EKS | High Availability | DR | AZ

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

Contents

[Introduction 3](#_Toc166669659)

[Executive Summary 3](#_Toc166669660)

[Current implementation and Best Practices 3](#_Toc166669661)

[Current state architecture (PDS) 6](#_Toc166669662)

[EKS Cluster Architecture Overview 13](#_Toc166669663)

[Recommendations at PDS by Infomagnus 15](#_Toc166669664)

[Recommendations at PDS by AWS 17](#_Toc166669665)

[EKS | High Availability (HA)Design 17](#_Toc166669666)

[Implementation recommendations steps using Terraform scripts at PDS 19](#_Toc166669667)

[EKS | Disaster Recovery Strategy (DR) 20](#_Toc166669668)

[EKS | Availability Zones configuration (AZ) 23](#_Toc166669669)

[EKS | Security considerations 25](#_Toc166669670)

[EKS | Incorporating additional services (RDS, MSK, S3) 27](#_Toc166669671)

[EKS | Monitoring and Logging 28](#_Toc166669672)

[EKS | Cost considerations 29](#_Toc166669673)

[EKS | Benefits of HA, DR and AZ 29](#_Toc166669674)

[EKS | Benefits for Dental services client 30](#_Toc166669675)

[EKS | Conclusion 31](#_Toc166669676)

[Appendices 31](#_Toc166669677)

# 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?)

**Put in a basic EKS cluster that represents the PDS implementation. –More basic than the example in the diagram below.**

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 CoreDNS 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.

# Current state architecture (PDS)

The EKS Network Architecture document was provided by the customer, describes a Kubernetes-based infrastructure on AWS.

A computer screen shot of a computer network

Description automatically generated

Current Network Flow at PDS

1. App Authentication:

The process starts with the ROC application authenticating via Okta, which retrieves a JWT token from an Okta application. The ROC

application then makes an API call to a service within the EKS cluster

2. Ingress ALB

The API call is directed to the ALB ingress resource on the EKS cluster, which gets routed based on the host header. The ALB uses the

wildcard certificate from AWS Certificate Manager for SSL termination.

3. Istio Gateway

The Istio Gateway receives traffic from ALB and re-encrypts it using certificate from TLS secret.

The Istio Gateway validates the JWT token as part of the authentication policy using the Okta authorization server public key and issuer

URL.

4. Virtual Service

The traffic is routed to the appropriate virtual service within the cluster, which is determined by the path and/or host specified in the API

call, which matches with the rules defined in the Istio configuration (App Virtual Service 1 or App Virtual Service 2).

5. Service Handling

Once the traffic reaches the appropriate service (App Service 1 or App Service 2), it is handled by a Kubernetes pod.

1 "enableNetworkPolicy": "true"

2 "env": {

3 "ENABLE\_PREFIX\_DELEGATION": "true"

4 "AWS\_VPC\_K8S\_CNI\_CUSTOM\_NETWORK\_CFG": "true"

5 "ENI\_CONFIG\_LABEL\_DEF":"topology.kubernetes.io/zone"

6 }

**Current network flow best practices at PDS**

**1. App Authentication**

**Best Practices:**

* **Secure Authentication:** Use OAuth 2.0 and OpenID Connect (OIDC) for secure authentication and authorization.
* **Token Management:** Implement proper token storage and management. Use short-lived tokens and refresh tokens to enhance security.
* **Validate Tokens:** Ensure JWT tokens are validated in the backend service to confirm their authenticity.

**Example Implementation:**

// Sample Node.js code for authenticating with Okta and retrieving JWT

const OktaAuth = require('@okta/okta-auth-js');

const oktaAuth = new OktaAuth({

issuer: 'https://{yourOktaDomain}/oauth2/default',

clientId: '{yourClientId}',

});

async function authenticate(username, password) {

const transaction = await oktaAuth.signInWithCredentials({

username,

password,

});

if (transaction.status === 'SUCCESS') {

return transaction.sessionToken;

} else {

throw new Error('Authentication failed');

}

}

// Use the sessionToken to get the JWT

async function getToken(sessionToken) {

const tokenResponse = await oktaAuth.token.getWithoutPrompt({

sessionToken,

responseType: 'id\_token',

scopes: ['openid', 'profile', 'email'],

});

return tokenResponse.tokens.idToken;

}

**2. Ingress ALB**

**Best Practices:**

* **SSL Termination:** Use SSL termination at the ALB with AWS Certificate Manager (ACM) to manage SSL/TLS certificates.
* **Routing Rules:** Define clear and specific routing rules based on host headers and paths.
* **Security Groups:** Attach security groups to the ALB to control inbound and outbound traffic.

**Example Implementation:**

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: example-ingress

annotations:

alb.ingress.kubernetes.io/scheme: internet-facing

alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}]'

alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:region:account-id:certificate/certificate-id

spec:

rules:

- host: example.com

http:

paths:

- path: /api/\*

backend:

serviceName: my-service

servicePort: 80

**3. Istio Gateway**

**Best Practices:**

* **Re-Encryption:** Use Istio to re-encrypt traffic from the ALB for end-to-end encryption.
* **JWT Validation:** Configure Istio to validate JWT tokens using the Okta authorization server public key and issuer URL.

**Example Implementation:**

apiVersion: networking.istio.io/v1alpha3

kind: Gateway

metadata:

name: my-gateway

spec:

selector:

istio: ingressgateway

servers:

- port:

number: 443

name: https

protocol: HTTPS

tls:

mode: SIMPLE

credentialName: my-tls-secret

hosts:

- example.com

---

apiVersion: security.istio.io/v1beta1

kind: RequestAuthentication

metadata:

name: jwt-authentication

spec:

selector:

matchLabels:

app: my-app

jwtRules:

- issuer: "https://{yourOktaDomain}/oauth2/default"

jwksUri: "https://{yourOktaDomain}/oauth2/default/v1/keys"

**4. Virtual Service**

**Best Practices:**

* **Routing Rules:** Define detailed routing rules in the Virtual Service to direct traffic to the appropriate backend services based on the path and host headers.
* **Retries and Timeouts:** Configure retries and timeouts to handle transient failures.

**Example Implementation:**

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: my-virtual-service

spec:

hosts:

- example.com

gateways:

- my-gateway

http:

- match:

- uri:

prefix: /api/service1

route:

- destination:

host: service1

port:

number: 80

- match:

- uri:

prefix: /api/service2

route:

- destination:

host: service2

port:

number: 80

**5. Service Handling**

**Best Practices:**

* **Pod Scaling:** Use Kubernetes Horizontal Pod Autoscaler (HPA) to automatically scale pods based on demand.
* **Resource Requests and Limits:** Define resource requests and limits for CPU and memory to ensure optimal performance and resource usage.

**Example Implementation:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: my-app-image

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

---

apiVersion: autoscaling/v1

kind: HorizontalPodAutoscaler

metadata:

name: my-app-hpa

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: my-app

minReplicas: 1

maxReplicas: 10

targetCPUUtilizationPercentage: 50

**Network Policy Configuration**

**Best Practices:**

* **Network Policies:** Use Kubernetes Network Policies to control traffic between pods and ensure only authorized communication is allowed.

**Example Implementation:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: allow-app-traffic

spec:

podSelector:

matchLabels:

app: my-app

policyTypes:

- Ingress

- Egress

ingress:

- from:

- podSelector:

matchLabels:

app: my-app

egress:

- to:

- podSelector:

matchLabels:

app: my-app

**Environment Variables Configuration**

**Best Practices:**

* **Environment Variables:** Use environment variables to configure the Kubernetes CNI plugin and enable specific features like prefix delegation and custom network configurations.

**Example Implementation:**

apiVersion: v1

kind: ConfigMap

metadata:

name: cni-config

data:

ENABLE\_PREFIX\_DELEGATION: "true"

AWS\_VPC\_K8S\_CNI\_CUSTOM\_NETWORK\_CFG: "true"

ENI\_CONFIG\_LABEL\_DEF: "topology.kubernetes.io/zone"

By following these best practices and examples, you can ensure a robust, secure, and scalable network flow for your EKS cluster with high availability and disaster recovery capabilities

# EKS Cluster Architecture Overview

Amazon EKS is a fully managed service that simplifies running Kubernetes on AWS. It handles the Kubernetes control plane, including the API server and the etcd database, allowing users to focus on deploying and managing applications.

Key Components

EKS Control Plane: Managed by AWS, including the API server, etcd, and other control plane components.

Worker Nodes: EC2 instances running Kubernetes pods, managed by the user.

Networking: VPC, subnets, security groups, and network policies.

Storage: Persistent storage using EBS, EFS, and S3.

**Current Key components at PDS**

A computer screen shot of a computer network

Description automatically generated

* 1. EKS Cluster: The central component is the Amazon EKS cluster, which manages Kubernetes worker nodes and pods.
  2. Node Groups: The architecture utilizes Ubuntu AMI-based node groups, which are sets of EC2 instances that run the application containers.
  3. VPC and Subnets: The EKS cluster is deployed within an AWS VPC (Virtual Private Cloud) with a primary CIDR of 10.0.0.0/22 and a secondary subnet of 100.64.0.0/16. Availability Zones are specified as us-west-2a, 2b, and 2c.
  4. Load Balancers and Routing: An Ingress Application Load Balancer (ALB) is used with host-based routing and SSL termination. There's an Istio Gateway and SSL re-encryption for secure communication.
  5. AWS Certificate Manager: Used for managing SSL certificates (\*.pdsconnect.com).
  6. Storage and Secrets Management: Amazon Elastic Block Store (EBS) is used for persistent storage, with a defined EBS StorageClass. AWS Secrets Manager and Secrets CSI Driver are used for managing sensitive data.
  7. Networking and Security Policies: Istio is used for service mesh capabilities, along with Network Policies for controlling traffic and IstioSecurityPolicies for additional security layers.
  8. CI/CD and Development Workflow: Bitbucket is used for source code management, with a pipeline for continuous.

integration and deployment. Developers push changes to Bitbucket, triggering automated deployments.

* 1. Monitoring and Logging: The architecture includes Prometheus, Loki, and Grafana for monitoring, along with S3 buckets for tracing and logging.
  2. Additional Components: ArgoCD for continuous delivery, CoreDNS for DNS management, and Karpenter for provisioning nodes dynamically.

# Recommendations at PDS by Infomagnus

Here are the best practices and recommendations for each component of your Amazon EKS setup at PDS client:

**1. EKS Cluster**

**Recommendation:** Implement EKS Control Plane logging and auditing.

**Example:** Enable control plane logging to track actions made on your cluster for security and troubleshooting purposes:

aws eks update-cluster-config --region us-west-2 --name YourClusterName --logging '{"clusterLogging":[{"types":["api","audit"],"enabled":true}]}'

**2. Node Groups**

**Recommendation:** Implement multi-AZ deployments and Autoscaling Groups.

**Example:** Create node groups in multiple Availability Zones to ensure high availability and fault tolerance. Utilize Auto Scaling to manage the scale based on demand:

aws eks create-nodegroup --cluster-name YourClusterName --nodegroup-name multi-az-group --subnets subnet-1234abc subnet-5678def subnet-9012ghi --scaling-config minSize=1,maxSize=5,desiredSize=3

**3. VPC and Subnets**

**Recommendation:** Use VPC flow logs for monitoring and anomaly detection in network traffic. **Example:** Enable VPC flow logs to capture information about the IP traffic going to and from network interfaces in your VPC:

aws ec2 create-flow-logs --resource-type VPC --resource-id vpc-abc123 --traffic-type ALL --log-destination arn:aws:logs:us-west-2:123456789012:log-group:MyFlowLogs

**4. Load Balancers and Routing**

**Recommendation:** Enhance monitoring and alerting for your ALB performance and health.

**Example:** Set up CloudWatch alarms for your ALB to monitor HTTP 5XX error rates and alert when thresholds are exceeded, helping to identify issues early:

aws cloudwatch put-metric-alarm --alarm-name "High-HTTP-5XX-Errors" --metric-name HTTPCode\_ELB\_5XX\_Count --namespace AWS/ApplicationELB --statistic Sum --period 300 --threshold 10 --comparison-operator GreaterThanOrEqualToThreshold --evaluation-periods 2 --alarm-actions arn:aws:sns:us-west-2:123456789012:my-sns-topic

**5. AWS Certificate Manager**

**Recommendation:** Automate the renewal and deployment of SSL certificates.

**Example:** Integrate AWS Lambda with ACM to automate the checking and updating of SSL certificates, thereby minimizing manual intervention and reducing the risk of expired certificates affecting the availability.

**6. Storage and Secrets Management**

**Recommendation:** Implement automated backups for EBS and rotation policies for Secrets. **Example:** Use AWS Backup to automate the backup of EBS volumes and implement a lifecycle policy in Secrets Manager to rotate secrets automatically:

aws backup create-backup-plan --backup-plan file://backup-plan.json

aws secretsmanager rotate-secret --secret-id MyDatabaseSecret

**7. Networking and Security Policies**

**Recommendation:** Regularly update and review Istio and Network Policies to adapt to changing security needs.

**Example:** Review and apply updates to Istio configurations and Kubernetes Network Policies periodically to ensure they meet the latest security practices and organizational policies.

**8. CI/CD and Development Workflow**

**Recommendation:** Implement automated security scans and compliance checks in your CI/CD pipeline.

**Example:** Integrate tools like SonarQube for code quality and security scans into your Bitbucket pipelines to ensure code quality and security before deployment:

pipelines:

default:

- step:

name: Code Quality and Security Scan

script:

- sonar-scanner

**9. Monitoring and Logging**

**Recommendation:** Utilize centralized logging and implement anomaly detection.

**Example:** Aggregate logs into a central S3 bucket and use Amazon Athena for querying large log datasets. Set up anomaly detection in Grafana to monitor unexpected behavior:

SELECT \* FROM s3\_logs WHERE type = 'error'

**10. Additional Components**

**Recommendation:** Regularly review and optimize your Kubernetes resource allocations.

**Example:** Use Karpenter for more efficient resource management and cost optimization by dynamically adjusting node groups based on the cluster's actual needs.

These tailored recommendations and examples should guide you in enhancing the robustness, efficiency, and security of your Kubernetes architecture at PDS client.

# Recommendations at PDS by AWS

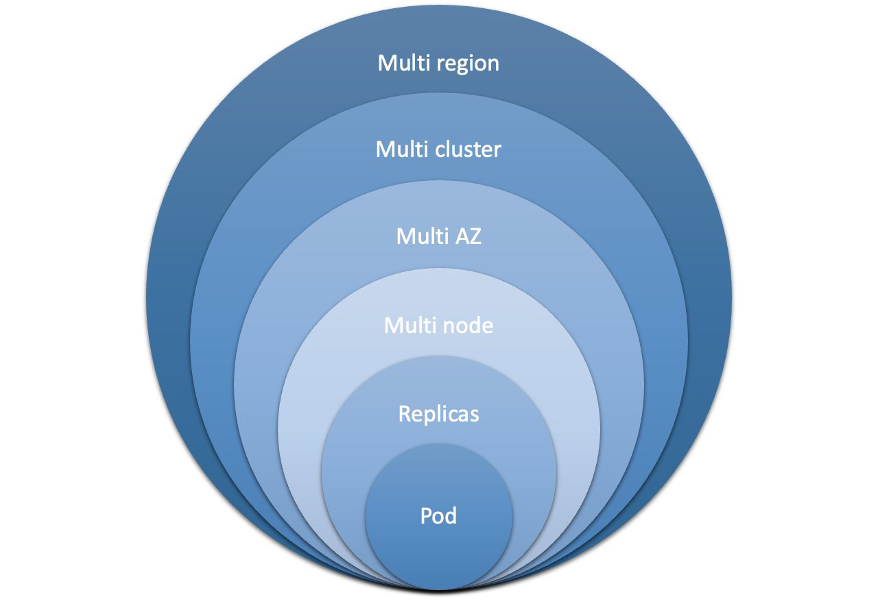
1. Auto-Scaling: Ensure that Karpenter or the Horizontal Pod Autoscaler is properly configured for dynamic scaling based on load, which can optimize costs and performance.
2. Monitoring and Alerting: Strengthen monitoring and alerting mechanisms. Make sure that all critical metrics and logs are being monitored and set up alerts for any anomalies.
3. Disaster Recovery: Implement a robust disaster recovery plan. This could include multi-region deployment, regular backups, and a clear rollback strategy.
4. Security Enhancements: Regularly audit security policies and IAM roles. Implement stricter network policies if necessary and ensure that all traffic is encrypted, both at rest and in transit.
5. Cost Optimization: Regularly review and optimize costs. This includes right-sizing resources, using spot instances where appropriate, and turning off unused resources.
6. Performance Tuning: Continuously monitor the performance and tune the configurations of the EKS cluster, databases, and other services for optimal performance.
7. CI/CD Improvements: Streamline and automate the CI/CD pipeline further to reduce deployment times and improve developer productivity.
8. Documentation and Training: Maintain comprehensive documentation of the architecture and ensure that the team is well-trained on Kubernetes and AWS best practices.
9. Use of Managed Services: Where possible, use AWS managed services to reduce the overhead of maintenance and management. *AWS managed prometheus and Grafana*
10. Regular Audits: Conduct regular audits of the infrastructure for compliance, performance, and security

# 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:

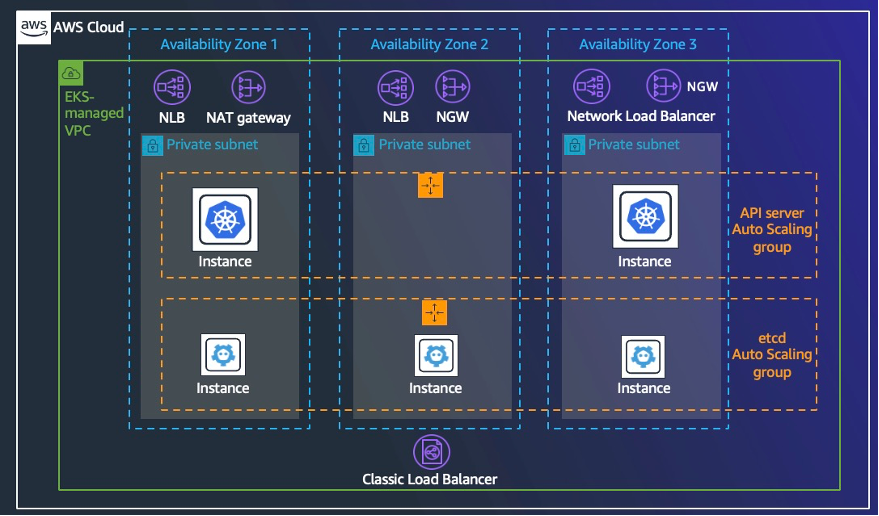
1. Replication and scaling: Kubernetes allows you to define the desired number of replicas for your workloads, ensuring that multiple instances of the application are running concurrently. If a replica fails or becomes unresponsive, Kubernetes automatically replaces it with a healthy one.
2. Health checks 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.



# Implementation recommendations steps using Terraform scripts at PDS

1. Create VPC using Terraform.
2. VPC with /22 CIDR block and 3 private subnets (/24)
3. Transit gateway attachment.
4. Private route table
5. Create EKS cluster using tf-eks module.

**Best Practices at PDS**

* **Managed Node Groups** in Amazon EKS simplify the lifecycle management of nodes (instances) in a Kubernetes cluster. These node groups automate tasks such as patching, scaling, and updating nodes.When a new Kubernetes version is available, you can easily update your node group to use the newer AMI with the latest patches. This ensures your cluster is secure without manually upgrading each node

Example: Deployment: You can set up a managed node group through the AWS Management Console,CLI, or Terraform.

**Health Checks:** Implement regular health checks and failover testing. Implementing comprehensive health checks ensures that the system automatically handles failures by replacing instances that fail the configured checks, thereby increasing reliability

Application-level Checks: These checks involve more than just checking for an HTTP 200 response. They ensure that the application is returning the correct content and connecting to backend databases or services successfully.

System-level Checks: These might include checks for memory usage, CPU spikes, or disk I/O, which can be configured using CloudWatch alarms to trigger actions in response to specific thresholds being met.

**Examples of Health Checks:**

**Kubernetes Probes:** Utilize Liveness, Readiness, and Startup probes in your deployments:

* + **Liveness Probe:** Checks if the application is running. If the check fails, Kubernetes restarts the container.
  + **Readiness Probe:** Determines if the application is ready to serve traffic. Kubernetes does not send traffic to a pod until the readiness probe succeeds.
  + **Startup Probe:** Ensures that the application starts up without issues before the Liveness probe takes over.
* **Auto Scaling**: Use cluster autoscaler to adjust the number of nodes based on demand and The cluster autoscaler automatically adjusts the number of nodes in your node groups based on the observed CPU, memory, or custom metrics demand to ensure that there are sufficient resources to run your pods.

**Example of Metrics for Scaling:**

* 1. CPU and Memory Utilization: The most common metrics used for scaling. Scale out when the utilization goes beyond 80%, and scale in when it drops below 20%.
  2. Custom Metrics: Based on specific application or business metrics, such as queue length or transaction volume, using external metrics sources like Prometheus.

# EKS | Disaster Recovery Strategy (DR)

**Objectives**

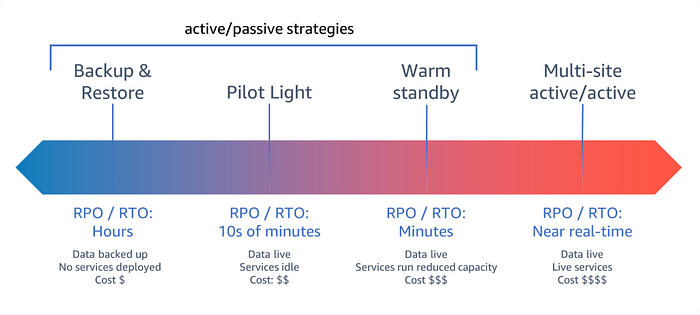
The DR strategy aims to minimize downtime and data loss during catastrophic events, ensuring a quick and efficient recovery.

**Backup and Restore Procedures**

* **Data Backups**: Regularly back up persistent data using EBS snapshots and EFS backups.
* **Control Plane Backups**: While AWS manages the control plane, consider additional backups for critical configuration data.

**DR Architecture**

* **Cross-Region Replication**: Replicate critical data and services to a different AWS region.
* **Backup Storage**: Store backups in durable storage solutions like Amazon S3.
* **The EKS Disaster Recovery project ensures the continuity of Amazon EKS Kubernetes clusters by implementing a multi-region deployment strategy, backup and restore mechanisms, and automated failover procedures.**



**DR Implementation Steps**

1. **Configure Automated Backups**: Set up regular snapshots for EBS volumes and backups for EFS.
2. **Set Up Cross-Region Replication**: Use AWS services like S3 Cross-Region Replication and AWS Backup.
3. **Implement DR Runbooks**: Document the steps to restore services and data in a different region.

**Testing and Validation**

* **Simulate Failures**: Regularly test failure scenarios to ensure the DR plan is effective.
* **Restore from Backups**: Practice restoring from backups to verify data integrity and recovery time.
* **Cross-Region Failover**: Test the failover process to a different region to ensure seamless recovery.

**Best Practices at PDS**

**Backup and Restore Procedures**

**Data Backups**

**Best Practices:**

* **Regular Backups: Schedule regular backups of EBS volumes and EFS filesystems to capture critical data.**
* **Automated Processes: Automate backup processes to reduce the risk of human error.**
* **Retention Policies: Define and implement retention policies to manage backup lifecycle and storage costs.**

**Example Implementation:**

**EBS Snapshots:**

**aws ec2 create-snapshot --volume-id vol-1234567890abcdef0 --description "EKS cluster EBS backup"**

**EFS Backups:**

**aws backup create-backup-plan --backup-plan '{**

**"BackupPlanName": "EFSBackupPlan",**

**"Rules": [**

**{**

**"RuleName": "DailyBackup",**

**"TargetBackupVaultName": "EFSBackupVault",**

**"ScheduleExpression": "cron(0 12 \* \* ? \*)",**

**"StartWindowMinutes": 60,**

**"CompletionWindowMinutes": 180,**

**"Lifecycle": {**

**"MoveToColdStorageAfterDays": 30,**

**"DeleteAfterDays": 365**

**},**

**"RecoveryPointTags": {**

**"Environment": "Production"**

**}**

**}**

**]**

**}'**

**Control Plane Backups:**

**Best Practices:**

* **Configuration Backups: Regularly back up critical configuration data such as Kubernetes manifests and etcd snapshots.**
* **Version Control: Use version control systems (e.g., Git) to store and manage configuration files.**

**Example Implementation:**

**# Example script to back up Kubernetes manifests**

**kubectl get all --all-namespaces -o yaml > all-k8s-resources-backup.yaml**

**# Backup etcd snapshots (run on the etcd host)**

**ETCDCTL\_API=3 etcdctl snapshot save /backup/etcd-snapshot.db**

**DR Architecture**

**Cross-Region Replication**

**Best Practices:**

* **Data Replication: Replicate critical data and services to a different AWS region using AWS services.**
* **Multi-Region Deployment: Deploy critical components of your EKS cluster across multiple regions to enhance resilience.**

**Example Implementation:**

**S3 Cross-Region Replication:**

**# S3 bucket replication configuration**

**{**

**"Role": "arn:aws:iam::123456789012:role/s3-replication-role",**

**"Rules": [**

**{**

**"Status": "Enabled",**

**"Prefix": "",**

**"Destination": {**

**"Bucket": "arn:aws:s3:::destination-bucket"**

**}**

**}**

**]**

**}**

**AWS Backup Cross-Region:**

**aws backup create-backup-vault --backup-vault-name "cross-region-backup-vault" --region us-west-2**

**Backup Storage**

**Best Practices:**

* **Durable Storage: Store backups in Amazon S3, which provides durability and availability.**
* **Lifecycle Policies: Use lifecycle policies to transition objects to cheaper storage classes and eventually delete them.**

**Example Implementation:**

**aws s3 cp all-k8s-resources-backup.yaml s3://my-backup-bucket/k8s-backups/**

# 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.

**Best practices at PDS:**

**1. VPC Creation**

**Recommendation: Create a VPC that spans at least three AZs to ensure high availability. Use separate subnets for each AZ to manage network traffic and resource distribution effectively.**

**Example:**

**aws ec2 create-vpc --cidr-block 10.0.0.0/16**

**aws ec2 create-subnet --vpc-id vpc-abcde012 --cidr-block 10.0.1.0/24 --availability-zone us-west-2a**

**aws ec2 create-subnet --vpc-id vpc-abcde012 --cidr-block 10.0.2.0/24 --availability-zone us-west-2b**

**aws ec2 create-subnet --vpc-id vpc-abcde012 --cidr-block 10.0.3.0/24 --availability-zone us-west-2c**

**2. EKS Cluster Setup**

**Recommendation: Deploy the EKS cluster with control plane instances in multiple AZs to enhance fault tolerance. This ensures that the cluster remains operational even if one AZ goes down.**

**aws eks create-cluster --name my-cluster --role-arn arn:aws:iam::123456789012:role/eksServiceRole --resources-vpc-config subnetIds=subnet-abcde01234,subnet-abcdef56789,subnet-abcdefgh123,securityGroupIds=sg-78912345**

**3. Worker Node Configuration**

**Recommendation: Use Auto Scaling Groups (ASGs) for worker nodes across multiple AZs. Configure the ASGs to automatically replace unhealthy instances and scale based on demand.**

**Example:**

**aws autoscaling create-auto-scaling-group --auto-scaling-group-name my-asg --launch-configuration-name my-launch-config --min-size 1 --max-size 10 --desired-capacity 3 --vpc-zone-identifier "subnet-abcde01234,subnet-abcdef56789,subnet-abcdefgh123"**

**4. Network Configuration**

**Recommendation: Configure security groups and network access control lists (NACLs) to control inbound and outbound traffic effectively. Set up routing tables to ensure optimal routing of traffic between AZs.**

**Example:**

* **Security Groups:**

**aws ec2 create-security-group --group-name my-sg --description "My security group for EKS" --vpc-id vpc-abcde012**

**aws ec2 authorize-security-group-ingress --group-id sg-903004f8 --protocol tcp --port 443 --cidr 0.0.0.0/0**

* **NACLs:**

**aws ec2 create-network-acl --vpc-id vpc-abcde012**

**aws ec2 create-network-acl-entry --network-acl-id acl-abc1234d --ingress --rule-number 100 --protocol tcp --port-range From=1024,To=65535 --cidr-block 0.0.0.0/0 --rule-action allow**

* **Routing Tables:**

**aws ec2 create-route-table --vpc-id vpc-abcde012**

**aws ec2 create-route --route-table-id rtb-abc1234d --destination-cidr-block 0.0.0.0/0 --gateway-id igw-abc1234**

**aws ec2 associate-route-table --route-table-id rtb-abc1234d --subnet-id subnet-abcde01234**

# EKS | Security considerations

**Network Security**

* **Network Policies**: Implement Kubernetes network policies to control traffic between pods.
* **PrivateLink and VPC Endpoints**: Use AWS PrivateLink and VPC endpoints to securely connect services.

**Identity and Access Management**

* **IAM Roles and Policies**: Define IAM roles and policies for the EKS cluster and worker nodes.
* **RBAC**: Use Kubernetes Role-Based Access Control (RBAC) to manage permissions within the cluster.

**Data Encryption**

* **Encryption at Rest**: Use AWS KMS to encrypt data stored in EBS, EFS, and S3.
* **Encryption in Transit**: Use TLS to encrypt data in transit between services.

**Best practices at PDS:**

**Network Policies**

**Best Practice:** Implement granular Kubernetes network policies to restrict traffic between pods based on specific needs and security requirements.

**Example:**

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: restrict-db-access

spec:

podSelector:

matchLabels:

role: database

ingress:

- from:

- podSelector:

matchLabels:

role: frontend

ports:

- protocol: TCP

port: 5432

This policy allows only pods with the **frontend** role to access pods with the **database** role over TCP port 5432.

**PrivateLink and VPC Endpoints**

**Best Practice:** Use AWS PrivateLink and VPC endpoints to securely access AWS services without exposing data to the public internet.

**Example:** Create a VPC endpoint for Amazon S3 to ensure all data traffic between your EKS cluster and S3 stays within the AWS network.

aws ec2 create-vpc-endpoint --vpc-id vpc-abc123 --service-name com.amazonaws.us-west-2.s3 --vpc-endpoint-type Interface --subnet-ids subnet-abc123 --security-group-ids sg-abc123

**IAM Roles and Policies**

**Best Practice:** Define fine-grained IAM roles and attach policies that strictly follow the principle of least privilege.

**Example:** Create an IAM role for EKS worker nodes and attach policies that only allow actions required by the worker nodes.

**RBAC**

**Best Practice:** Utilize Kubernetes RBAC to control user and application access to Kubernetes resources. **Example:** Create a Role and RoleBinding to allow reading secrets only in a specific namespace.

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: mynamespace

name: secret-reader

rules:

- apiGroups: [""]

resources: ["secrets"]

verbs: ["get", "watch", "list"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: read-secrets

namespace: mynamespace

subjects:

- kind: User

name: jane

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: secret-reader

apiGroup: rbac.authorization.k8s.io

**Data Encryption**

**Encryption at Rest**

**Best Practice:** Encrypt sensitive data at rest using AWS KMS to secure data and meet compliance requirements. **Example:** Enable encryption on an EBS volume using an AWS KMS key.

Bash

aws ec2 create-volume --size 100 --region us-west-2 --availability-zone us-west-2a --volume-type gp2 --encrypted --kms-key-id alias/my-key

**Encryption in Transit**

**Best Practice:** Use TLS for all internal and external communications to protect data in transit.

**Example:** Configure Istio to enforce mTLS (mutual TLS) for all communications within the cluster.

apiVersion: networking.istio.io/v1beta1

kind: PeerAuthentication

metadata:

name: default

namespace: Istio-system

spec:

mtls:

mode: STRICT

# 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.

# EKS | Monitoring and Logging

**Monitoring Tools and Metrics**

* **CloudWatch**: Use Amazon CloudWatch for monitoring EKS clusters.
* **Prometheus and Grafana**: Set up Prometheus for metrics collection and Grafana for visualization.

**Logging and Alerting**

* **CloudTrail**: Enable AWS CloudTrail for auditing API calls.
* **ELK Stack**: Use Elasticsearch, Logstash, and Kibana for centralized logging and analysis.

# EKS | Cost considerations

**Cost Estimation**

* **Multi-AZ Deployment**: Estimate costs for running resources in multiple AZs.
* **Backup and DR**: Calculate costs for storage and data transfer related to backups and DR.

**Cost Optimization Strategies**

* **Right-Sizing**: Continuously monitor and adjust the size of instances and resources.
* **Spot Instances**: Use EC2 Spot Instances for cost savings on non-critical workloads.

# EKS | Benefits of HA, DR and AZ

**High Availability (HA)**

**Benefits:**

* **Increased Uptime**: Ensures that applications remain available even if one or more components fail.
* **Improved Performance**: Load balancing distributes traffic across multiple nodes, improving performance and response times.
* **Enhanced User Experience**: Consistent availability leads to a better user experience and higher satisfaction.

**Examples:**

* **E-commerce Platform**: An online store that continues to process transactions and handle customer queries even during a node failure, preventing loss of sales.
* **Healthcare System**: Ensuring that patient data and critical health applications remain accessible, which is vital for providing timely medical care.

**Disaster Recovery (DR)**

**Benefits:**

* **Data Protection**: Regular backups and replication protect against data loss.
* **Quick Recovery**: Enables rapid recovery of services and data after a disaster, minimizing downtime.
* **Compliance**: Meets regulatory requirements for data protection and disaster recovery.

**Examples:**

* **Financial Services**: Banks and financial institutions that can quickly restore services after a cyberattack or natural disaster, ensuring continuous access to financial data.
* **Educational Institutions**: Universities that can recover student records and online learning platforms after a major outage.

**Availability Zones (AZ)**

**Benefits:**

* **Fault Isolation**: Isolates faults to a single AZ, preventing widespread outages.
* **Resilience**: Distributes resources across multiple AZs, increasing fault tolerance.
* **Cost Efficiency**: Optimizes costs by using resources more efficiently across different AZs.

**Examples:**

* **SaaS Applications**: Software-as-a-Service providers can ensure their applications remain available to users globally, even if one AZ goes down.

# EKS | Benefits for Dental services client

**High Availability (HA)**

**Benefits:**

* **Increased Uptime and Reliability**: Ensures that critical dental applications such as appointment scheduling, patient records management, and telehealth services remain available even during infrastructure failures. This minimizes disruptions and maintains patient trust.
* **Improved Performance**: Load balancing across multiple nodes improves application responsiveness, providing a seamless experience for both patients and staff.
* **Enhanced Patient Experience**: Consistent availability of online services such as appointment bookings and patient portals enhances patient satisfaction and loyalty.

**Examples:**

* **Appointment Scheduling**: Even if one server node fails, patients can still book, reschedule, and cancel appointments without any interruption.
* **Patient Records Access**: Dentists and staff can access patient records and treatment history without delays, ensuring continuous care and efficient operations.

**Disaster Recovery (DR)**

**Benefits:**

* **Data Protection**: Regular backups of patient records, billing information, and treatment plans ensure data integrity and protection against data loss.
* **Quick Recovery**: Enables rapid restoration of services and data after a disaster, minimizing downtime and ensuring business continuity.
* **Regulatory Compliance**: Meets stringent healthcare regulations and data protection laws, ensuring that patient data is securely stored and recoverable.

**Examples:**

* **Data Backups**: Regular snapshots and backups of patient records to Amazon S3 ensure that in the event of data corruption or accidental deletion, data can be quickly restored.
* **Cross-Region Replication**: Critical patient data and application configurations are replicated to a different AWS region, ensuring availability even in case of a regional disaster.

**Availability Zones (AZ)**

**Benefits:**

* **Fault Isolation**: By distributing resources across multiple AZs, a failure in one AZ does not impact the overall system, ensuring continuous operation.
* **Resilience and Fault Tolerance**: Resources distributed across multiple AZs can handle high traffic loads and continue to operate seamlessly even if one AZ goes down.
* **Cost Efficiency**: Optimizes costs by efficiently utilizing resources across different AZs, reducing the need for over-provisioning.

**Examples:**

* **Telehealth Services**: Dental telehealth services remain operational even if one AZ experiences an outage, ensuring that virtual consultations can continue without interruption.
* **Patient Portal**: The patient portal remains accessible from different geographic locations, providing reliable access to patients for viewing their treatment history, scheduling appointments, and communicating with their dentist.

# EKS | Conclusion

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.

# 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