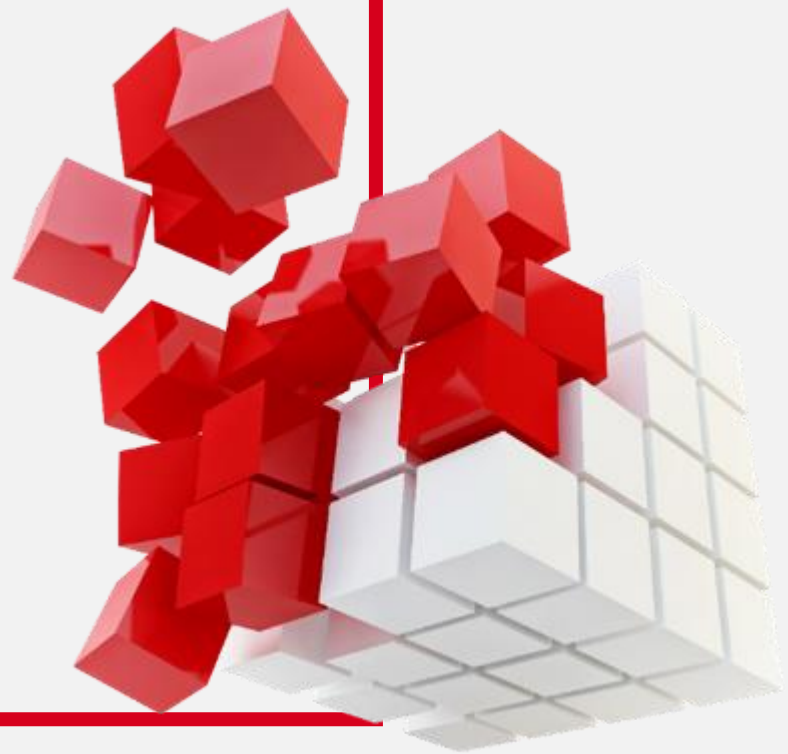
# AWS Best Practices
# v1.0



**Minh Tran/ NashTech**

Jun/2023

Nash Tech.

# Agenda

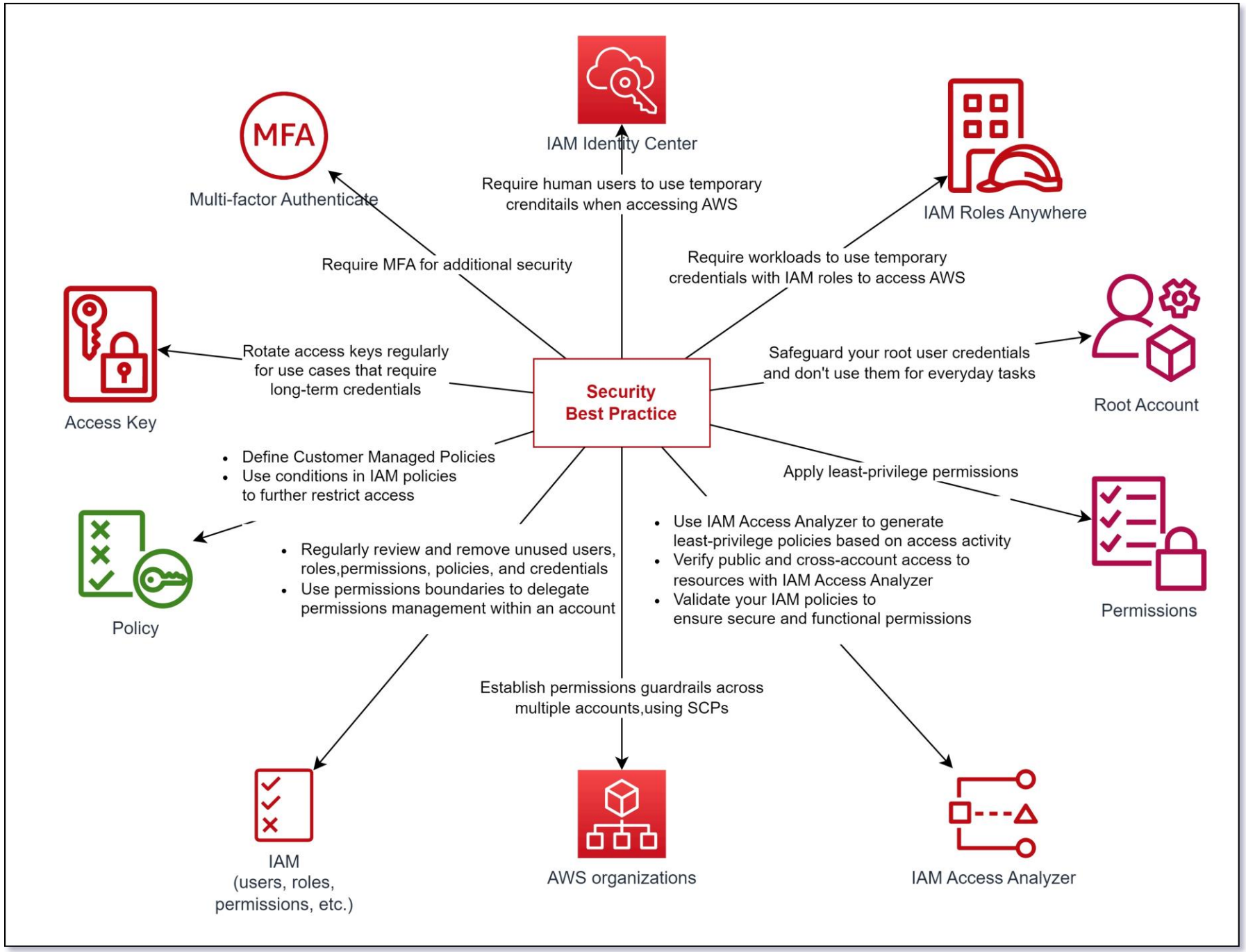
1. AWS Identity and Access Management (IAM)
2. AWS Organizations
3. Amazon EC2 (Elastic Cloud Computing)
4. Amazon RDS (Relational Database Services)
5. Amazon Aurora
6. Amazon S3 (Simple Storage Service)
7. Amazon Lambda
8. Amazon Cognito
9. Amazon SNS (Simple Notification Service)
10. Amazon VPC (Virtual Private Cloud)
11. Amazon Kinesis
12. Amazon Auto-scaling
13. Dynamo DB
14. Amazon SQS (Simple Queue Service)
15. AWS CloudFormation

# AWS Identity and Access Management (IAM)

Nash Tech.

# Security Best Practice

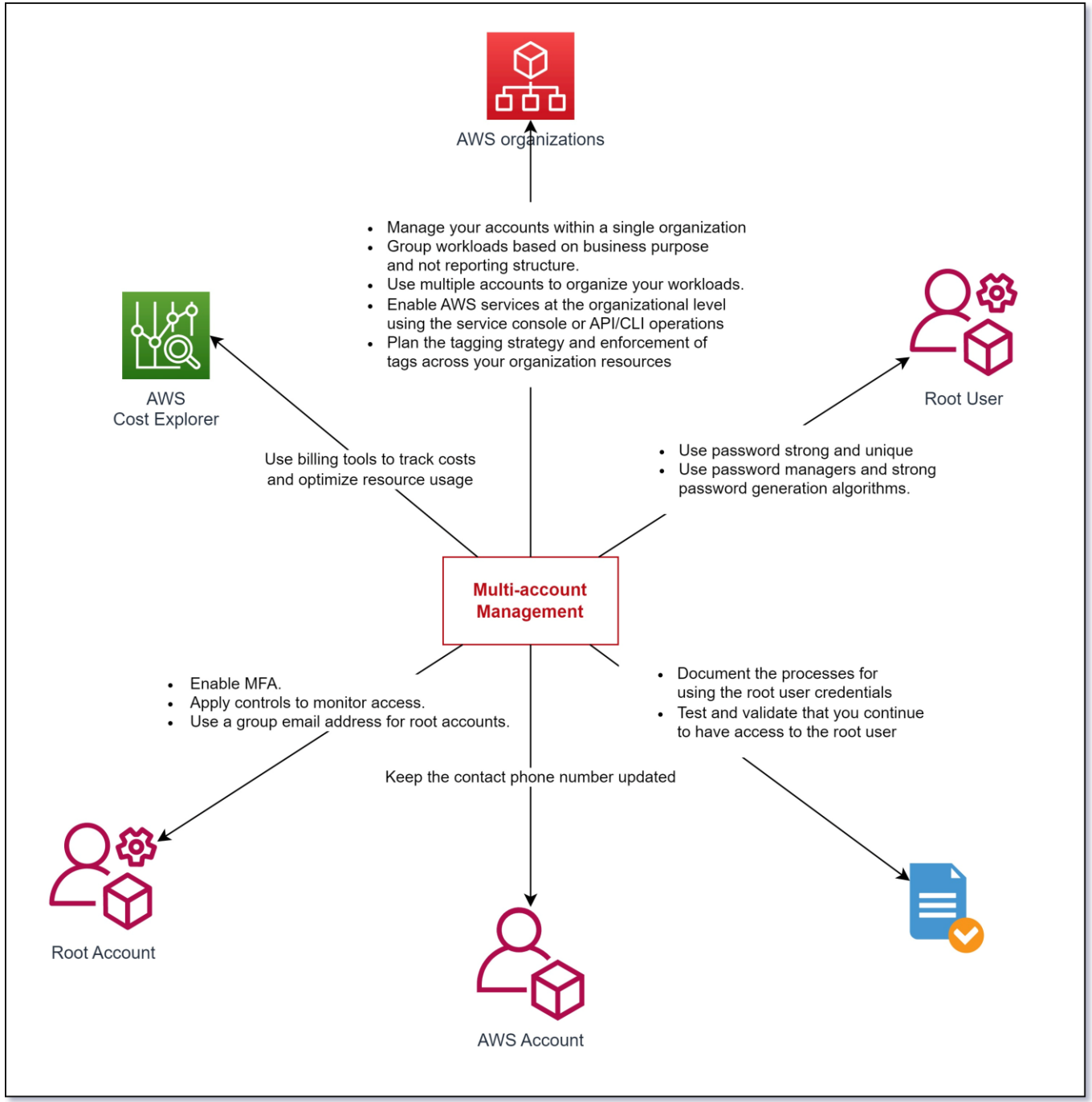
- Require human users to use federation with an identity provider to access AWS using temporary credentials
- Require workloads to use temporary credentials with IAM roles to access AWS
- Require multi-factor authentication (MFA)
- Rotate access keys regularly for use cases that require long-term credentials
- Safeguard your root user credentials and don't use them for everyday tasks
- Apply least-privilege permissions
- Get started with AWS managed policies and move toward least-privilege permissions
- Use IAM Access Analyzer to generate least-privilege policies based on access activity
- Regularly review and remove unused users, roles, permissions, policies, and credentials
- Use conditions in IAM policies to further restrict access
- Verify public and cross-account access to resources with IAM Access Analyzer
- Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions
- Establish permissions guardrails across multiple accounts
- Use permissions boundaries to delegate permissions management within an account


MFA
Multi-factor Authenticate
IAM Identity Center
Require human users to use temporary crenditails when accessing AWS
IAM Roles Anywhere
Require MFA for additional security
Require workloads to use temporary credentials with IAM roles to access AWS
Access Key
Rotate access keys regularly for use cases that require long-term credentials
Safeguard your root user credentials and don't use them for everyday tasks
Root Account
Security Best Practice
Define Customer Managed Policies
Use conditions in IAM policies to further restrict access
Apply least-privilege permissions
Policy
Regularly review and remove unused users, roles,permissions, policies, and credentials
Use permissions boundaries to delegate permissions management within an account
Use IAM Access Analyzer to generate least-privilege policies based on access activity
Verify public and cross-account access to resources with IAM Access Analyzer
Validate your IAM policies to ensure secure and functional permissions
Permissions
Establish permissions guardrails across multiple accounts,using SCPs
IAM (users, roles, permissions, etc.)
AWS organizations
IAM Access Analyzer


AWS Organizations
Nash
Tech.

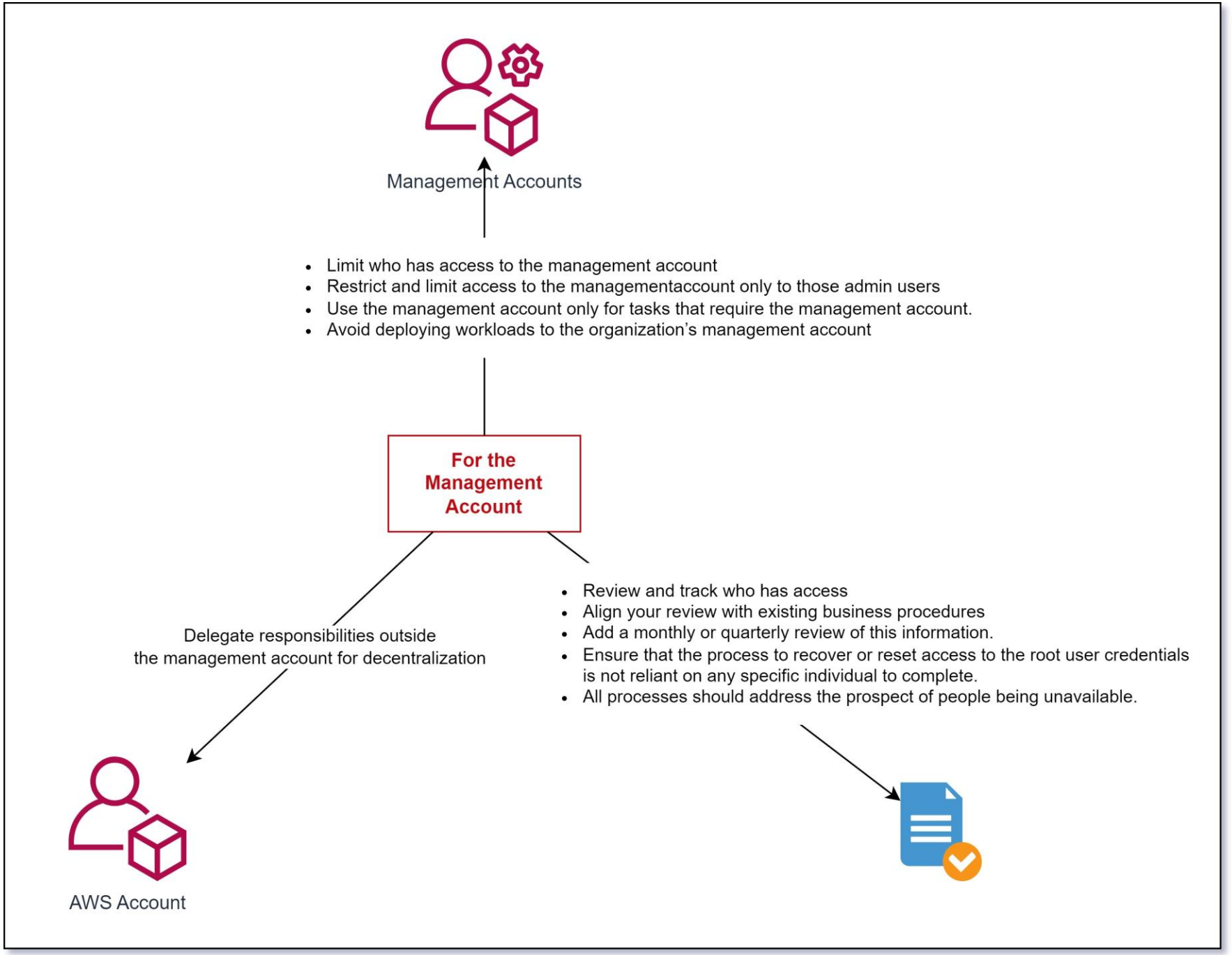
# Best practices for multi-account management

- Manage your accounts within a single organization
- Use a strong password for the root user
- Document the processes for using the root user credentials
- Enable MFA for your root user credentials
- Apply controls to monitor access to the root user credentials
- Keep the contact phone number updated
- Use a group email address for root accounts
- Group workloads based on business purpose and not reporting structure
- Use multiple accounts to organize your workloads
- Enable AWS services at the organizational level using the service console or API/CLI operations
- Use billing tools to track costs and optimize resource usage
- Plan the tagging strategy and enforcement of tags across your organization’s resources


AWS organizations
Manage your accounts within a single organization
Group workloads based on business purpose and not reporting structure.
Use multiple accounts to organize your workloads.
Enable AWS services at the organizational level using the service console or API/CLI operations
Plan the tagging strategy and enforcement of tags across your organization resources
AWS Cost Explorer
Use billing tools to track costs and optimize resource usage
Root User
Use password strong and unique
Use password managers and strong password generation algorithms.
Multi-account Management
Enable MFA.
Apply controls to monitor access.
Use a group email address for root accounts.
Root Account
Keep the contact phone number updated
AWS Account
Document the processes for using the root user credentials
Test and validate that you continue to have access to the root user

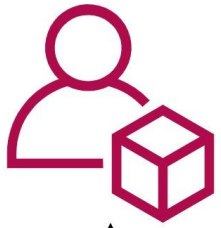
# Best practices for the management account

- Limit who has access to the management account
- Review and track who has access
- Use the management account only for tasks that require the management account
- Avoid deploying workloads to the organization’s management account
- Delegate responsibilities outside the management account for decentralization


Management Accounts
Limit who has access to the management account
Restrict and limit access to the managementaccount only to those admin users
Use the management account only for tasks that require the management account.
Avoid deploying workloads to the organization's management account
For the Management Account
Delegate responsibilities outside the management account for decentralization
Review and track who has access
Align your review with existing business procedures
Add a monthly or quarterly review of this information.
Ensure that the process to recover or reset access to the root user credentials is not reliant on any specific individual to complete.
All processes should address the prospect of people being unavailable.
AWS Account

# Best practices for member accounts

- Define account name and attributes
- Efficiently scale your environment and account usage
- Use an SCP to restrict what the root user in your member accounts can do

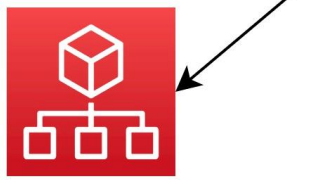


AWS Account

- Define account name and attributes.
- Before creating new accounts, make sure accounts for similar needs do not already exist.

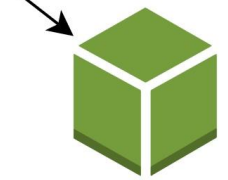
**For the Member Account**

- Before closing accounts, note that they are subject to close account quota limits.
- Check Quotas for AWS Organizations.
- Check CloseAccountAPI limits.



AWS organizations

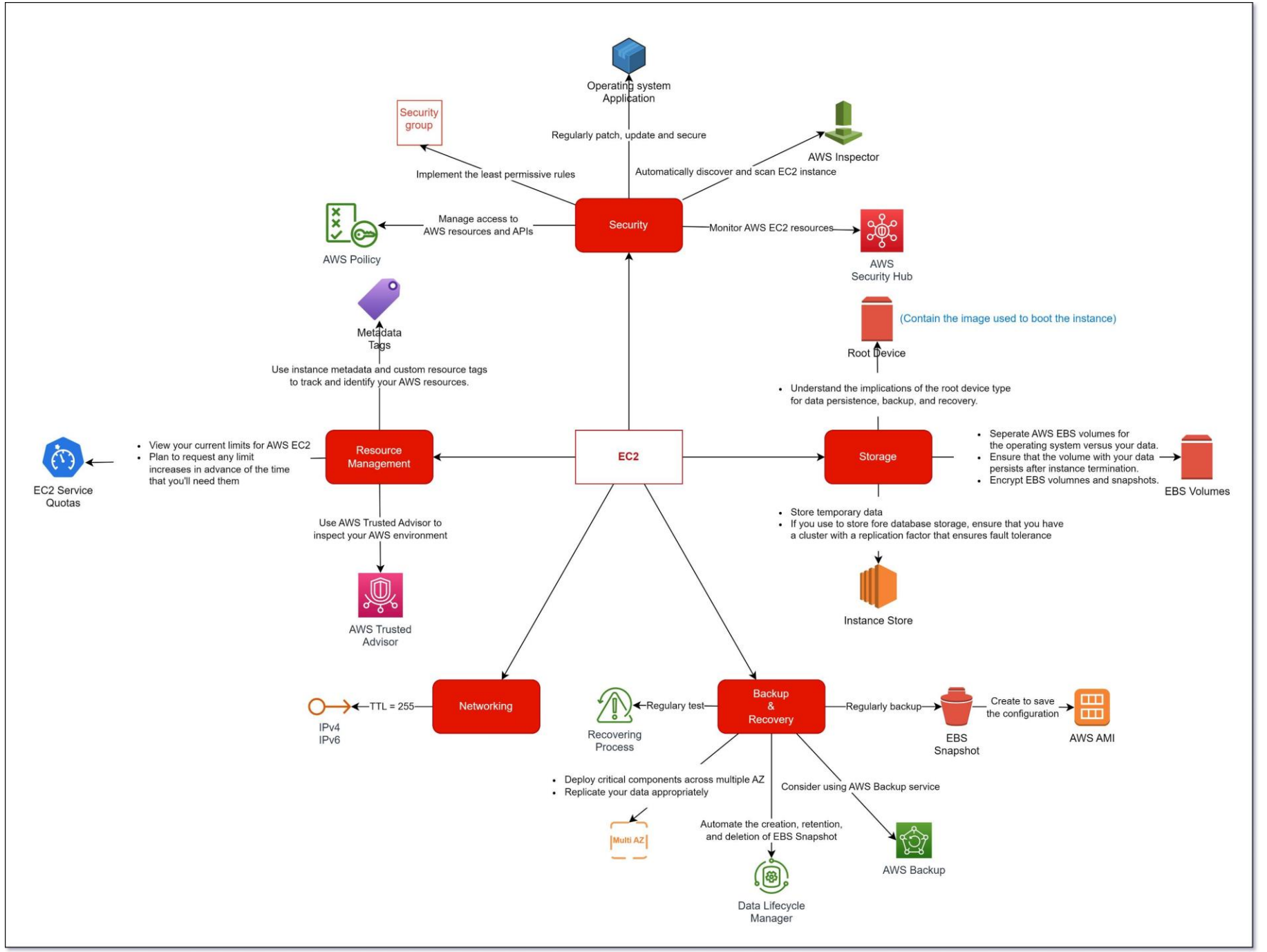
- Clean up unneeded resources or workloads from the accounts
- Save the Accounts for a future use, if you're planning to reuse the accounts.



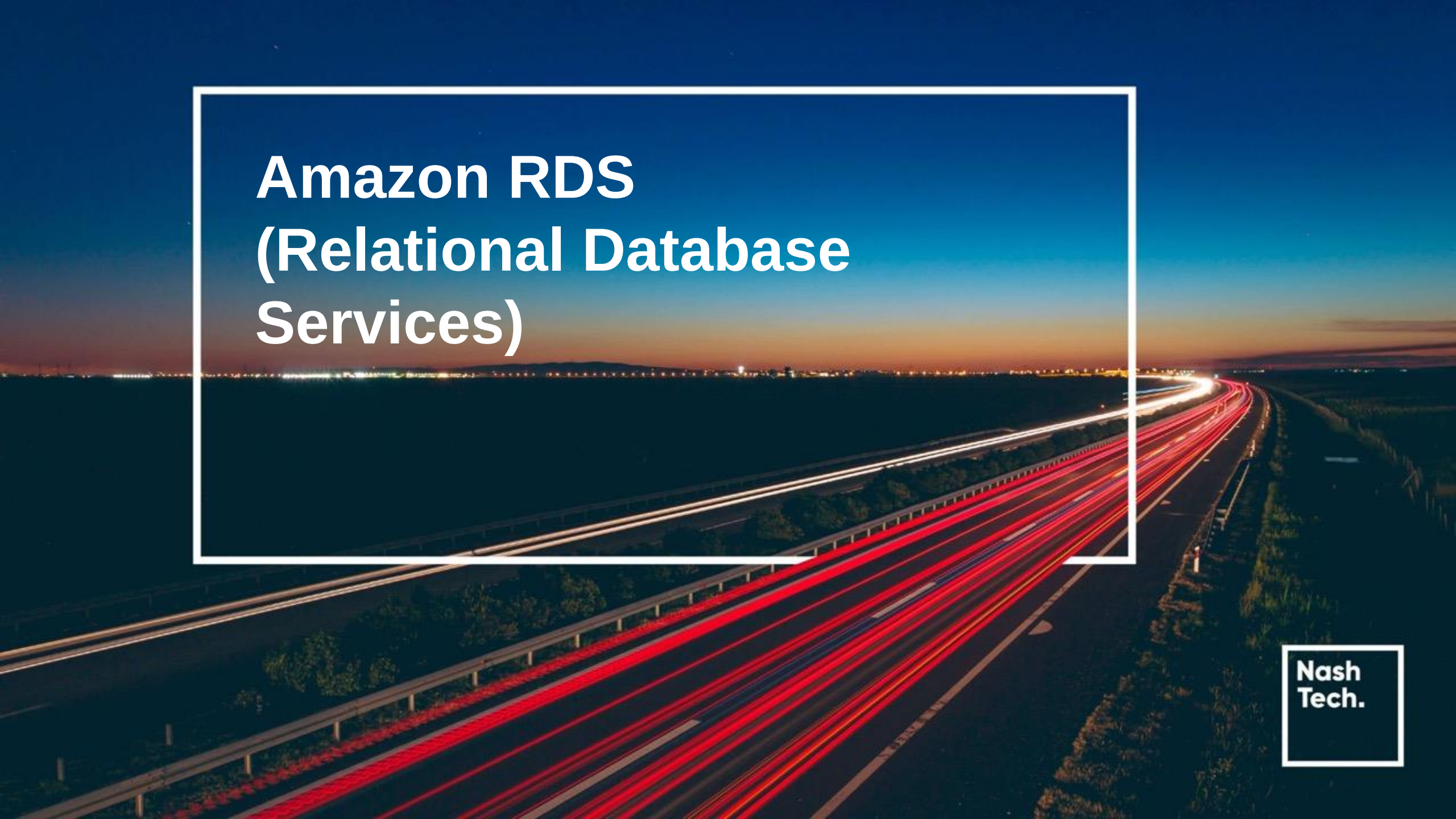
AWS Resources

# Amazon EC2 (Elastic Cloud Computing)

Nash Tech.


Operating system
Application
Regularly patch, update and secure
Security group
Implement the least permissive rules
AWS Inspector
Automatically discover and scan EC2 instance
Security
Manage access to
AWS resources and APIs
AWS Poilicy
Monitor AWS EC2 resources
AWS
Security Hub
Metadata
Tags
Use instance metadata and custom resource tags
to track and identify your AWS resources.
(Contain the image used to boot the instance)
Root Device
Understand the implications of the root device type
for data persistence, backup, and recovery.
EC2 Service
Quotas
View your current limits for AWS EC2
Plan to request any limit
increases in advance of the time
that you'll need them
Resource
Management
EC2
Storage
Seperate AWS EBS volumes for
the operating system versus your data.
Ensure that the volume with your data
persists after instance termination.
Encrypt EBS volumnes and snapshots.
EBS Volumes
Use AWS Trusted Advisor to
inspect your AWS environment
Store temporary data
If you use to store fore database storage, ensure that you have
a cluster with a replication factor that ensures fault tolerance
AWS Trusted
Advisor
Instance Store
IPv4
IPv6
TTL = 255
Networking
Recovering
Process
Regulary test
Backup
&
Recovery
Regularly backup
EBS
Snapshot
Create to save
the configuration
AWS AMI
Deploy critical components across multiple AZ
Replicate your data appropriately
Multi AZ
Automate the creation, retention,
and deletion of EBS Snapshot
Consider using AWS Backup service
Data Lifecycle
Manager
AWS Backup

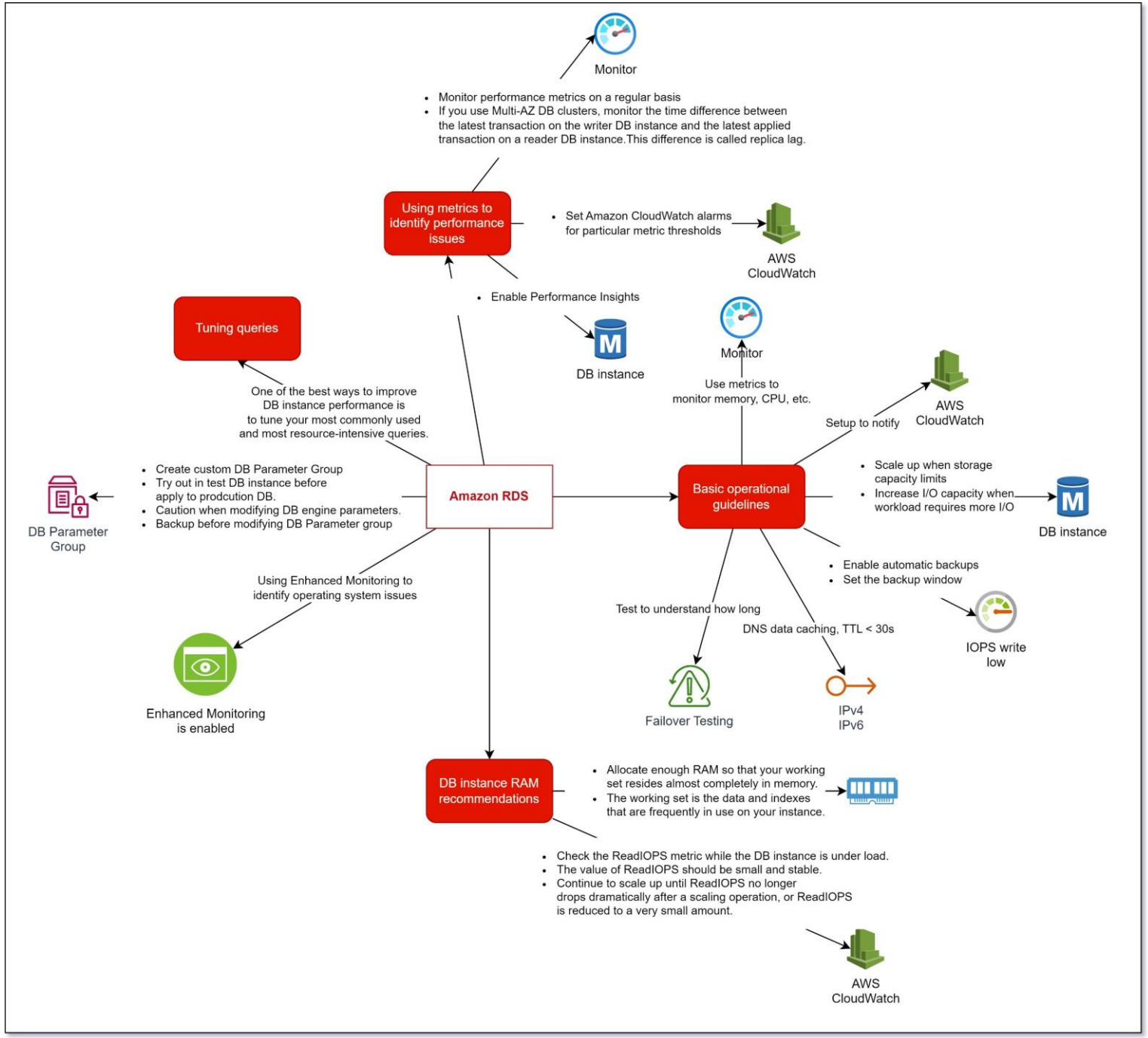
# Amazon RDS (Relational Database Services)

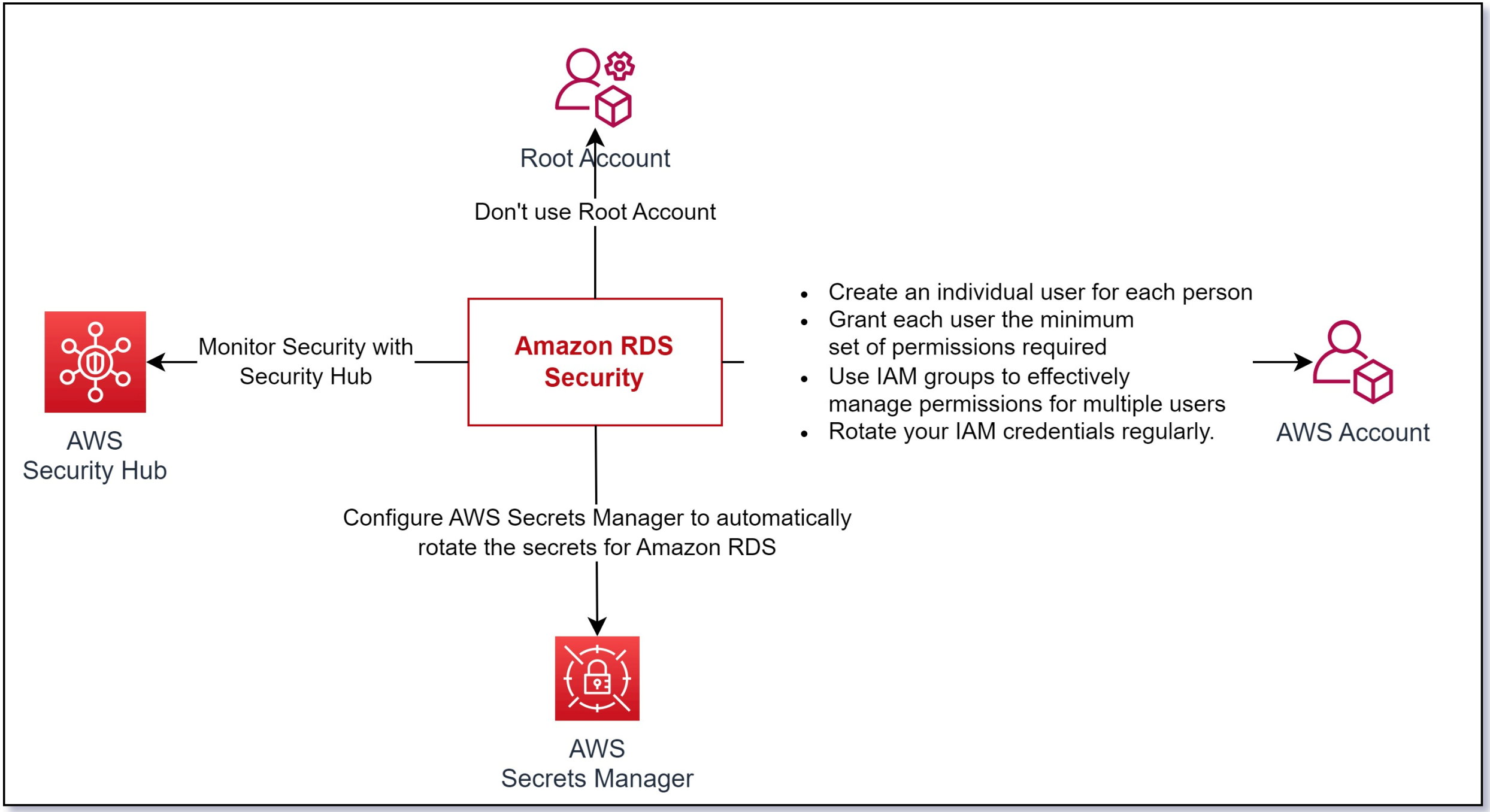


Nash Tech.

# Best practices for Amazon RDS

- Amazon RDS basic operational guidelines
- DB instance RAM recommendations
- Using Enhanced Monitoring to identify operating system issues
- Using metrics to identify performance issues
- Tuning queries
- Best practices for working with MySQL
- Best practices for working with MariaDB
- Best practices for working with Oracle
- Best practices for working with PostgreSQL
- Best practices for working with SQL Server
- Working with DB parameter groups
- Best practices for automating DB instance creation
- Amazon RDS new features and best practices presentation video


Monitor
Monitor performance metrics on a regular basis
If you use Multi-AZ DB clusters, monitor the time difference between the latest transaction on the writer DB instance and the latest applied transaction on a reader DB instance.This difference is called replica lag.
Using metrics to identify performance issues
Set Amazon CloudWatch alarms for particular metric thresholds
AWS CloudWatch
Enable Performance Insights
DB instance
Tuning queries
One of the best ways to improve DB instance performance is to tune your most commonly used and most resource-intensive queries.
Monitor
Use metrics to monitor memory, CPU, etc.
Setup to notify
AWS CloudWatch
Create custom DB Parameter Group
Try out in test DB instance before apply to prodcution DB.
Caution when modifying DB engine parameters.
Backup before modifying DB Parameter group
DB Parameter Group
Amazon RDS
Basic operational guidelines
Scale up when storage capacity limits
Increase I/O capacity when workload requires more I/O
DB instance
Using Enhanced Monitoring to identify operating system issues
Enhanced Monitoring is enabled
Enable automatic backups
Set the backup window
IOPS write low
Test to understand how long
DNS data caching, TTL < 30s
Failover Testing
IPv4
IPv6
DB instance RAM recommendations
Allocate enough RAM so that your working set resides almost completely in memory.
The working set is the data and indexes that are frequently in use on your instance.
Check the ReadIOPS metric while the DB instance is under load.
The value of ReadIOPS should be small and stable.
Continue to scale up until ReadIOPS no longer drops dramatically after a scaling operation, or ReadIOPS is reduced to a very small amount.
AWS CloudWatch

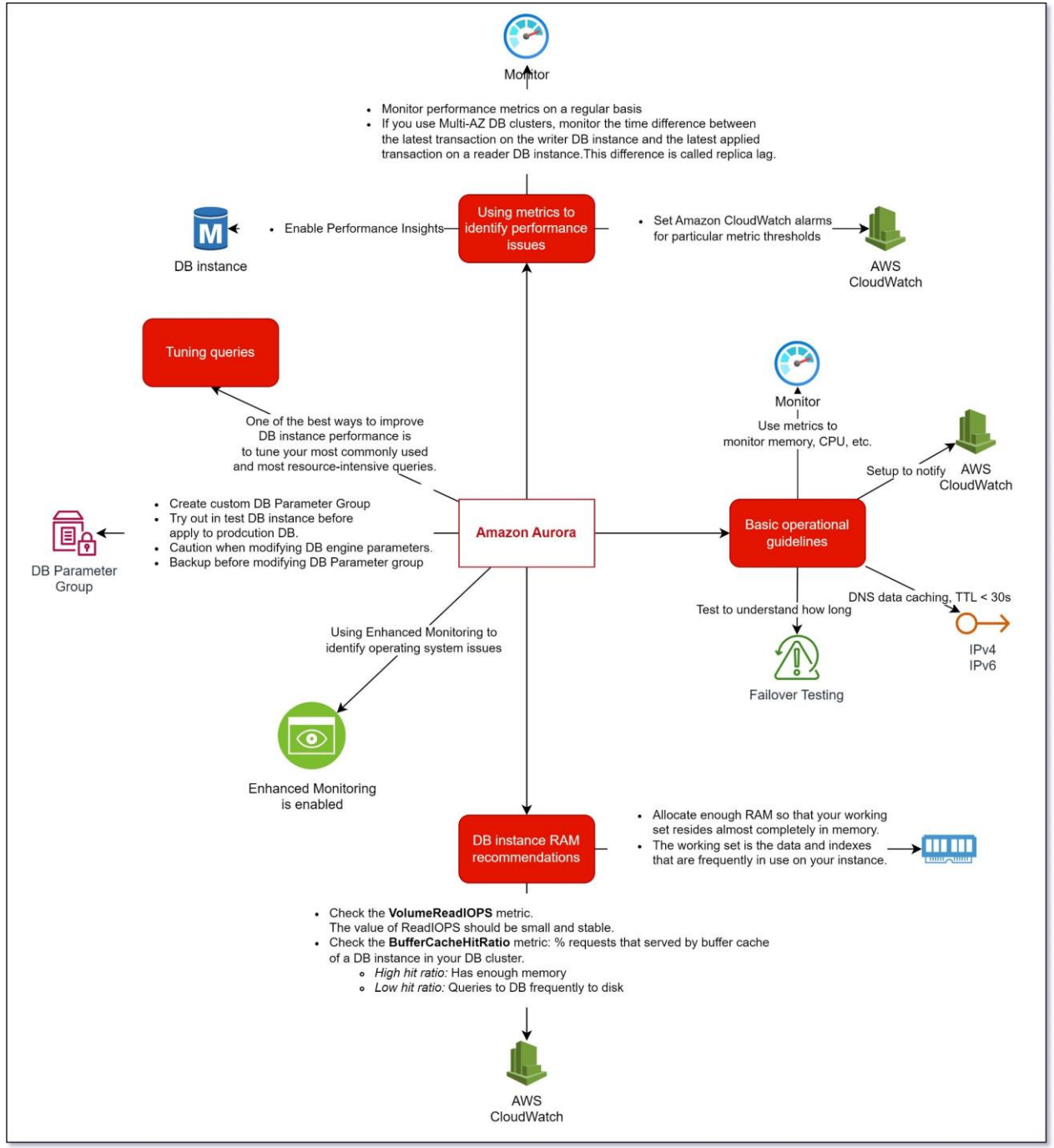

Root Account
Don't use Root Account
Amazon RDS Security
Monitor Security with Security Hub
AWS Security Hub
Create an individual user for each person
Grant each user the minimum set of permissions required
Use IAM groups to effectively manage permissions for multiple users
Rotate your IAM credentials regularly.
AWS Account
Configure AWS Secrets Manager to automatically rotate the secrets for Amazon RDS
AWS Secrets Manager

# Amazon Aurora

Nash Tech.

# Best practices with Amazon Aurora

- Basic operational guidelines for Amazon Aurora
- DB instance RAM recommendations
- Monitoring Amazon Aurora
- Working with DB parameter groups and DB cluster parameter groups
- Amazon Aurora best practices video


Monitor
Monitor performance metrics on a regular basis
If you use Multi-AZ DB clusters, monitor the time difference between the latest transaction on the writer DB instance and the latest applied transaction on a reader DB instance.This difference is called replica lag.
Using metrics to identify performance issues
Enable Performance Insights
DB instance
Set Amazon CloudWatch alarms for particular metric thresholds
AWS CloudWatch
Tuning queries
One of the best ways to improve DB instance performance is to tune your most commonly used and most resource-intensive queries.
Monitor
Use metrics to monitor memory, CPU, etc.
Setup to notify
AWS CloudWatch
Create custom DB Parameter Group
Try out in test DB instance before apply to prodcution DB.
Caution when modifying DB engine parameters.
Backup before modifying DB Parameter group
DB Parameter Group
Amazon Aurora
Basic operational guidelines
Test to understand how long
DNS data caching, TTL < 30s
IPv4
IPv6
Failover Testing
Using Enhanced Monitoring to identify operating system issues
Enhanced Monitoring is enabled
DB instance RAM recommendations
Allocate enough RAM so that your working set resides almost completely in memory.
The working set is the data and indexes that are frequently in use on your instance.
Check the VolumeReadIOPS metric.
The value of ReadIOPS should be small and stable.
Check the BufferCacheHitRatio metric: % requests that served by buffer cache of a DB instance in your DB cluster.
High hit ratio: Has enough memory
Low hit ratio: Queries to DB frequently to disk
AWS CloudWatch

# Best practices with Amazon Aurora MySQL

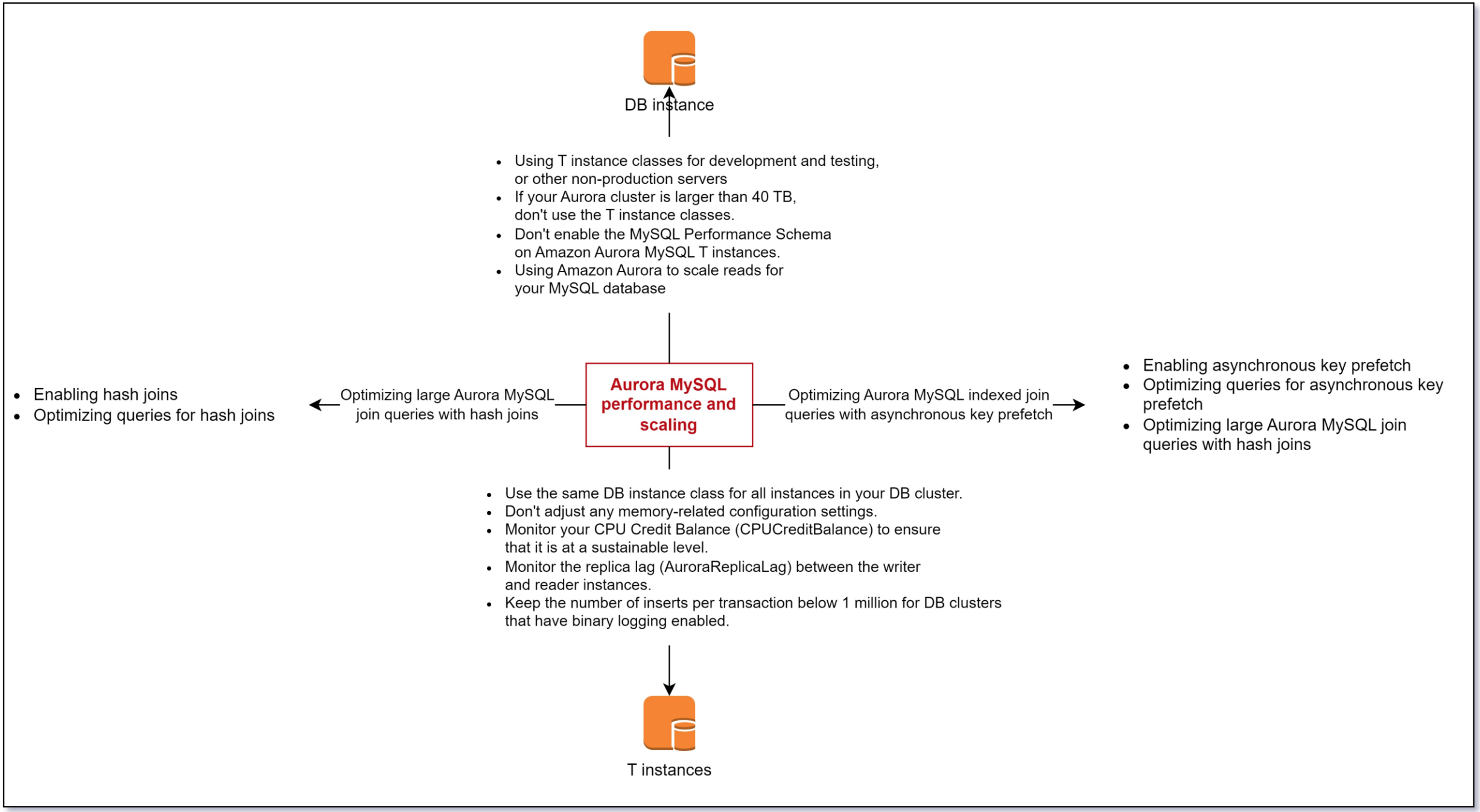
- Determining which DB instance you are connected to
- Best practices for Aurora MySQL performance and scaling
  - Using T instance classes for development and testing
  - Optimizing Aurora MySQL indexed join queries with asynchronous key prefetch
    - Enabling asynchronous key prefetch
    - Optimizing queries for asynchronous key prefetch
  - Optimizing large Aurora MySQL join queries with hash joins
    - Enabling hash joins
    - Optimizing queries for hash joins
  - Using Amazon Aurora to scale reads for your MySQL database
- Best practices for Aurora MySQL high availability
  - Using Amazon Aurora for Disaster Recovery with your MySQL databases
  - Migrating from MySQL to Amazon Aurora MySQL with reduced downtime
  - Avoiding slow performance, automatic restart, and failover for Aurora MySQL DB instances
- Recommendations for Aurora MySQL
  - Using multithreaded replication in Aurora MySQL version 3
  - Invoking AWS Lambda functions using native MySQL functions
  - Avoiding XA transactions with Amazon Aurora MySQL
  - Keeping foreign keys turned on during DML statements
  - Configuring how frequently the log buffer is flushed
  - Minimizing and troubleshooting Aurora MySQL deadlocks
    - Minimizing InnoDB deadlocks
    - Monitoring InnoDB deadlocks

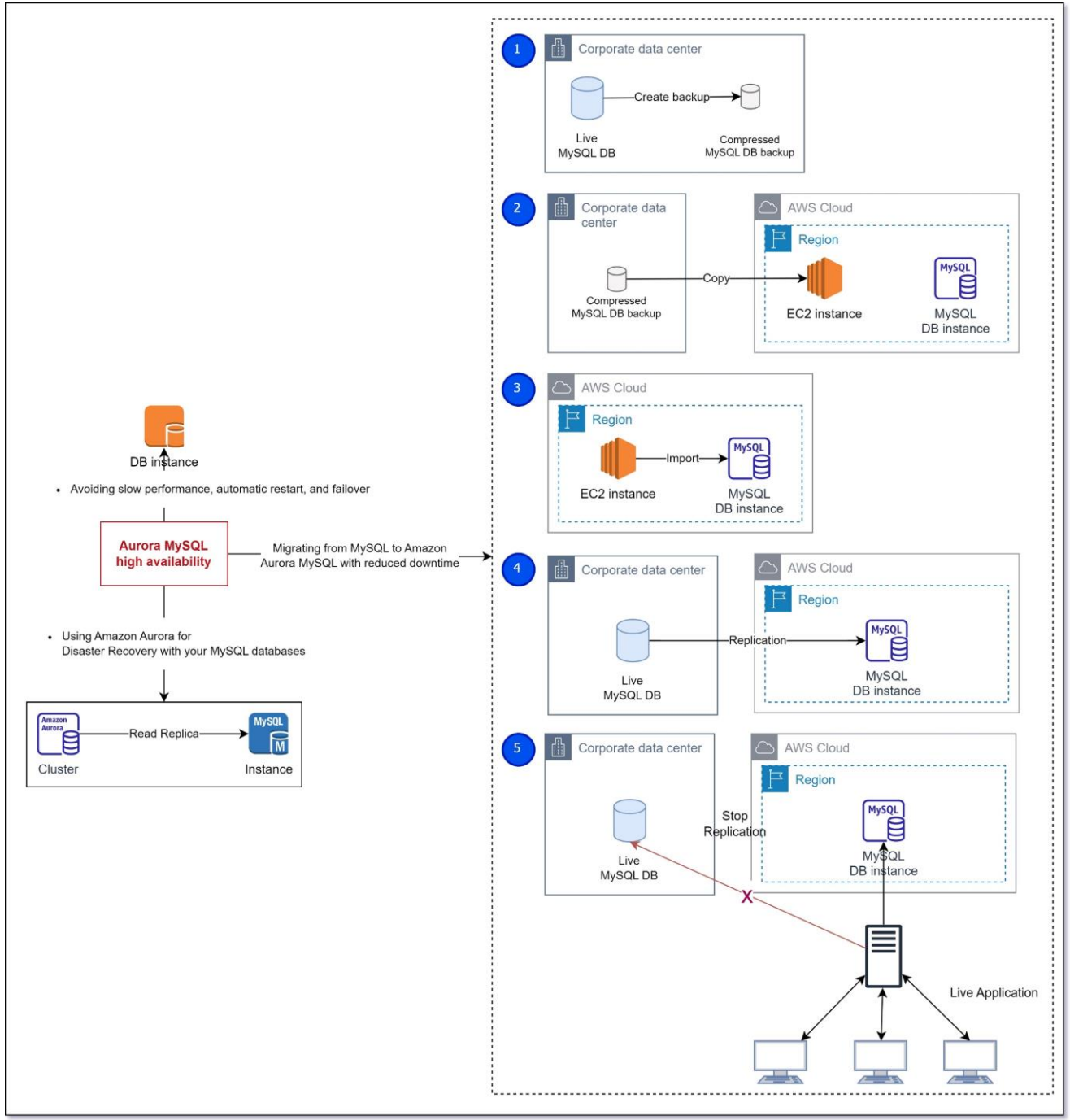
# Determining which DB instance

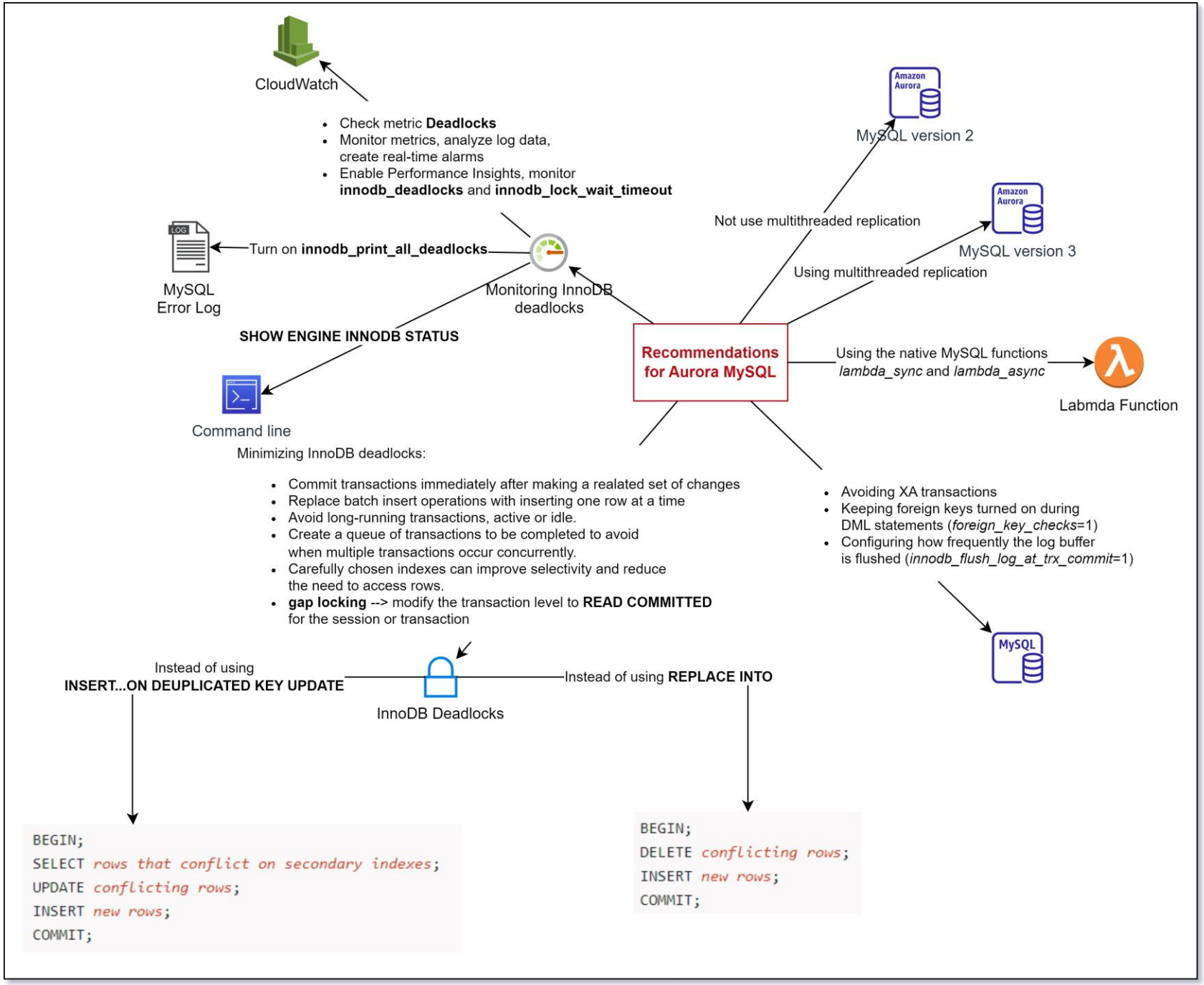
- To determine which DB instance in an Aurora MySQL DB cluster a connection is connected to, check the innodb_read_only global variable, as shown in the following example.

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

- The innodb_read_only variable is set to ON if you are connected to a reader DB instance. This setting is OFF if you are connected to a writer DB instance, such as a primary instance in a provisioned cluster.
- This approach can be helpful if you want to add logic to your application code to balance the workload or ensure that a write operation uses the correct connection.

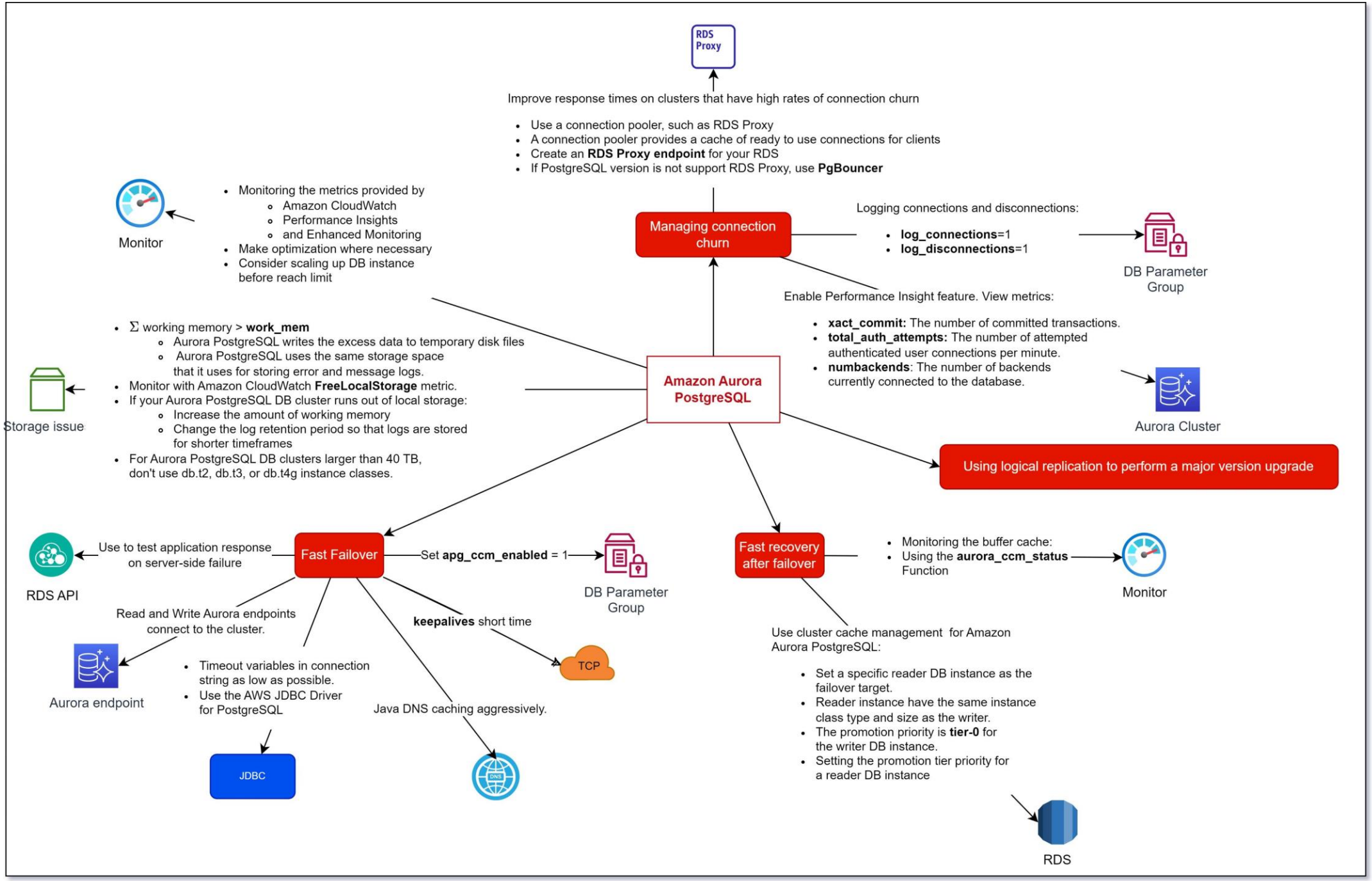

DB instance
Using T instance classes for development and testing, or other non-production servers
If your Aurora cluster is larger than 40 TB, don't use the T instance classes.
Don't enable the MySQL Performance Schema on Amazon Aurora MySQL T instances.
Using Amazon Aurora to scale reads for your MySQL database
Enabling hash joins
Optimizing queries for hash joins
Optimizing large Aurora MySQL join queries with hash joins
Aurora MySQL performance and scaling
Optimizing Aurora MySQL indexed join queries with asynchronous key prefetch
Enabling asynchronous key prefetch
Optimizing queries for asynchronous key prefetch
Optimizing large Aurora MySQL join queries with hash joins
Use the same DB instance class for all instances in your DB cluster.
Don't adjust any memory-related configuration settings.
Monitor your CPU Credit Balance (CPUCreditBalance) to ensure that it is at a sustainable level.
Monitor the replica lag (AuroraReplicaLag) between the writer and reader instances.
Keep the number of inserts per transaction below 1 million for DB clusters that have binary logging enabled.
T instances


DB instance
• Avoiding slow performance, automatic restart, and failover
Aurora MySQL high availability
Migrating from MySQL to Amazon Aurora MySQL with reduced downtime
• Using Amazon Aurora for Disaster Recovery with your MySQL databases
Amazon Aurora
Read Replica
MySQL
Cluster
Instance
1
Corporate data center
Create backup
Live MySQL DB
Compressed MySQL DB backup
2
Corporate data center
Compressed MySQL DB backup
Copy
AWS Cloud
Region
EC2 instance
MySQL
MySQL DB instance
3
AWS Cloud
Region
Import
EC2 instance
MySQL
MySQL DB instance
4
Corporate data center
Live MySQL DB
Replication
AWS Cloud
Region
MySQL
MySQL DB instance
5
Corporate data center
Live MySQL DB
Stop Replication
AWS Cloud
Region
MySQL
MySQL DB instance
X
Live Application


CloudWatch
Check metric Deadlocks
Monitor metrics, analyze log data, create real-time alarms
Enable Performance Insights, monitor innodb_deadlocks and innodb_lock_wait_timeout
Amazon Aurora
MySQL version 2
Not use multithreaded replication
Amazon Aurora
MySQL version 3
Using multithreaded replication
Turn on innodb_print_all_deadlocks
MySQL Error Log
Monitoring InnoDB deadlocks
SHOW ENGINE INNODB STATUS
Command line
Recommendations for Aurora MySQL
Using the native MySQL functions lambda_sync and lambda_async
Labmda Function
Minimizing InnoDB deadlocks:
Commit transactions immediately after making a realated set of changes
Replace batch insert operations with inserting one row at a time
Avoid long-running transactions, active or idle.
Create a queue of transactions to be completed to avoid when multiple transactions occur concurrently.
Carefully chosen indexes can improve selectivity and reduce the need to access rows.
gap locking --> modify the transaction level to READ COMMITTED for the session or transaction
Avoiding XA transactions
Keeping foreign keys turned on during DML statements (foreign_key_checks=1)
Configuring how frequently the log buffer is flushed (innodb_flush_log_at_trx_commit=1)
MySQL
Instead of using INSERT...ON DEUPLICATED KEY UPDATE
InnoDB Deadlocks
Instead of using REPLACE INTO
BEGIN;
SELECT rows that conflict on secondary indexes;
UPDATE conflicting rows;
INSERT new rows;
COMMIT;
BEGIN;
DELETE conflicting rows;
INSERT new rows;
COMMIT;

# Best practices with Amazon Aurora PostgreSQL

- Troubleshooting storage issues
- Avoiding slow performance, automatic restart, and failover for Aurora PostgreSQL DB instances
- Fast failover with Amazon Aurora PostgreSQL
- Fast recovery after failover with cluster cache management for Aurora PostgreSQL
- Managing Aurora PostgreSQL connection churn with pooling
- Using logical replication to perform a major version upgrade for Aurora PostgreSQL
- Tuning memory parameters for Aurora PostgreSQL
- Using Amazon CloudWatch metrics to analyze resource usage for Aurora PostgreSQL
- Diagnosing table and index bloat

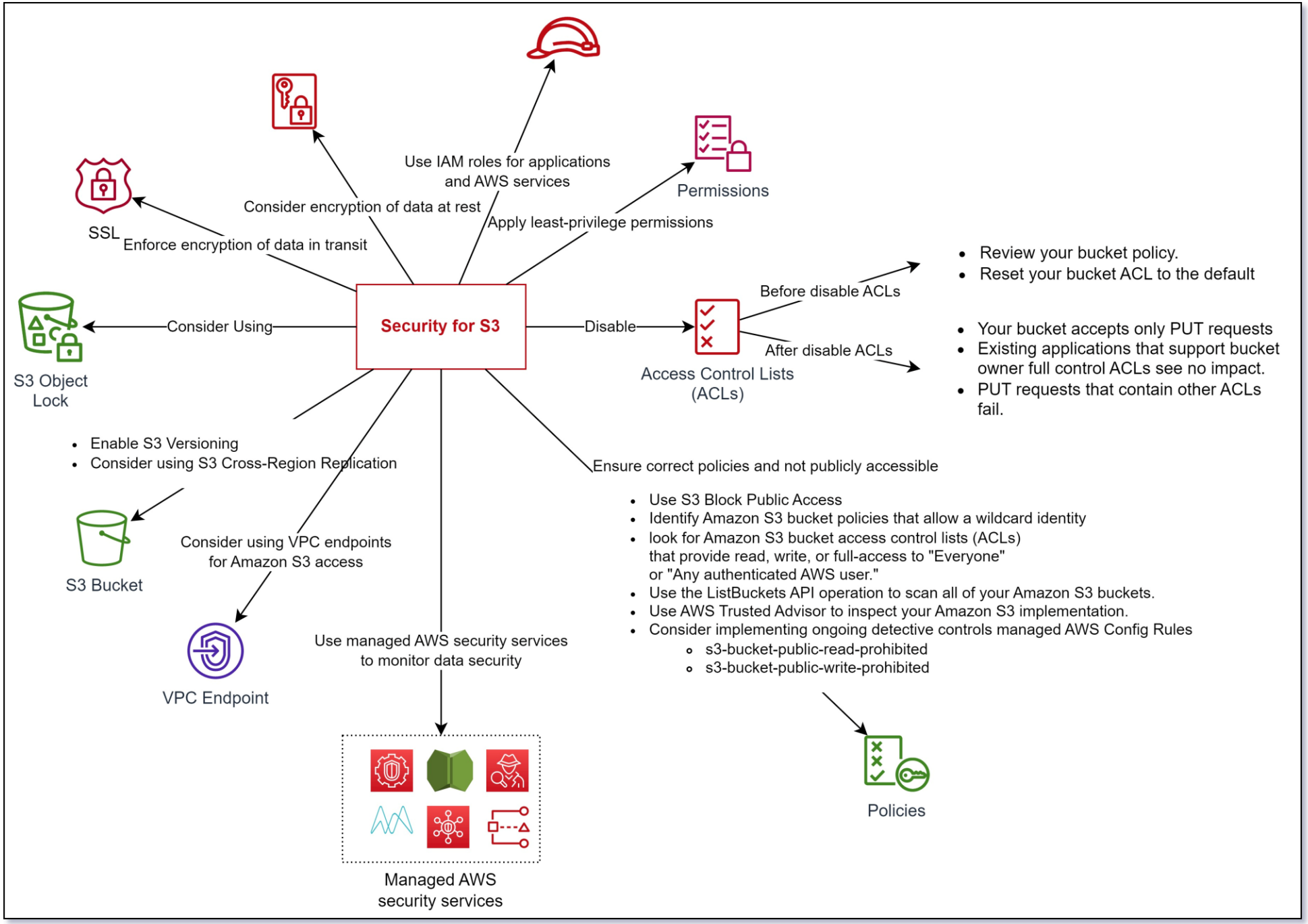

RDS Proxy
Improve response times on clusters that have high rates of connection churn
Use a connection pooler, such as RDS Proxy
A connection pooler provides a cache of ready to use connections for clients
Create an RDS Proxy endpoint for your RDS
If PostgreSQL version is not support RDS Proxy, use PgBouncer
Monitor
Monitoring the metrics provided by
Amazon CloudWatch
Performance Insights
and Enhanced Monitoring
Make optimization where necessary
Consider scaling up DB instance before reach limit
Managing connection churn
Logging connections and disconnections:
log_connections=1
log_disconnections=1
DB Parameter Group
Enable Performance Insight feature. View metrics:
xact_commit: The number of committed transactions.
total_auth_attempts: The number of attempted authenticated user connections per minute.
numbackends: The number of backends currently connected to the database.
Aurora Cluster
Storage issue
Σ working memory > work_mem
Aurora PostgreSQL writes the excess data to temporary disk files
Aurora PostgreSQL uses the same storage space that it uses for storing error and message logs.
Monitor with Amazon CloudWatch FreeLocalStorage metric.
If your Aurora PostgreSQL DB cluster runs out of local storage:
Increase the amount of working memory
Change the log retention period so that logs are stored for shorter timeframes
For Aurora PostgreSQL DB clusters larger than 40 TB, don't use db.t2, db.t3, or db.t4g instance classes.
Amazon Aurora PostgreSQL
Using logical replication to perform a major version upgrade
RDS API
Use to test application response on server-side failure
Fast Failover
Set apg_ccm_enabled = 1
DB Parameter Group
Read and Write Aurora endpoints connect to the cluster.
keepalives short time
TCP
Aurora endpoint
Timeout variables in connection string as low as possible.
Use the AWS JDBC Driver for PostgreSQL
JDBC
Java DNS caching aggressively.
DNS
Fast recovery after failover
Monitoring the buffer cache:
Using the aurora_ccm_status Function
Monitor
Use cluster cache management for Amazon Aurora PostgreSQL:
Set a specific reader DB instance as the failover target.
Reader instance have the same instance class type and size as the writer.
The promotion priority is tier-0 for the writer DB instance.
Setting the promotion tier priority for a reader DB instance
RDS

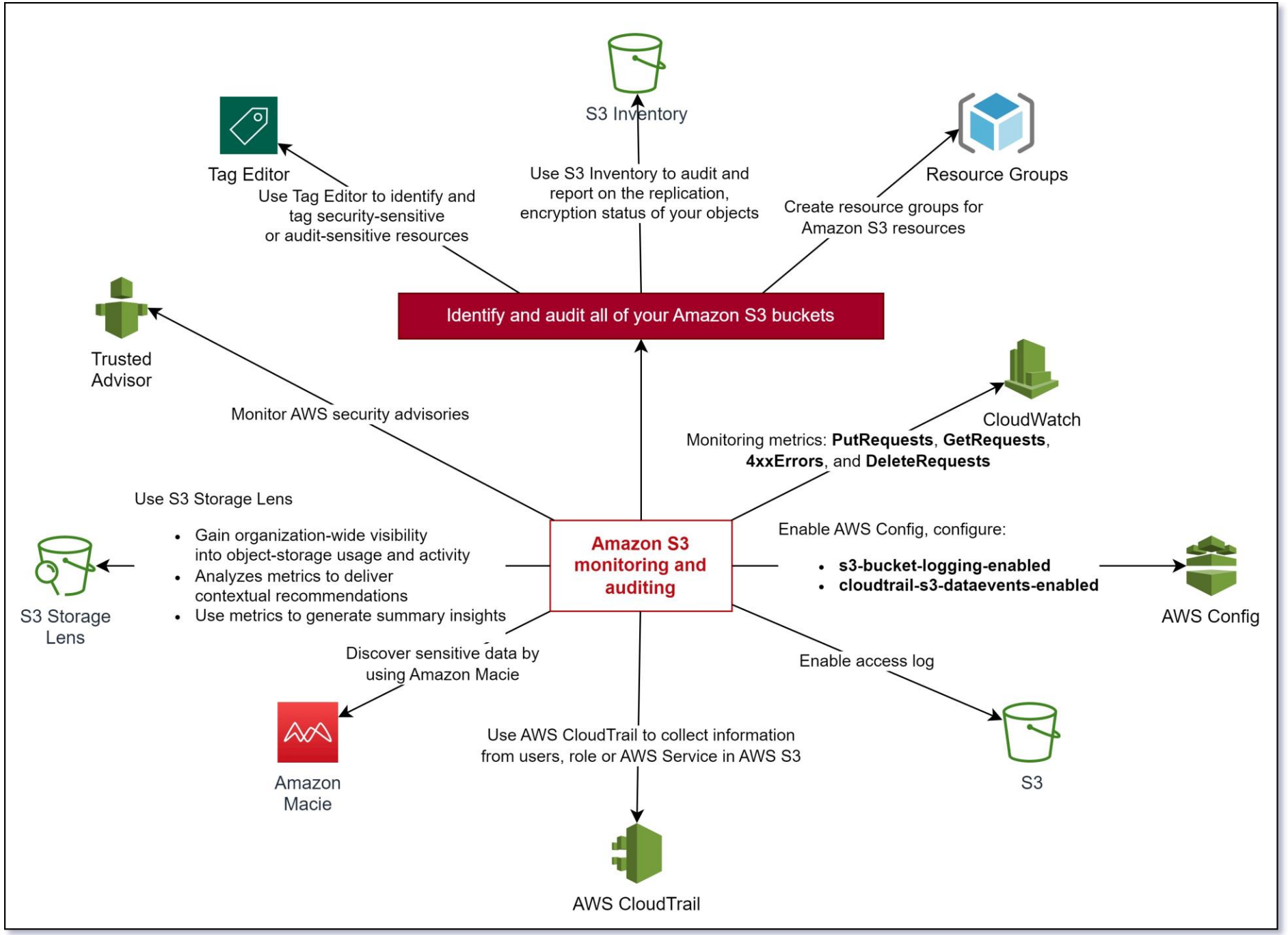
# Amazon S3 (Simple Storage Service)

Nash Tech.

# Security best practice for Amazon S3

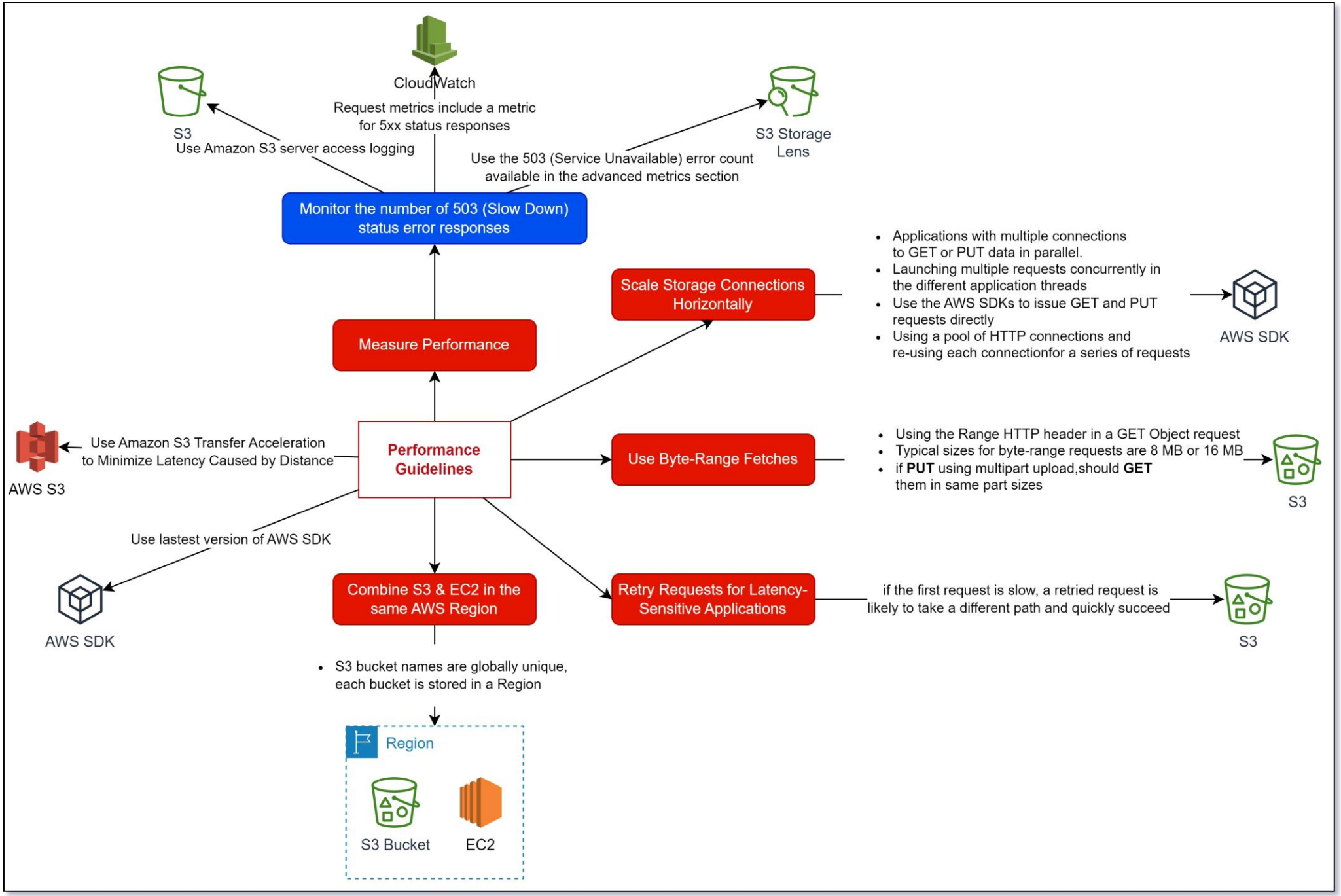
- Amazon S3 security best practices
- Amazon S3 monitoring and auditing best practices

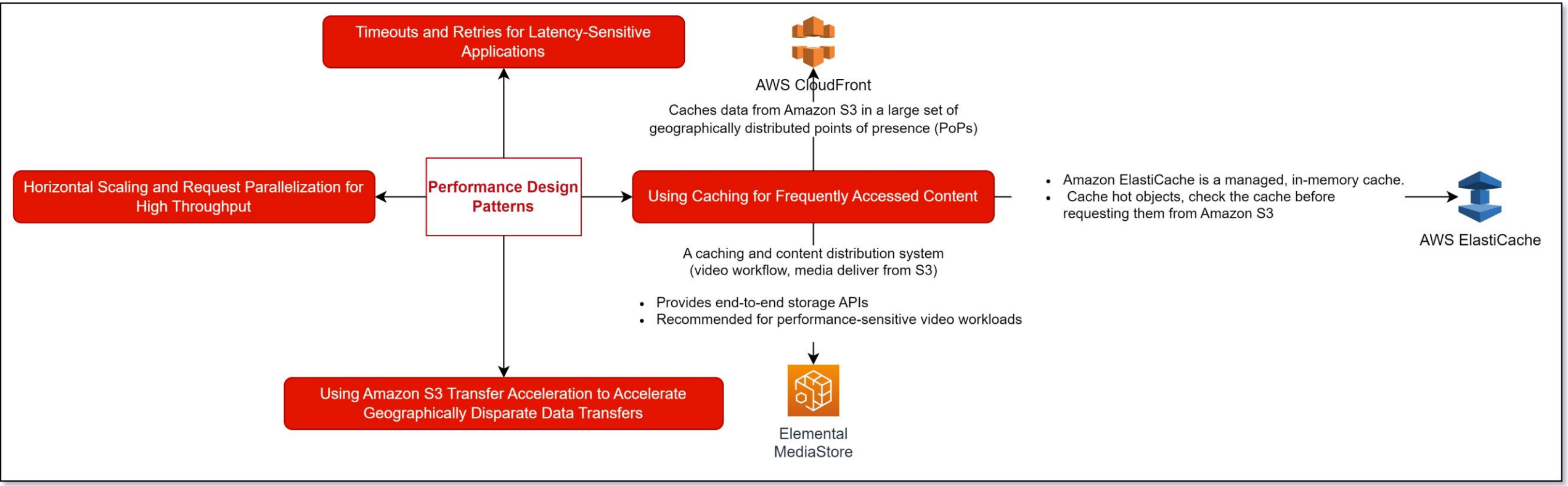

Security for S3
Consider encryption of data at rest
Use IAM roles for applications and AWS services
Apply least-privilege permissions
Permissions
SSL
Enforce encryption of data in transit
Consider Using
S3 Object Lock
Disable
Access Control Lists (ACLs)
Before disable ACLs
After disable ACLs
Review your bucket policy.
Reset your bucket ACL to the default
Your bucket accepts only PUT requests
Existing applications that support bucket owner full control ACLs see no impact.
PUT requests that contain other ACLs fail.
Enable S3 Versioning
Consider using S3 Cross-Region Replication
S3 Bucket
Consider using VPC endpoints for Amazon S3 access
VPC Endpoint
Use managed AWS security services to monitor data security
Managed AWS security services
Ensure correct policies and not publicly accessible
Use S3 Block Public Access
Identify Amazon S3 bucket policies that allow a wildcard identity
look for Amazon S3 bucket access control lists (ACLs) that provide read, write, or full-access to "Everyone" or "Any authenticated AWS user."
Use the ListBuckets API operation to scan all of your Amazon S3 buckets.
Use AWS Trusted Advisor to inspect your Amazon S3 implementation.
Consider implementing ongoing detective controls managed AWS Config Rules
s3-bucket-public-read-prohibited
s3-bucket-public-write-prohibited
Policies


S3 Inventory
Tag Editor
Use Tag Editor to identify and tag security-sensitive or audit-sensitive resources
Use S3 Inventory to audit and report on the replication, encryption status of your objects
Resource Groups
Create resource groups for Amazon S3 resources
Identify and audit all of your Amazon S3 buckets
Trusted Advisor
Monitor AWS security advisories
CloudWatch
Monitoring metrics: PutRequests, GetRequests, 4xxErrors, and DeleteRequests
Use S3 Storage Lens
Gain organization-wide visibility into object-storage usage and activity
Analyzes metrics to deliver contextual recommendations
Use metrics to generate summary insights
S3 Storage Lens
Amazon S3 monitoring and auditing
Enable AWS Config, configure:
s3-bucket-logging-enabled
cloudtrail-s3-dataevents-enabled
AWS Config
Discover sensitive data by using Amazon Macie
Amazon Macie
Use AWS CloudTrail to collect information from users, role or AWS Service in AWS S3
AWS CloudTrail
Enable access log
S3

# Optimizing Amazon S3 performance

- Performance Guidelines
- Performance Design Patterns

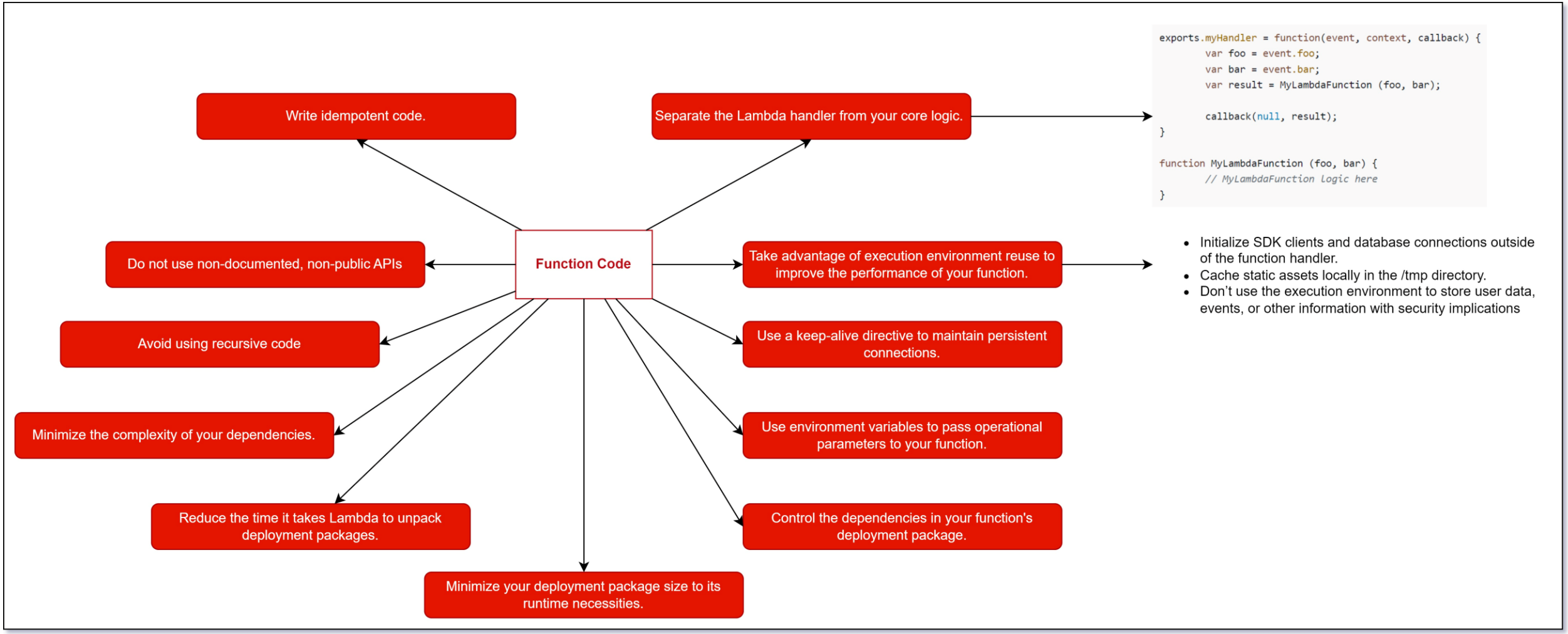

CloudWatch
Request metrics include a metric for 5xx status responses
S3
Use Amazon S3 server access logging
S3 Storage Lens
Use the 503 (Service Unavailable) error count available in the advanced metrics section
Monitor the number of 503 (Slow Down) status error responses
Measure Performance
Performance Guidelines
Scale Storage Connections Horizontally
Applications with multiple connections to GET or PUT data in parallel.
Launching multiple requests concurrently in the different application threads
Use the AWS SDKs to issue GET and PUT requests directly
Using a pool of HTTP connections and re-using each connectionfor a series of requests
AWS SDK
Use Byte-Range Fetches
Using the Range HTTP header in a GET Object request
Typical sizes for byte-range requests are 8 MB or 16 MB
if PUT using multipart upload,should GET them in same part sizes
S3
Use Amazon S3 Transfer Acceleration to Minimize Latency Caused by Distance
AWS S3
Use lastest version of AWS SDK
AWS SDK
Combine S3 & EC2 in the same AWS Region
S3 bucket names are globally unique, each bucket is stored in a Region
Region
S3 Bucket
EC2
Retry Requests for Latency-Sensitive Applications
if the first request is slow, a retried request is likely to take a different path and quickly succeed
S3

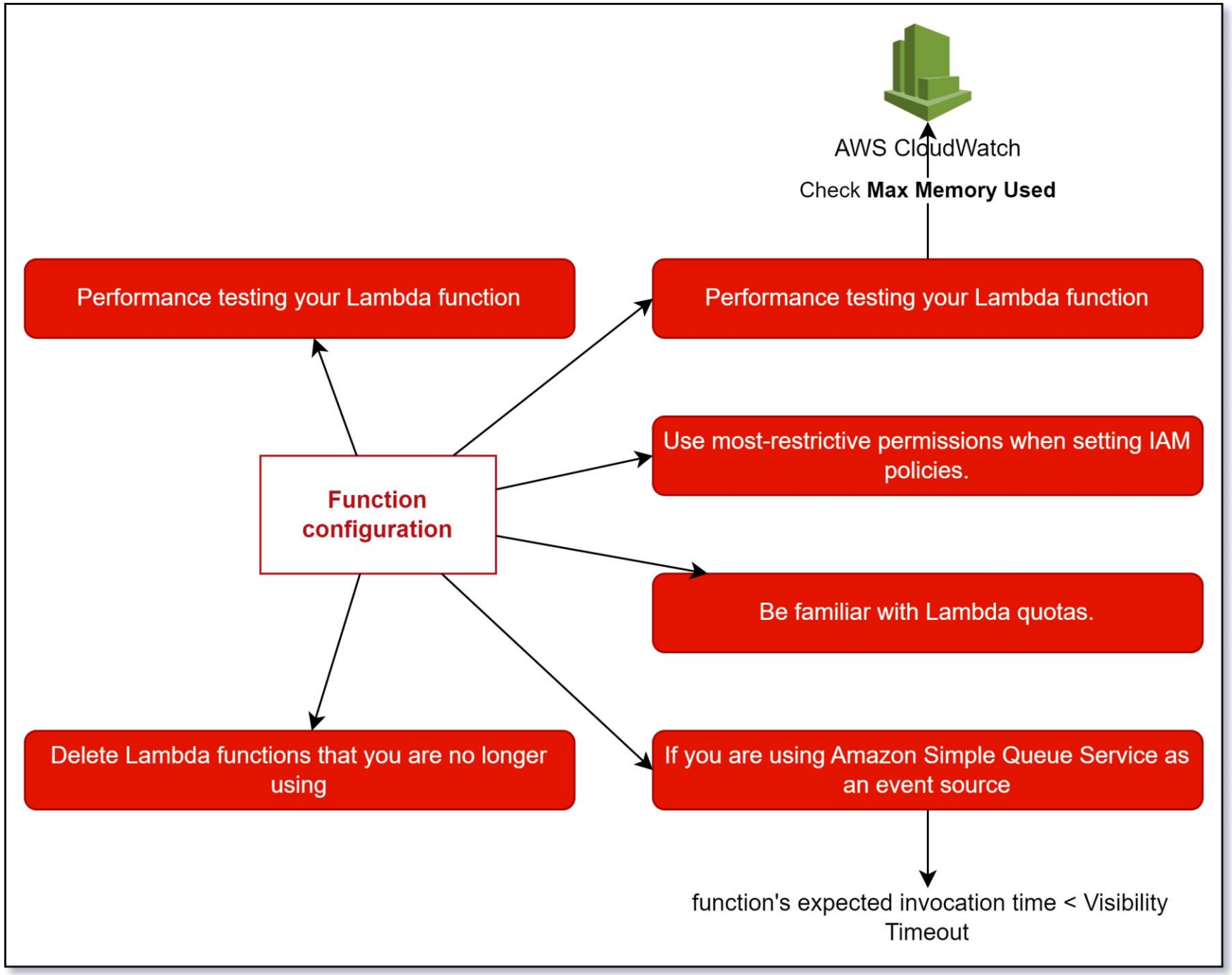

Timeouts and Retries for Latency-Sensitive Applications
AWS CloudFront
Caches data from Amazon S3 in a large set of geographically distributed points of presence (PoPs)
Horizontal Scaling and Request Parallelization for High Throughput
Performance Design Patterns
Using Caching for Frequently Accessed Content
Amazon ElastiCache is a managed, in-memory cache.
Cache hot objects, check the cache before requesting them from Amazon S3
AWS ElastiCache
A caching and content distribution system (video workflow, media deliver from S3)
Provides end-to-end storage APIs
Recommended for performance-sensitive video workloads
Using Amazon S3 Transfer Acceleration to Accelerate Geographically Disparate Data Transfers
Elemental MediaStore

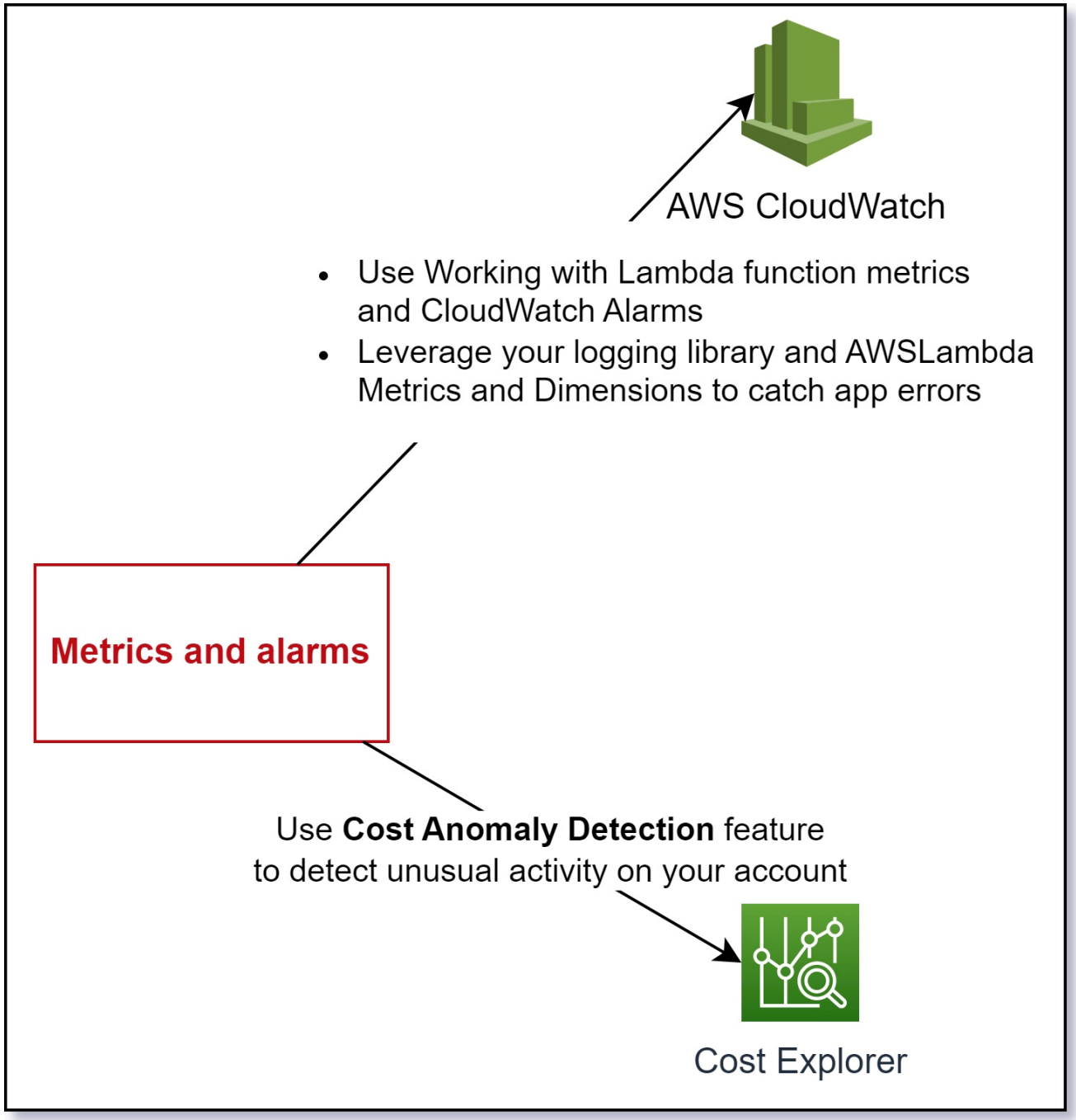

Amazon Lambda
Nash
Tech.

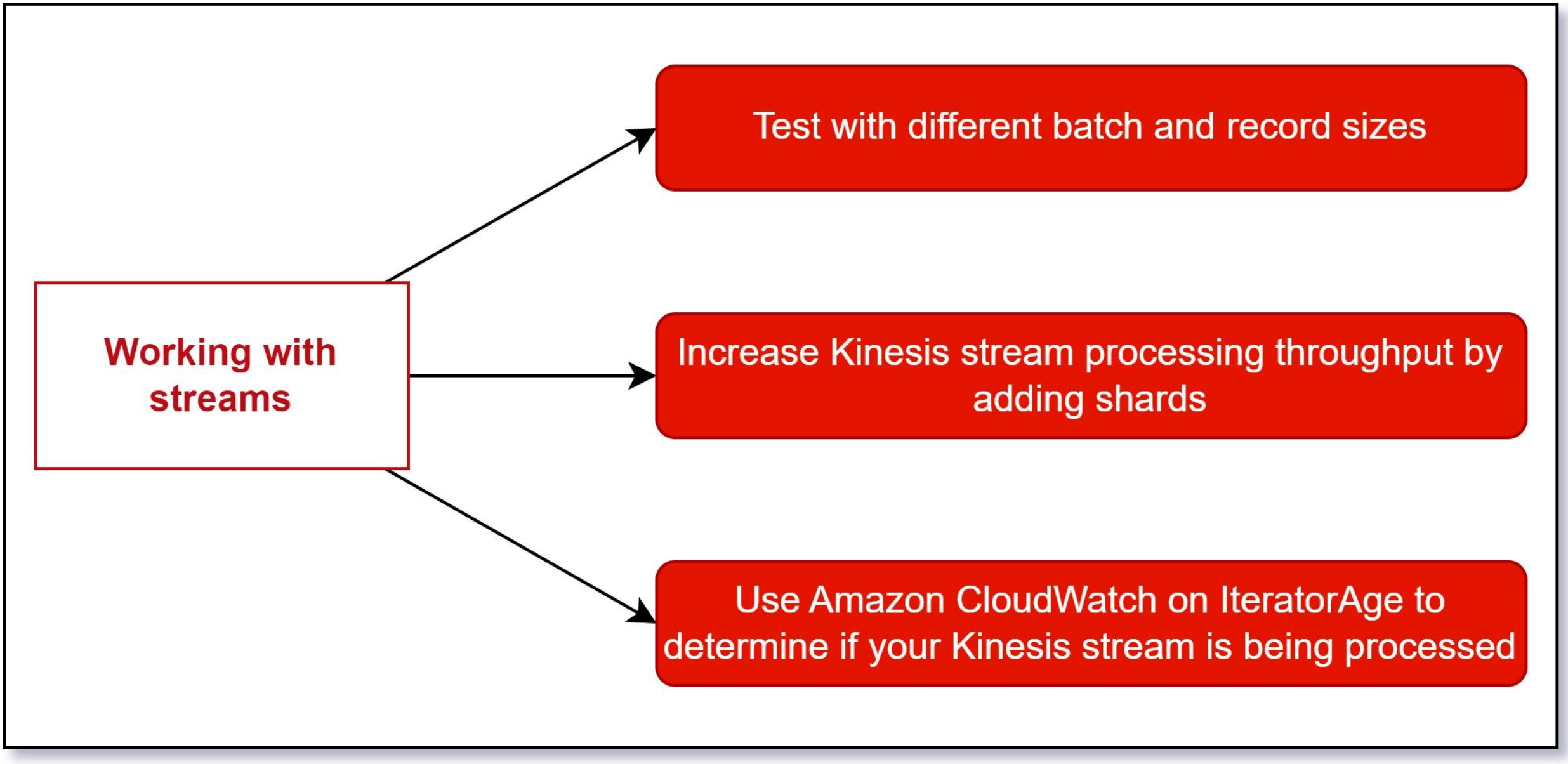
# Best practices for working with AWS Lambda functions

- Function code
- Function configuration
- Metrics and alarms
- Working with streams
- Security best practices


Write idempotent code.
Separate the Lambda handler from your core logic.
exports.myHandler = function(event, context, callback) {
var foo = event.foo;
var bar = event.bar;
var result = MyLambdaFunction (foo, bar);
callback(null, result);
}
function MyLambdaFunction (foo, bar) {
// MyLambdaFunction logic here
}
Do not use non-documented, non-public APIs
Function Code
Take advantage of execution environment reuse to improve the performance of your function.
Initialize SDK clients and database connections outside of the function handler.
Cache static assets locally in the /tmp directory.
Don't use the execution environment to store user data, events, or other information with security implications
Avoid using recursive code
Use a keep-alive directive to maintain persistent connections.
Minimize the complexity of your dependencies.
Use environment variables to pass operational parameters to your function.
Reduce the time it takes Lambda to unpack deployment packages.
Control the dependencies in your function's deployment package.
Minimize your deployment package size to its runtime necessities.


AWS CloudWatch
Check **Max Memory Used**
Performance testing your Lambda function
Performance testing your Lambda function
Function configuration
Use most-restrictive permissions when setting IAM policies.
Be familiar with Lambda quotas.
Delete Lambda functions that you are no longer using
If you are using Amazon Simple Queue Service as an event source
function's expected invocation time < Visibility Timeout


AWS CloudWatch
Use Working with Lambda function metrics and CloudWatch Alarms
Leverage your logging library and AWSLambda Metrics and Dimensions to catch app errors
Metrics and alarms
Use Cost Anomaly Detection feature to detect unusual activity on your account
Cost Explorer


Working with streams
Test with different batch and record sizes
Increase Kinesis stream processing throughput by adding shards
Use Amazon CloudWatch on IteratorAge to determine if your Kinesis stream is being processed

**Security**

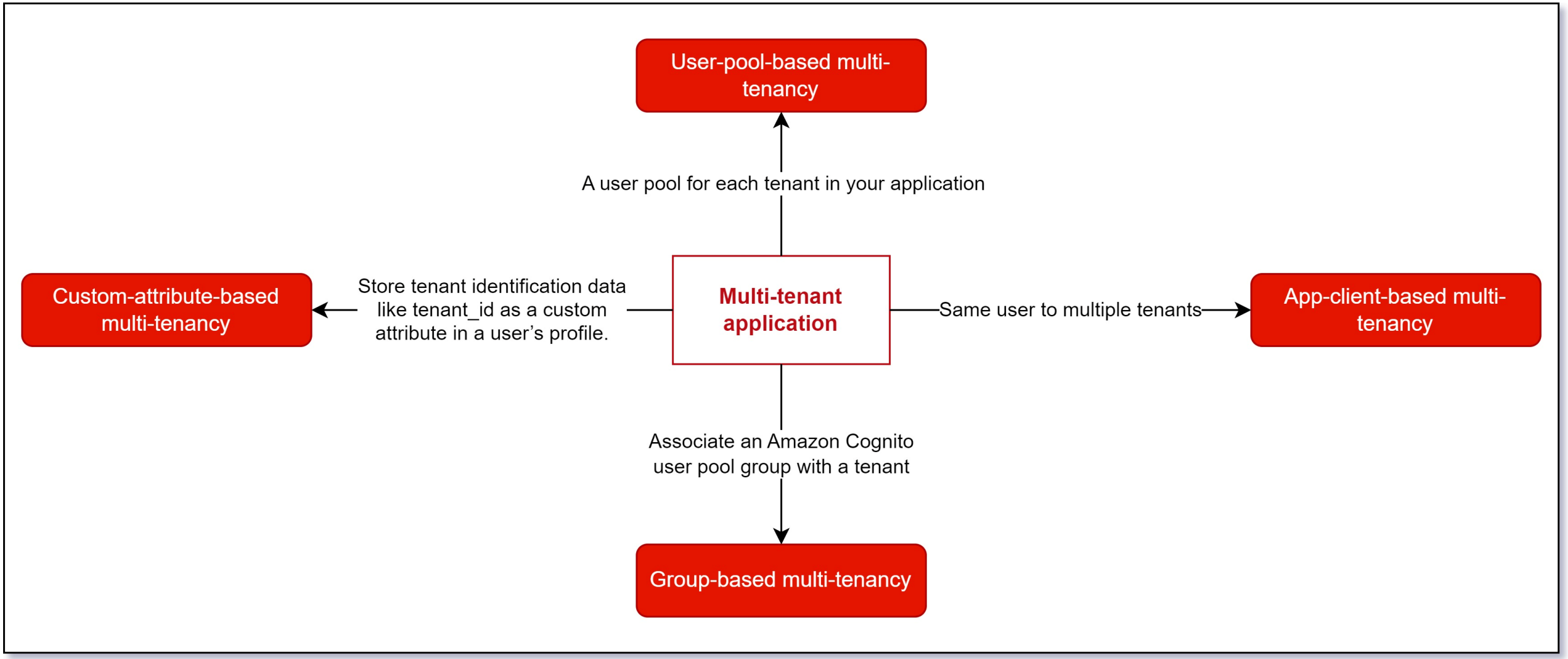
Monitor your usage of AWS Lambda as it relates to security best practices

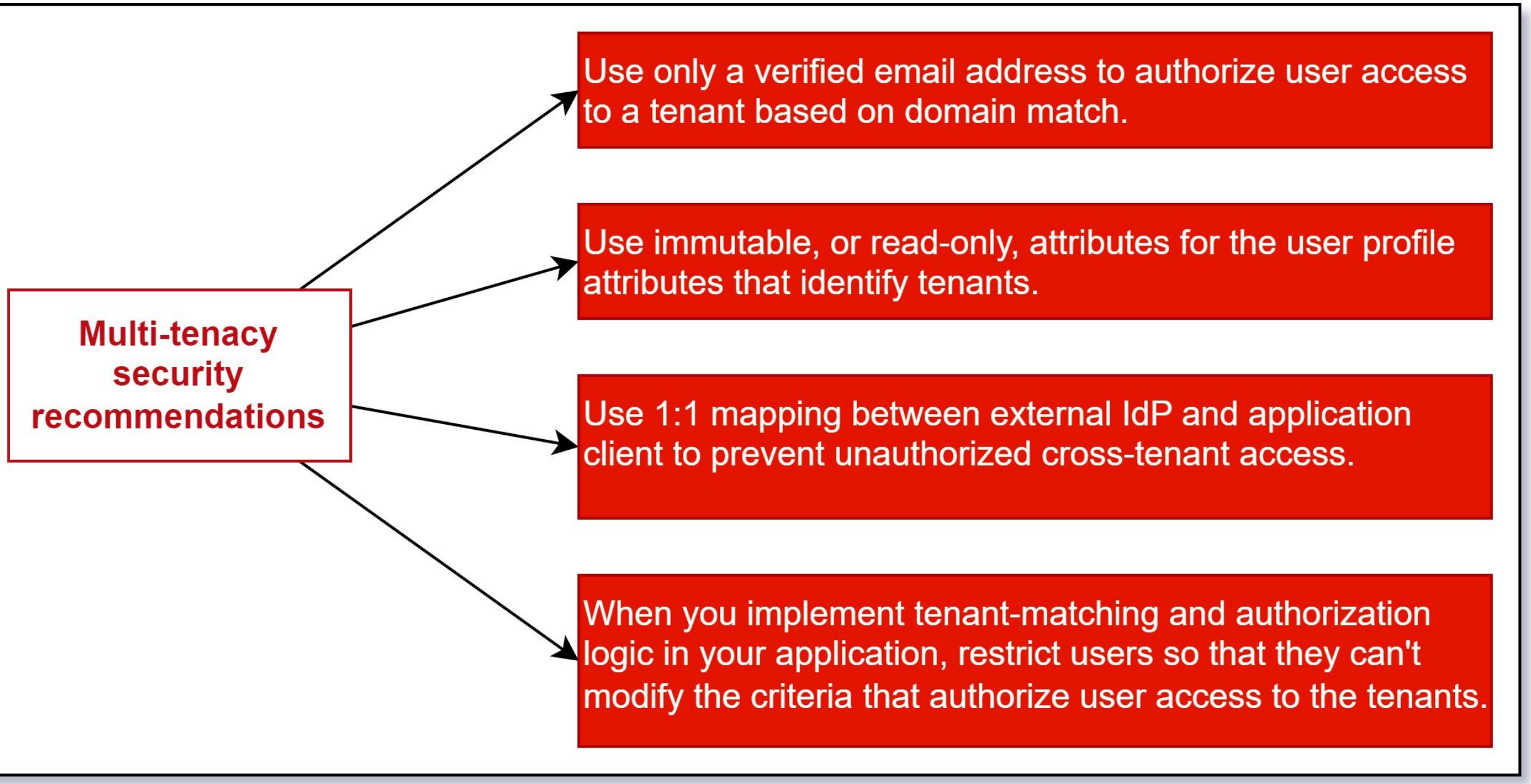
# Amazon Cognito

Nash Tech.

# Multi-tenant application best practice

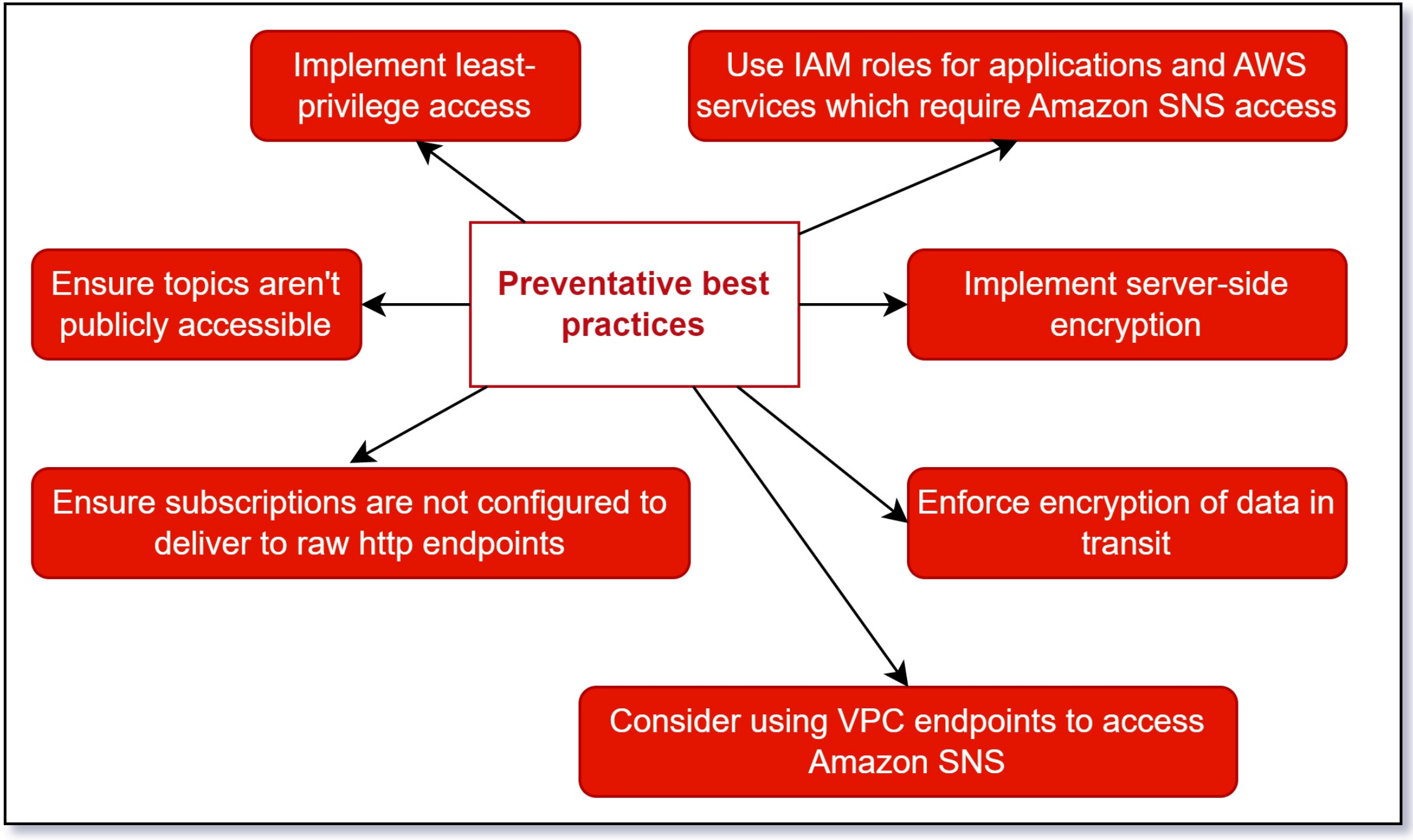
- User-pool-based multi-tenancy
- App-client-based multi-tenancy
- Group-based multi-tenancy
- Custom-attribute-based multi-tenancy
- Multi-tenancy security recommendations


User-pool-based multi-tenancy
A user pool for each tenant in your application
Custom-attribute-based multi-tenancy
Store tenant identification data like tenant_id as a custom attribute in a user's profile.
Multi-tenant application
Same user to multiple tenants
App-client-based multi-tenancy
Associate an Amazon Cognito user pool group with a tenant
Group-based multi-tenancy


Multi-tenacy security recommendations
Use only a verified email address to authorize user access to a tenant based on domain match.
Use immutable, or read-only, attributes for the user profile attributes that identify tenants.
Use 1:1 mapping between external IdP and application client to prevent unauthorized cross-tenant access.
When you implement tenant-matching and authorization logic in your application, restrict users so that they can't modify the criteria that authorize user access to the tenants.

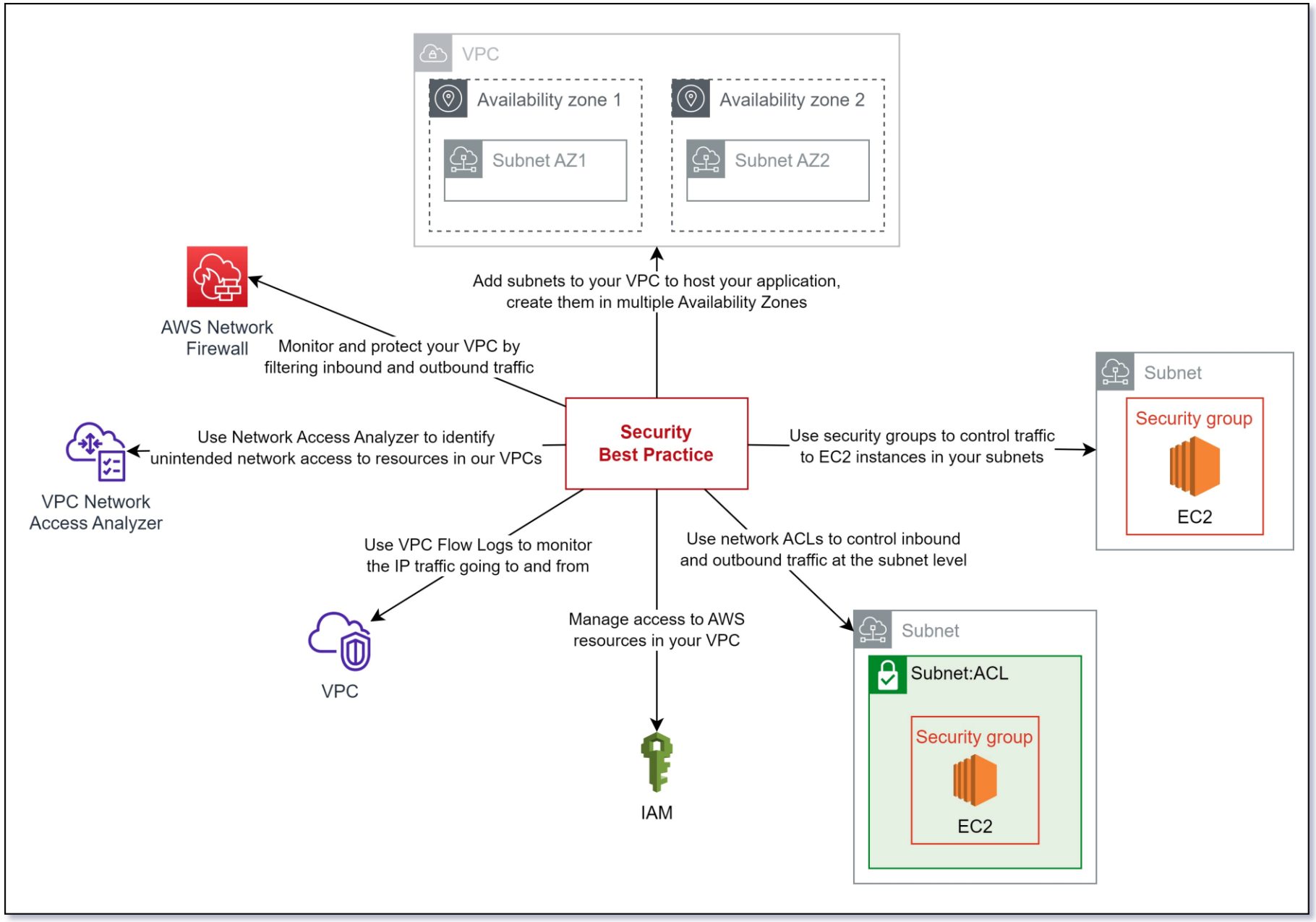
# Amazon SNS (Simple Notification Service)

Nash Tech.


Implement least-privilege access
Use IAM roles for applications and AWS services which require Amazon SNS access
Ensure topics aren't publicly accessible
Preventative best practices
Implement server-side encryption
Ensure subscriptions are not configured to deliver to raw http endpoints
Enforce encryption of data in transit
Consider using VPC endpoints to access Amazon SNS

# Amazon VPC (Virtual Private Cloud)

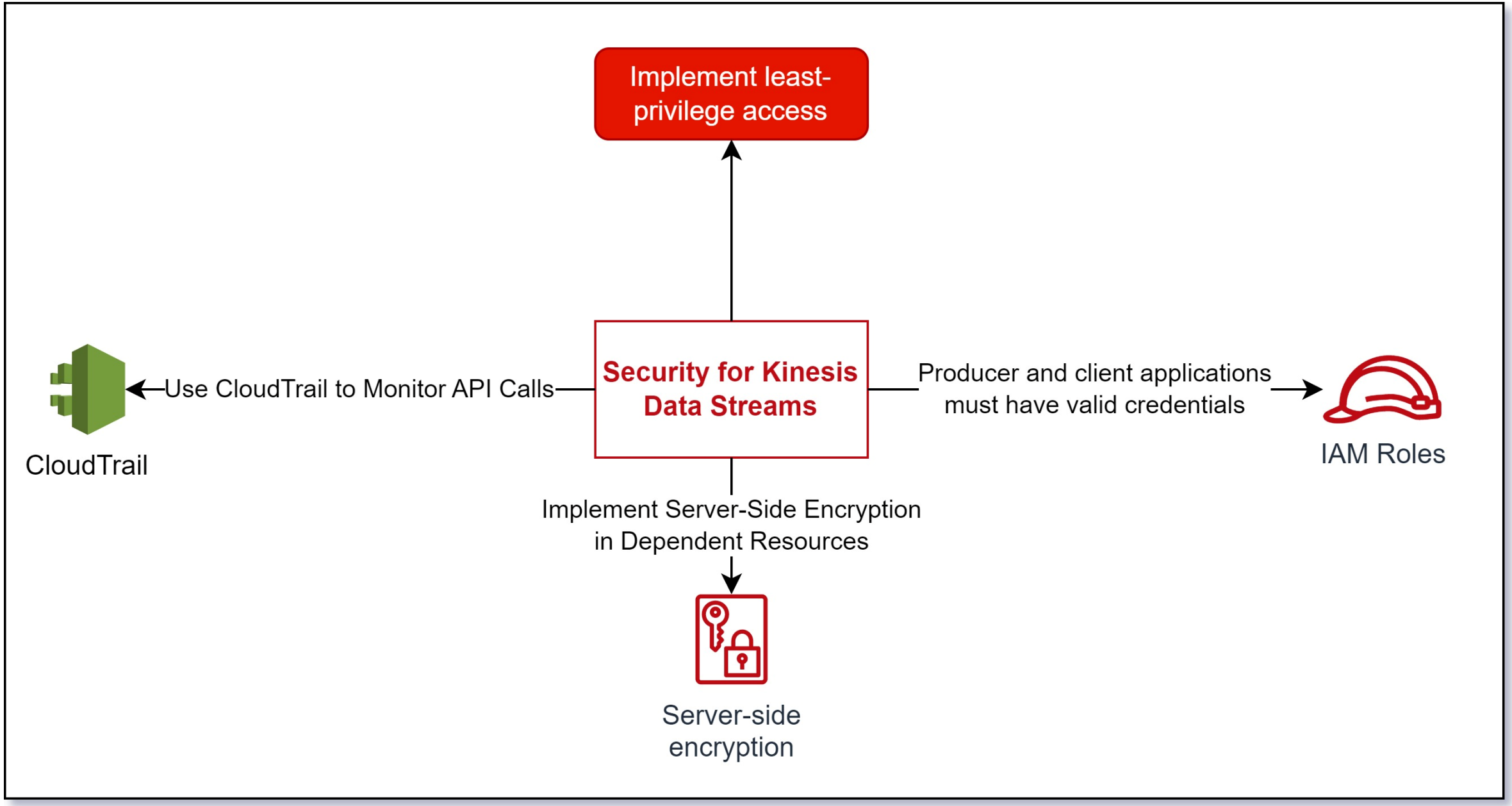
Nash Tech.



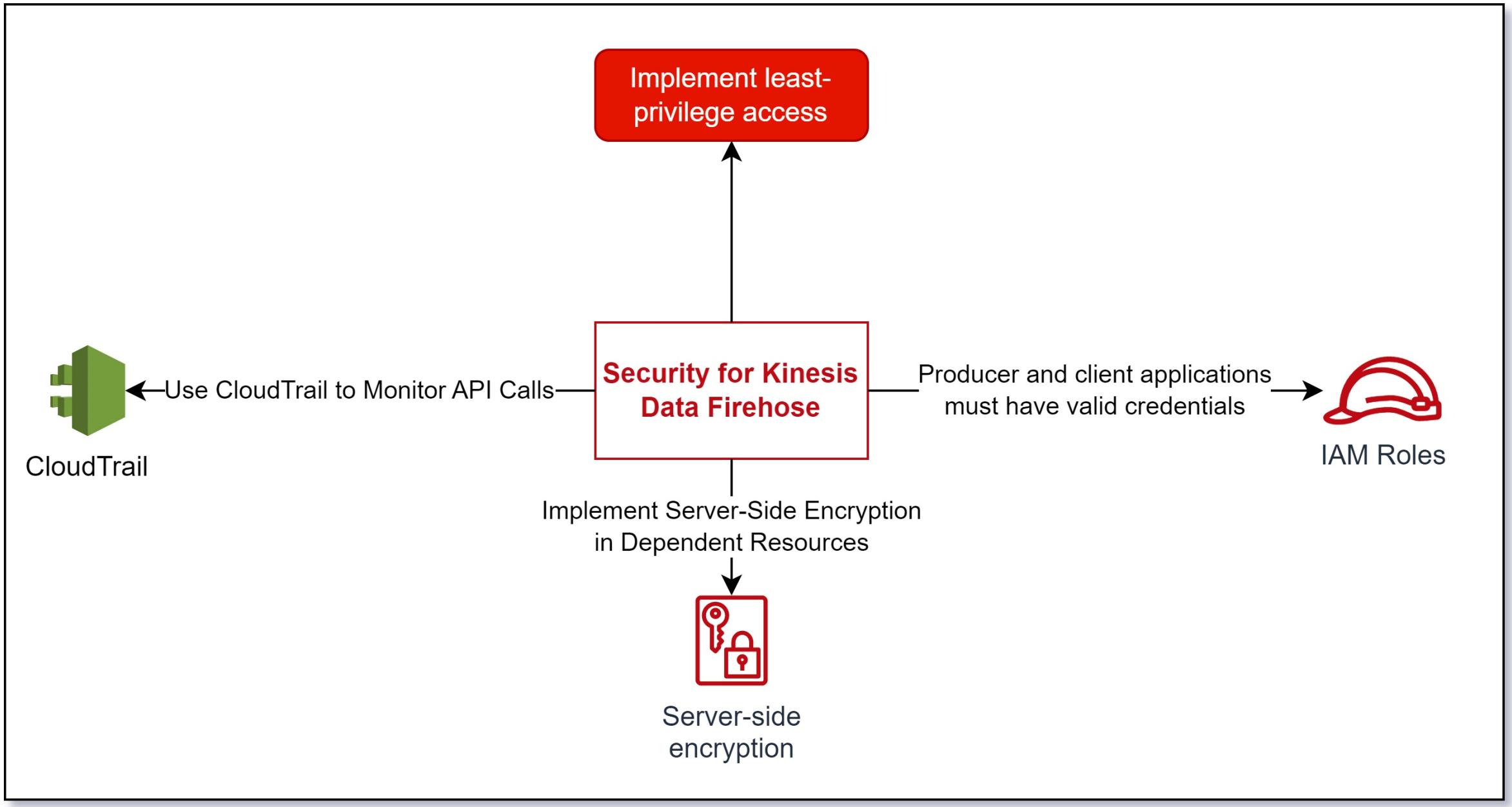

[Security best practices for your VPC - Amazon Virtual Private Cloud]

# Amazon Kinesis

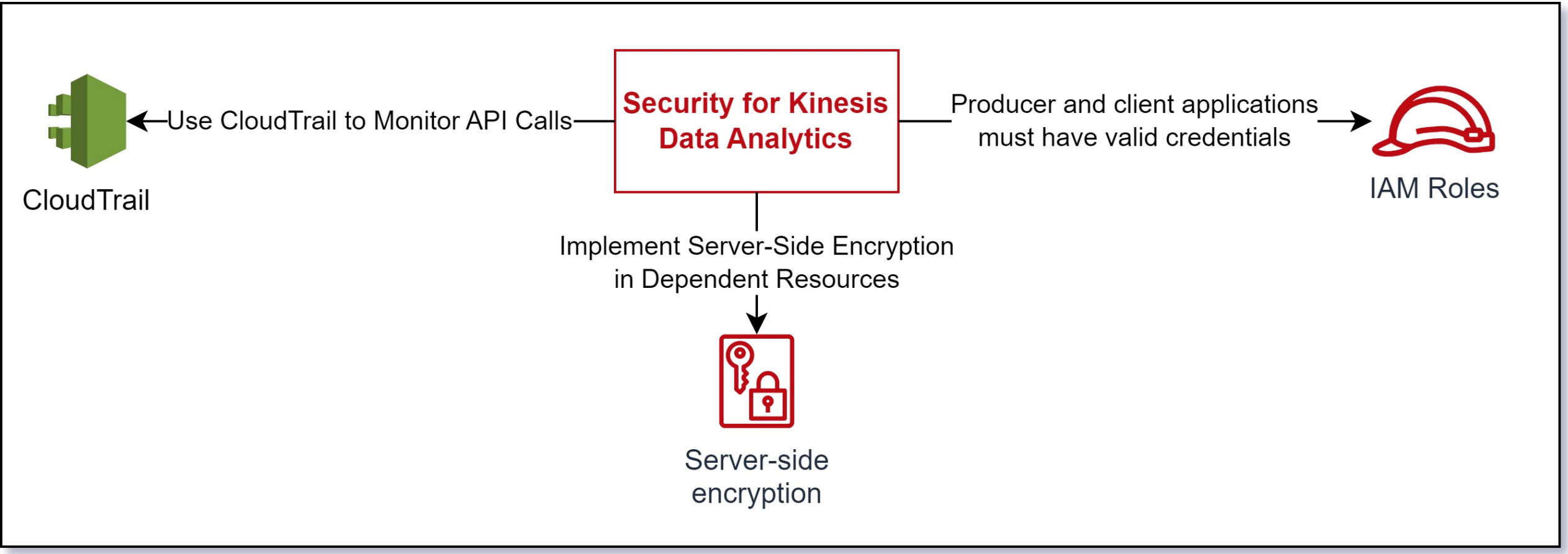
Nash Tech.




[Security Best Practices for Kinesis Data Streams - Amazon Kinesis Data Streams]



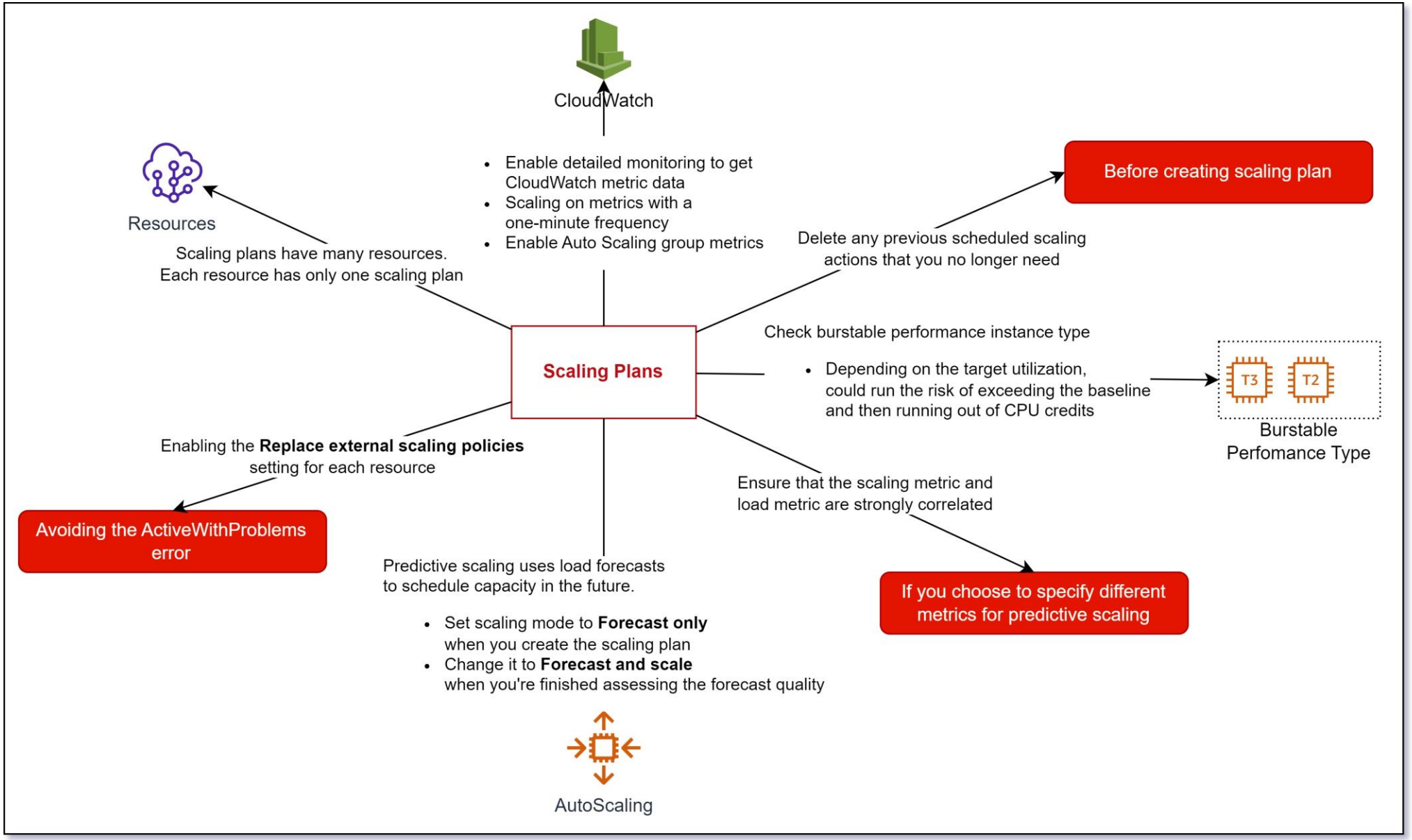

[Security Best Practices for Kinesis Data Firehose - Amazon Kinesis Data Firehose]




[Security Best Practices for Kinesis Data Analytics - Amazon Kinesis Data Analytics for SQL Applications Developer Guide]

# Amazon Auto-scaling

Nash Tech.



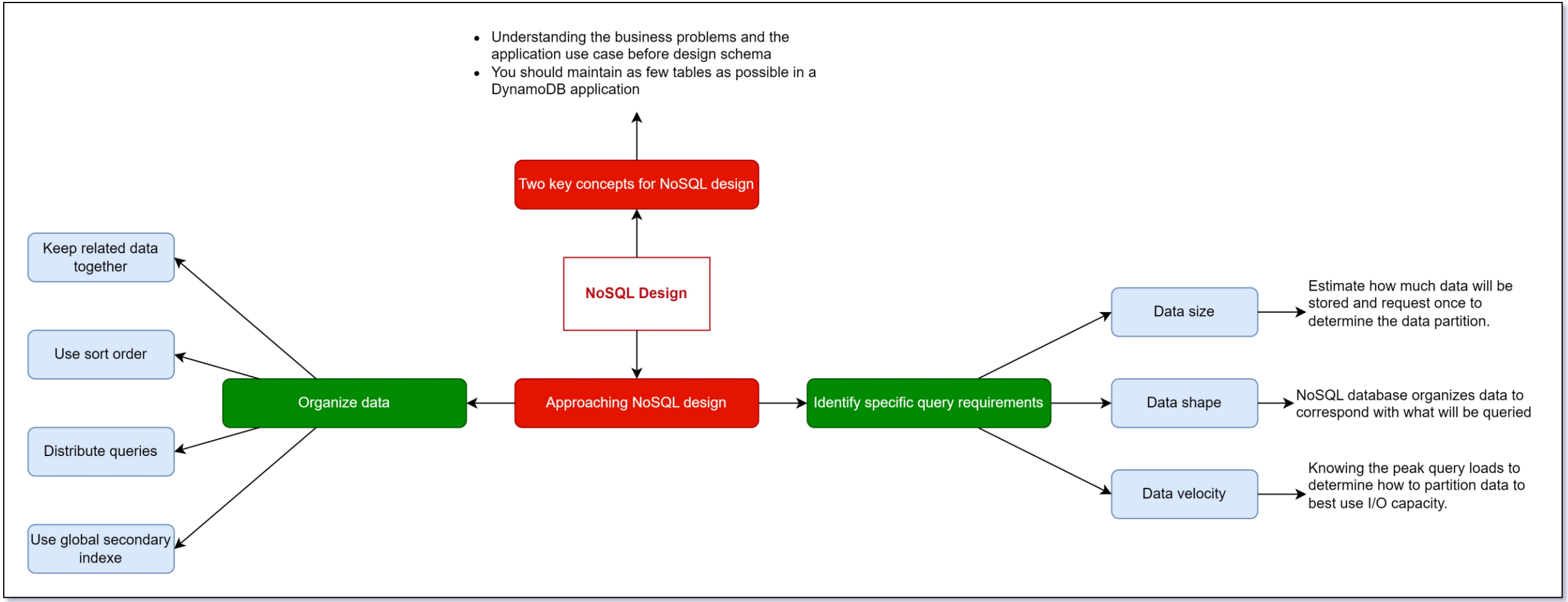

https://docs.aws.amazon.com/autoscaling/plans/userguide/best-practices-for-scaling-plans.html

# Dynamo DB

Nash Tech.

# DynamoDB best practices

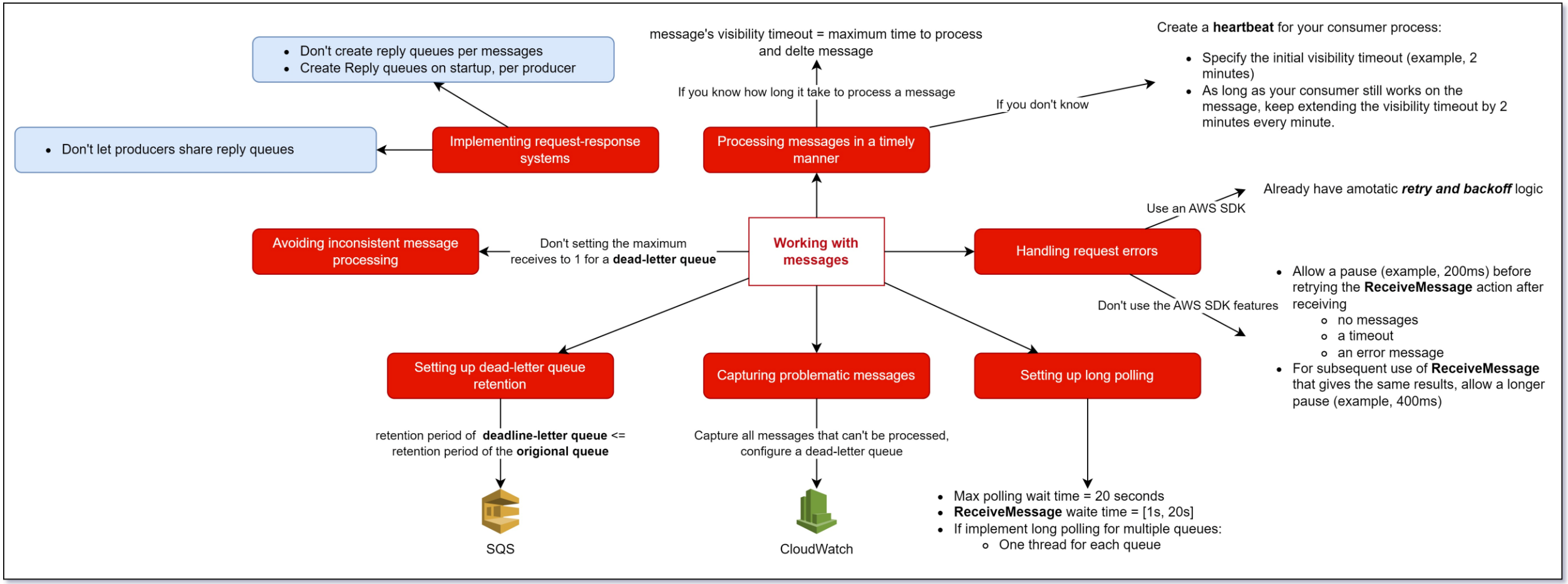
- NoSQL design for DynamoDB
- Using deletion protection to protect your table
- Using the DynamoDB Well-Architected Lens to optimize your DynamoDB workload
- Best practices for designing and using partition keys effectively
- Best practices for using sort keys to organize data
- Best practices for using secondary indexes in DynamoDB
- Best practices for storing large items and attributes
- Best practices for handling time series data in DynamoDB
- Best practices for managing many-to-many relationships
- Best practices for implementing a hybrid database system
- Best practices for modeling relational data in DynamoDB
- Best practices for querying and scanning data
- Best practices for DynamoDB table design
- Best practices for DynamoDB global table design
- Best practices for managing the control plane in DynamoDB

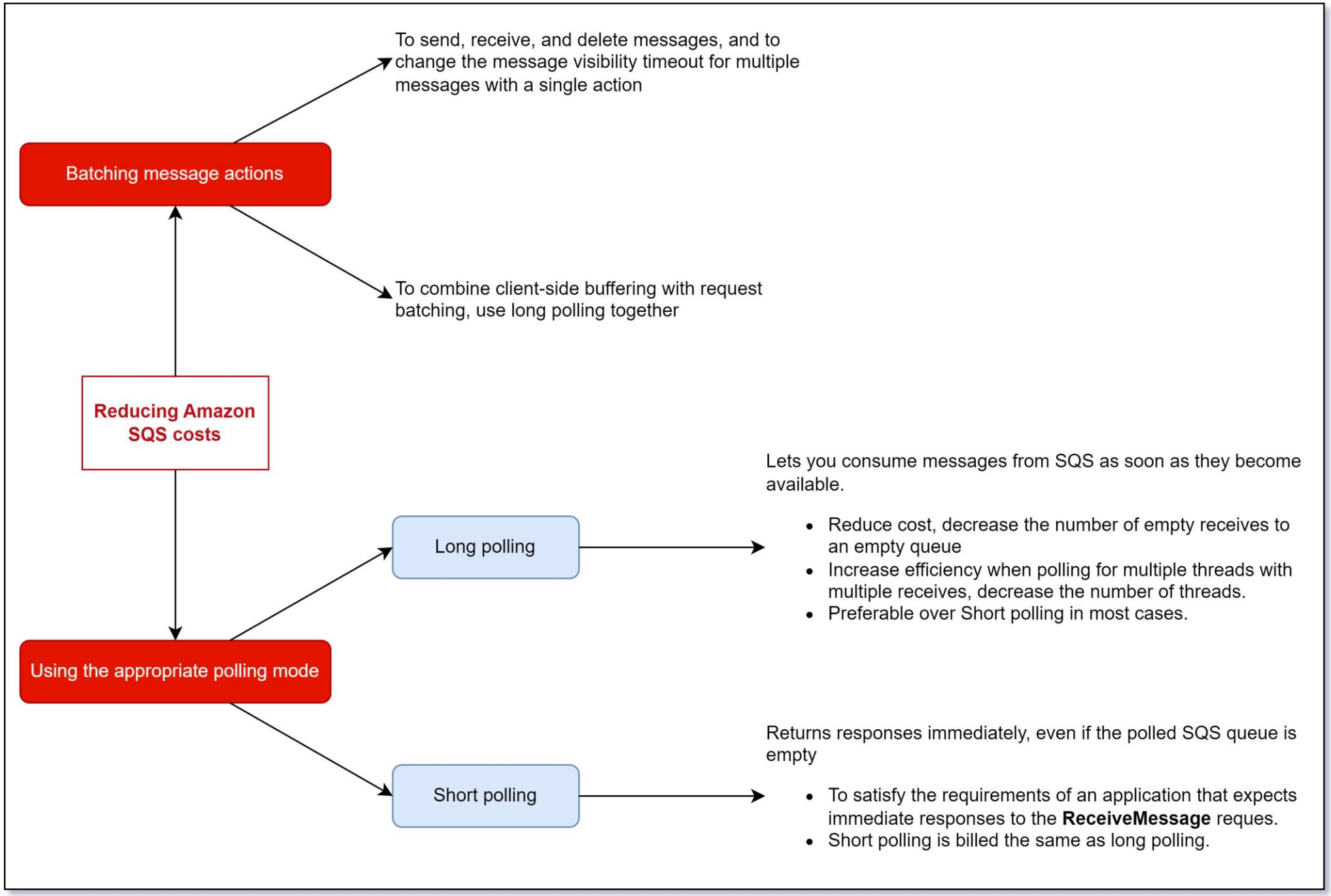

Understanding the business problems and the application use case before design schema
You should maintain as few tables as possible in a DynamoDB application
Two key concepts for NoSQL design
NoSQL Design
Approaching NoSQL design
Organize data
Keep related data together
Use sort order
Distribute queries
Use global secondary indexe
Identify specific query requirements
Data size
Estimate how much data will be stored and request once to determine the data partition.
Data shape
NoSQL database organizes data to correspond with what will be queried
Data velocity
Knowing the peak query loads to determine how to partition data to best use I/O capacity.

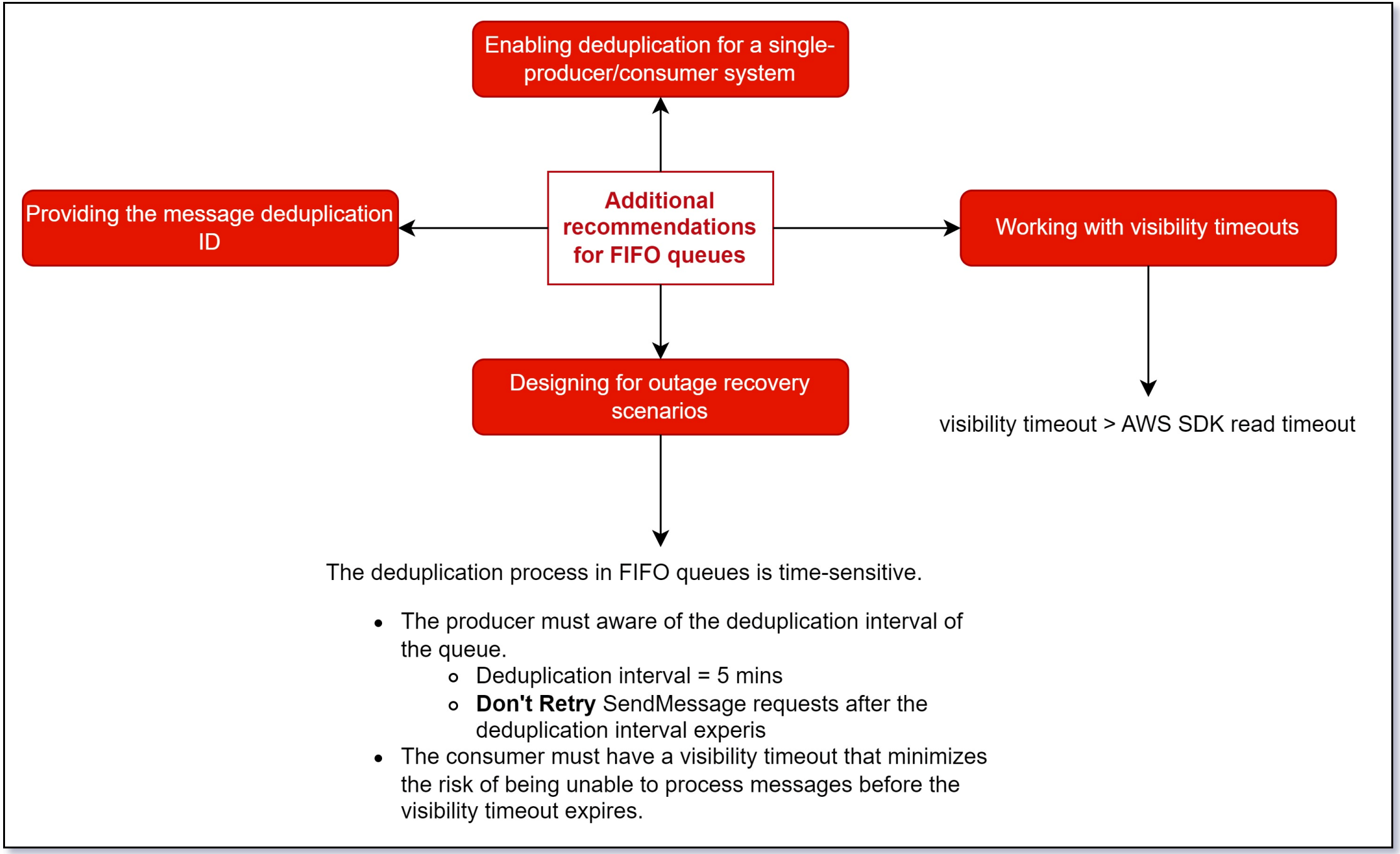

Amazon SQS (Simple Queue Service)
Nash Tech.

# Best practices for Amazon SQS

- Recommendations for standard and FIFO queues
- Additional recommendations for FIFO queues


Working with messages
Implementing request-response systems
Don't create reply queues per messages
Create Reply queues on startup, per producer
Don't let producers share reply queues
Processing messages in a timely manner
message's visibility timeout = maximum time to process and delte message
If you know how long it take to process a message
If you don't know
Create a heartbeat for your consumer process:
Specify the initial visibility timeout (example, 2 minutes)
As long as your consumer still works on the message, keep extending the visibility timeout by 2 minutes every minute.
Avoiding inconsistent message processing
Don't setting the maximum receives to 1 for a dead-letter queue
Handling request errors
Use an AWS SDK
Already have amotatic retry and backoff logic
Don't use the AWS SDK features
Allow a pause (example, 200ms) before retrying the ReceiveMessage action after receiving
no messages
a timeout
an error message
For subsequent use of ReceiveMessage that gives the same results, allow a longer pause (example, 400ms)
Setting up dead-letter queue retention
retention period of deadline-letter queue <= retention period of the origional queue
SQS
Capturing problematic messages
Capture all messages that can't be processed, configure a dead-letter queue
CloudWatch
Setting up long polling
Max polling wait time = 20 seconds
ReceiveMessage waite time = [1s, 20s]
If implement long polling for multiple queues:
One thread for each queue


To send, receive, and delete messages, and to change the message visibility timeout for multiple messages with a single action
Batching message actions
To combine client-side buffering with request batching, use long polling together
Reducing Amazon SQS costs
Lets you consume messages from SQS as soon as they become available.
Long polling
Reduce cost, decrease the number of empty receives to an empty queue
Increase efficiency when polling for multiple threads with multiple receives, decrease the number of threads.
Preferable over Short polling in most cases.
Using the appropriate polling mode
Returns responses immediately, even if the polled SQS queue is empty
Short polling
To satisfy the requirements of an application that expects immediate responses to the ReceiveMessage reques.
Short polling is billed the same as long polling.


Enabling deduplication for a single-producer/consumer system
Providing the message deduplication ID
Additional recommendations for FIFO queues
Working with visibility timeouts
Designing for outage recovery scenarios
visibility timeout > AWS SDK read timeout
The deduplication process in FIFO queues is time-sensitive.
The producer must aware of the deduplication interval of the queue.
Deduplication interval = 5 mins
Don't Retry SendMessage requests after the deduplication interval experis
The consumer must have a visibility timeout that minimizes the risk of being unable to process messages before the visibility timeout expires.

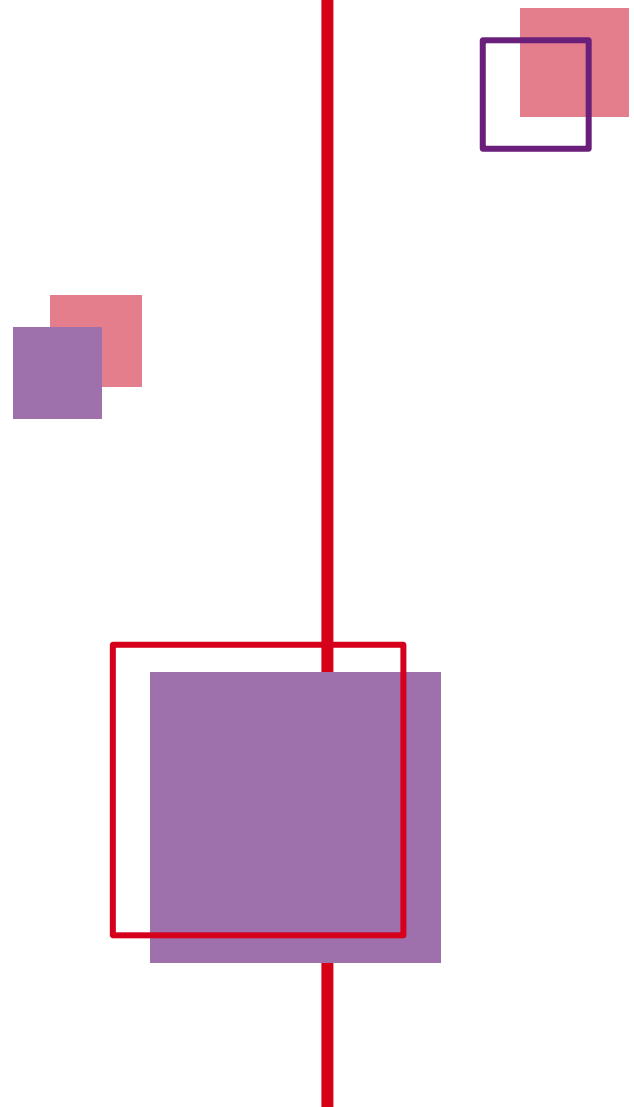
# AWS CloudFormation

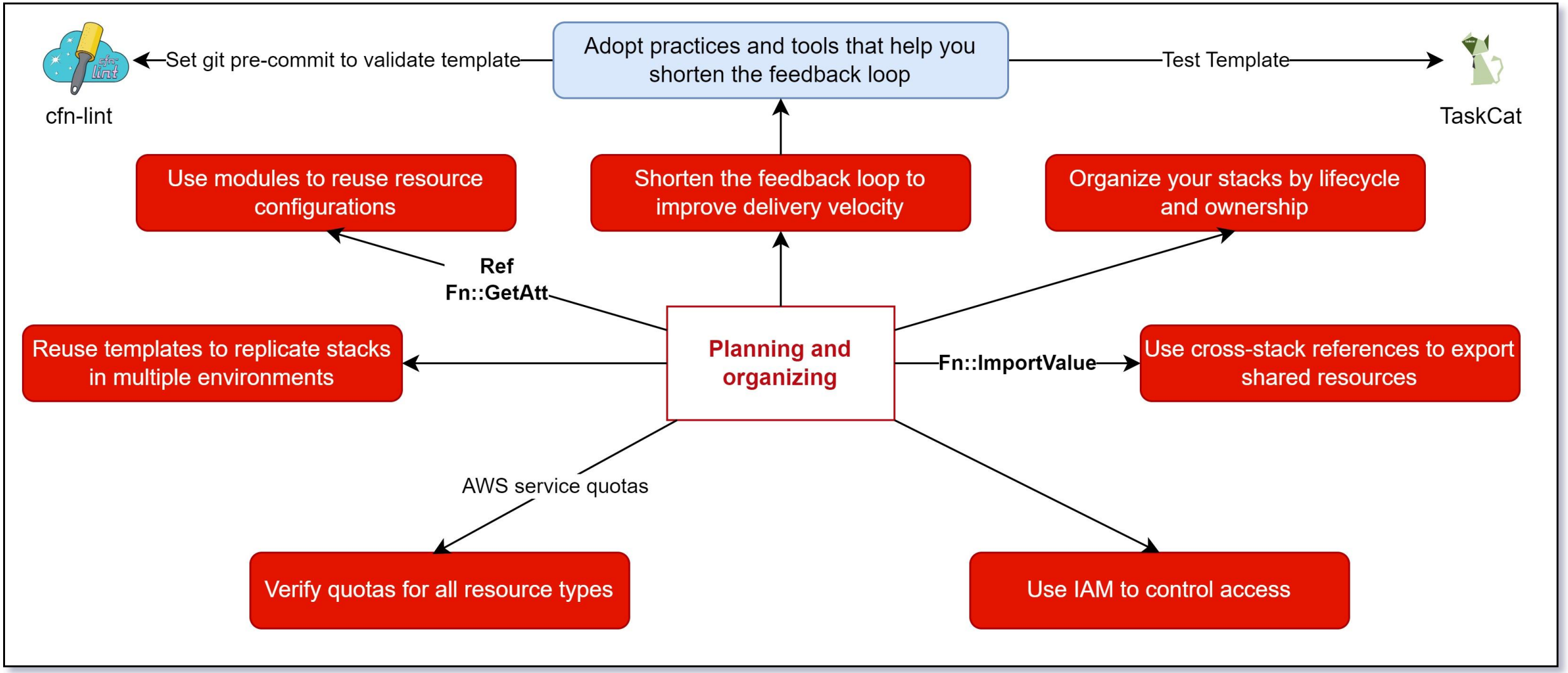
Nash Tech.

# Best practices

## Planning and organizing

- Shorten the feedback loop to improve delivery velocity
- Organize your stacks by lifecycle and ownership
- Use cross-stack references to export shared resources
- Use IAM to control access
- Reuse templates to replicate stacks in multiple environments
- Verify quotas for all resource types
- Use modules to reuse resource configurations

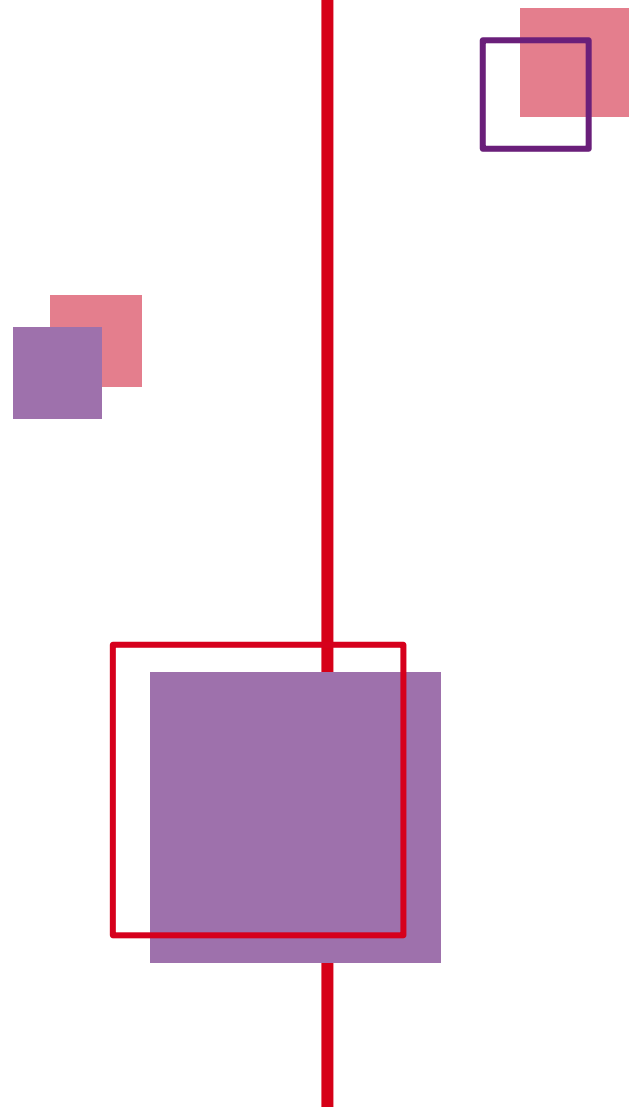


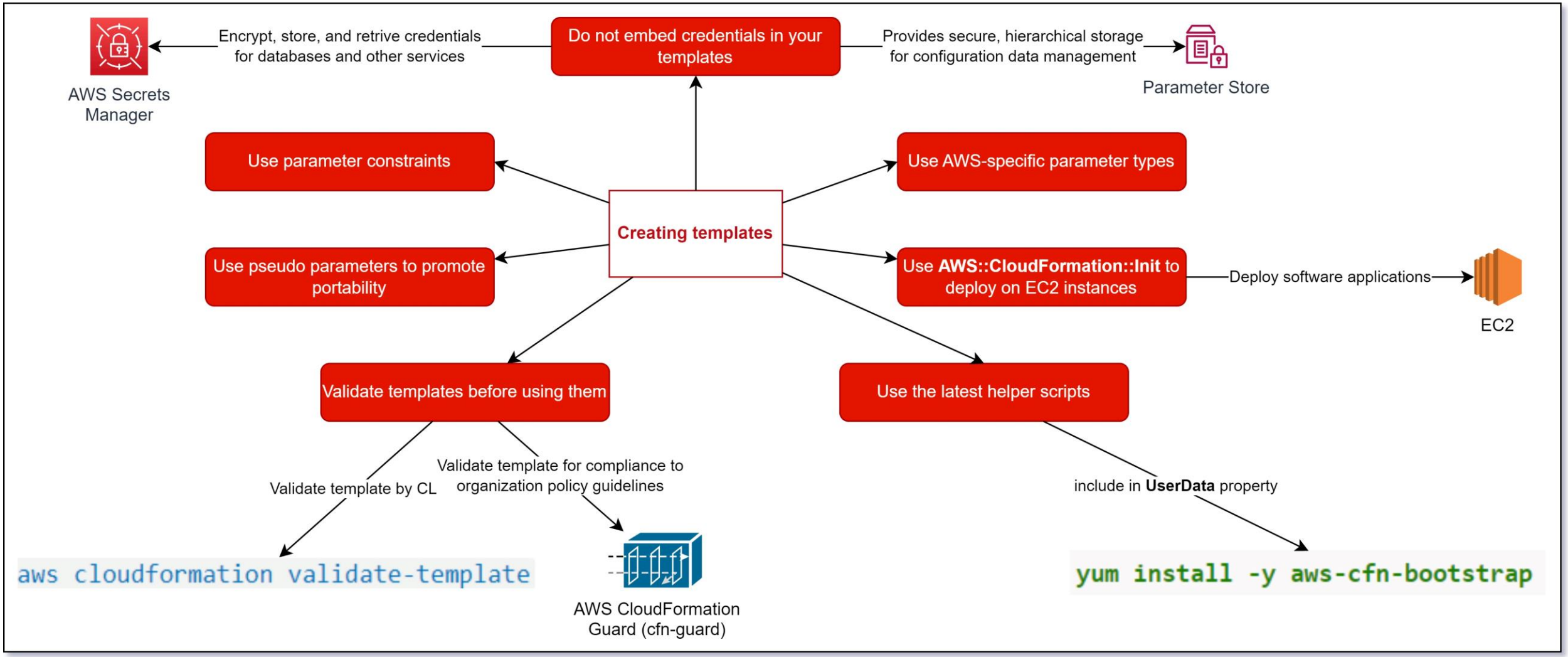

cfn-lint
Set git pre-commit to validate template
Adopt practices and tools that help you shorten the feedback loop
Test Template
TaskCat
Use modules to reuse resource configurations
Shorten the feedback loop to improve delivery velocity
Organize your stacks by lifecycle and ownership
Ref
Fn::GetAtt
Reuse templates to replicate stacks in multiple environments
Planning and organizing
Fn::ImportValue
Use cross-stack references to export shared resources
AWS service quotas
Verify quotas for all resource types
Use IAM to control access

# Best practices (2)

## Creating templates

- Do not embed credentials in your templates
- Use AWS-specific parameter types
- Use parameter constraints
- Use pseudo parameters to promote portability
- Use AWS::CloudFormation::Init to deploy software applications on Amazon EC2 instances
- Use the latest helper scripts
- Validate templates before using them

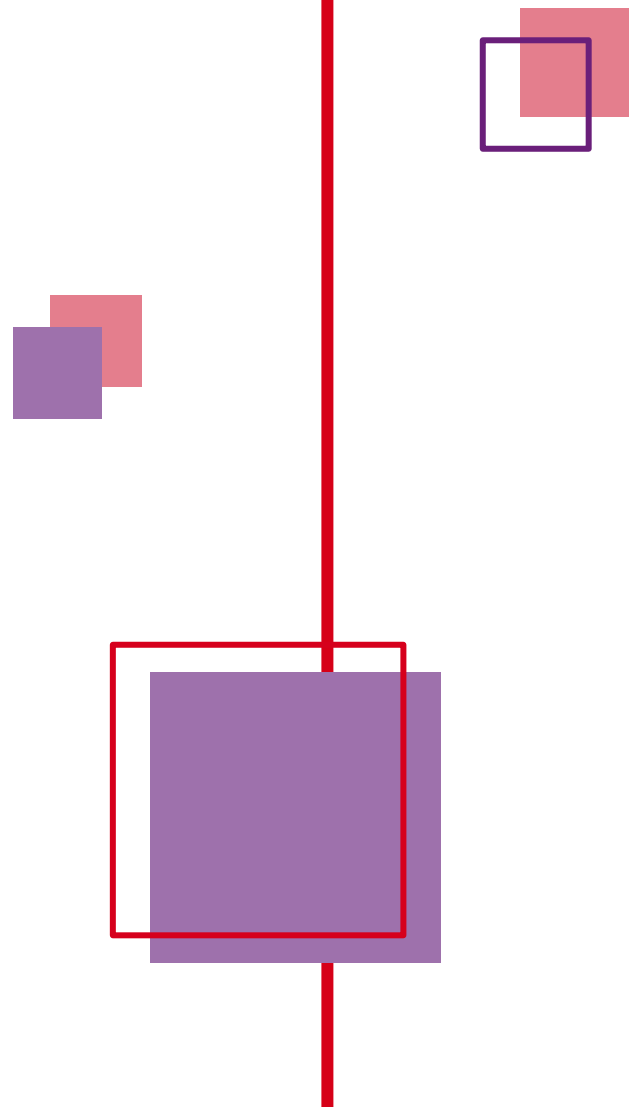


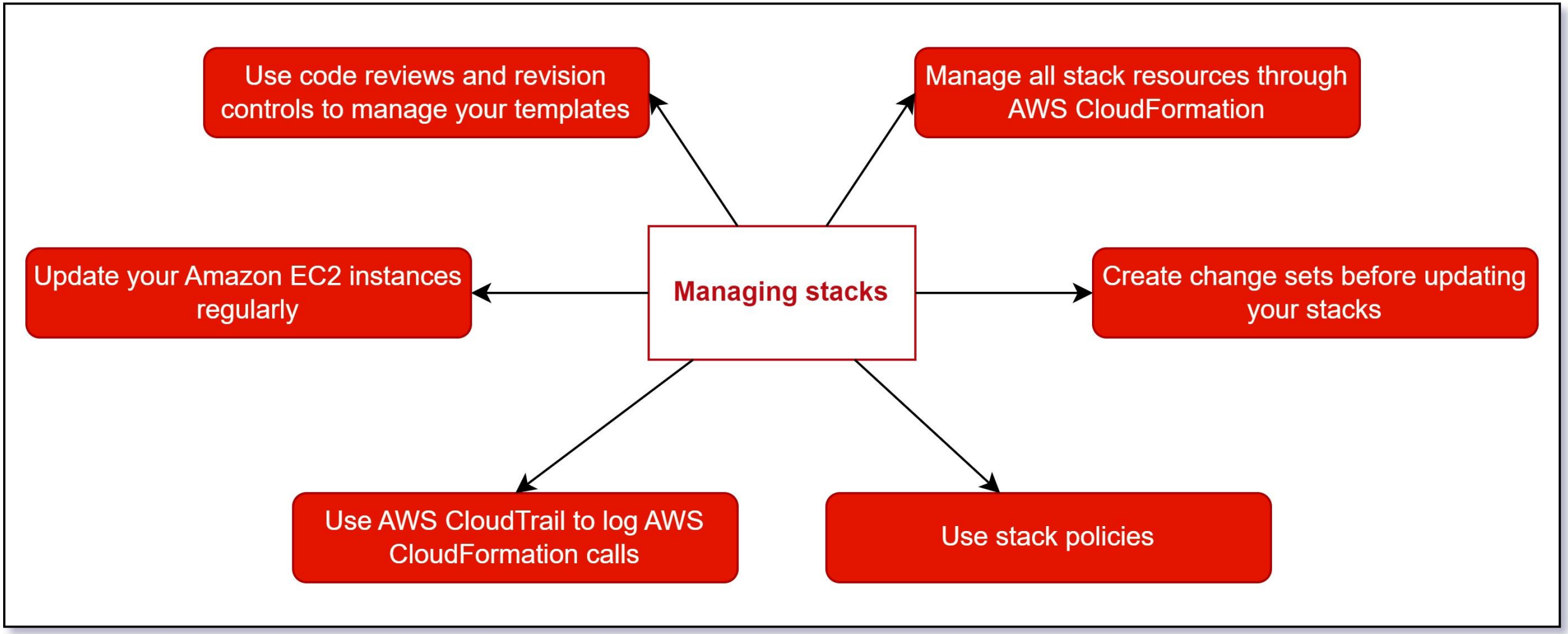

Creating templates
Do not embed credentials in your templates
Encrypt, store, and retrive credentials for databases and other services
AWS Secrets Manager
Provides secure, hierarchical storage for configuration data management
Parameter Store
Use parameter constraints
Use AWS-specific parameter types
Use pseudo parameters to promote portability
Use AWS::CloudFormation::Init to deploy on EC2 instances
Deploy software applications
EC2
Validate templates before using them
Use the latest helper scripts
Validate template by CL
Validate template for compliance to organization policy guidelines
include in UserData property
aws cloudformation validate-template
AWS CloudFormation Guard (cfn-guard)
yum install -y aws-cfn-bootstrap

# Best practices (3)

## Managing stacks

- Manage all stack resources through AWS CloudFormation
- Create change sets before updating your stacks
- Use stack policies
- Use AWS CloudTrail to log AWS CloudFormation calls
- Use code reviews and revision controls to manage your templates
- Update your Amazon EC2 instances regularly




Use code reviews and revision controls to manage your templates
Manage all stack resources through AWS CloudFormation
Update your Amazon EC2 instances regularly
Managing stacks
Create change sets before updating your stacks
Use AWS CloudTrail to log AWS CloudFormation calls
Use stack policies

# Thank you

Nash Tech.