon-eks/)

**Acronyms**

* **EBS**: Elastic Block Store
* **EFS**: Elastic File System
* **S3**: Simple Storage Service
* **TLS**: Transport Layer Security
* **API**: Application Programming Interface