

Cloud Application & Development Foundation

[BE SE Sixth Semester]

Nepal College of Information Technology
POKHARA UNIVERSITY

Unit II: Cloud Service Administration

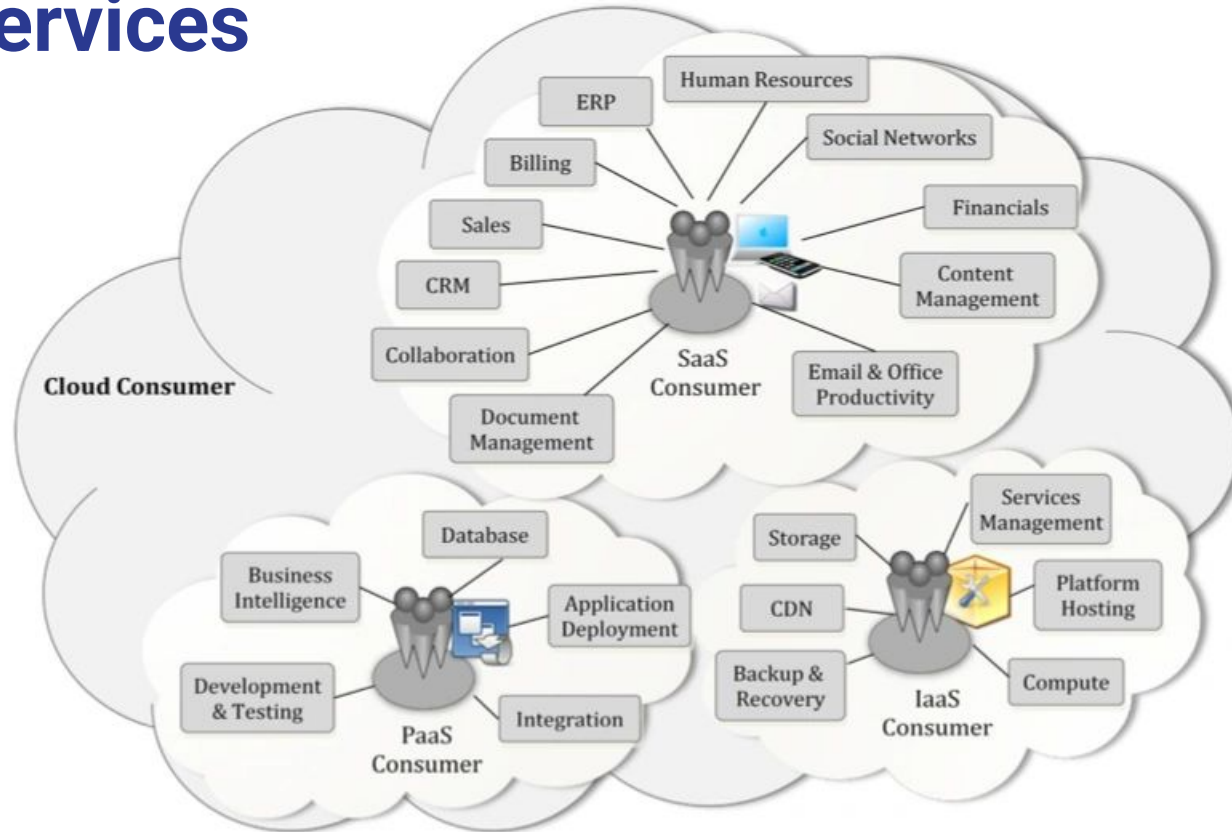
2.1 Monitoring and Support Services

2.2 Resource Management, Performance Management, Provisioning

2.3 IT Security in Cloud Infrastructure

2.4 Managing Environment Variables

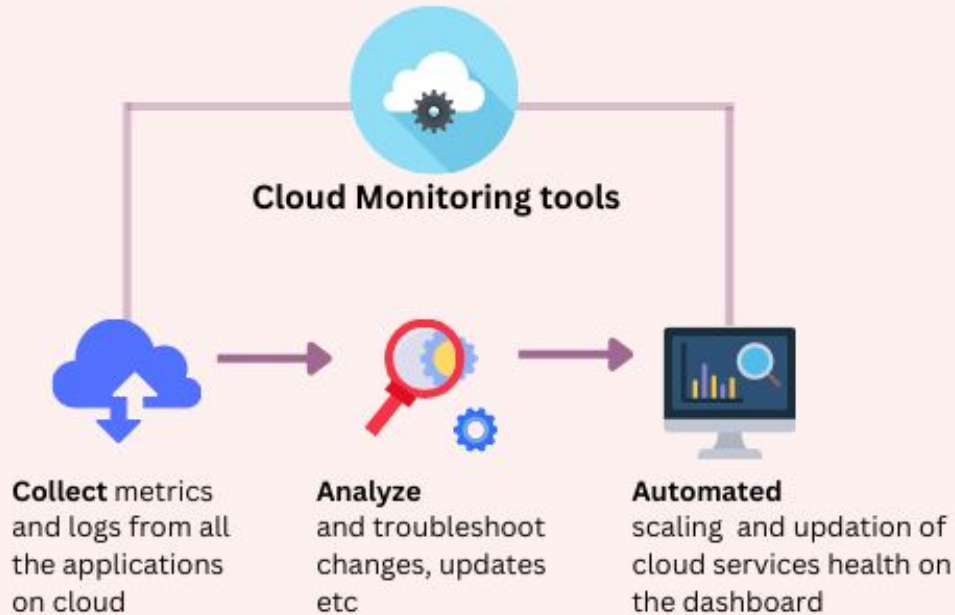
Cloud Services



Why Monitoring Matters

- Continuous observation of system health, performance, and security.
- Ensures system availability and reliability
- Detects performance bottlenecks proactively
- Prevents revenue loss from downtime
- Supports SLA compliance (e.g., 99.9%)
- Enables capacity planning and forecasting
- Critical for security breach detection
- Key Metrics: Latency, Error Rates, Traffic.

Cloud Monitoring Tools



Increased visibility into app



Application monitoring



Resource utilization



Improved operational health

Cloud Monitoring

- **Importance of monitoring cloud systems:**
Proactive detection of issues, performance optimization, capacity planning, and ensuring SLAs are met. Cloud isn't "set and forget."
- **Metrics vs. logs vs. traces:**
Metrics (numerical data over time – CPU utilization).
Logs (textual records of events).
Traces (request flow through distributed systems – useful for debugging).
- **Real-time vs. historical monitoring:**
Real-time for immediate alerting. Historical for trend analysis, capacity planning, and post-incident investigation.

"What metrics would you monitor to detect a database performance issue?"

Cloud Monitoring: AWS CloudWatch

- **CloudWatch** **metrics,** **dashboards,** **alarms:**
customizable dashboards thresholds and alerting severity levels.
- **Use cases:** monitoring EC2, Lambda, RDS: Specific examples showing how
- CloudWatch can be used to monitor key AWS services. EC2: CPU, memory.
Lambda: invocations, errors. RDS: database connections, query latency.
- CloudWatch Logs Insights for log querying
- Powerful tool for searching and analyzing logs.

Cloud Monitoring: Prometheus

- Open-source monitoring for Kubernetes and services:
Designed for dynamic, cloud-native environments. Its flexibility is a key strength.
- Data model: time series and metrics:
Prometheus stores data as time series. Focus on labels for adding metadata to metrics.
- PromQL overview with basic query examples:
Uses PromQL syntax (e.g., `rate(http_requests_total[5m])`), queries for calculating rates, averages, and percentiles.

Cloud Monitoring: Grafana

- Integration with Prometheus and CloudWatch:
Grafana can act as a central visualization platform for multiple data sources.
- **Dashboard creation and visualization:**
Different panel types (graphs, gauges, single stats), variable support for dynamic dashboards.
Emphasis on clear communication.
- Real-world example:
CPU/Memory dashboard for a microservice.
Has a pre-built dashboard with relevant metrics for a microservice.

Monitoring Tools Comparison

	ToolPros	Cons
● CloudWatch	Integrated, Easy Alerts	Limited Customization
● Prometheus	Open-Source, Scalable	Complex Setup
● Grafana	Rich Visualizations	No Native Alerting

Case

AWS US-East-1 outage (2021) → Loss of \$66M/hour due to poor monitoring.

Study:

Comparative Use-Cases and Best Practices

- When to use **CloudWatch** vs **Prometheus+Grafana**:
- **CloudWatch**: ease of use, AWS integration.
- **Prometheus+Grafana**: Kubernetes ecosystems, flexibility, vendor neutrality.
- Alerts, **SLAs**, and **SLOs**: Monitoring data supports meeting service level agreements (SLAs) and objectives (SLOs).
- Monitoring pitfalls and how to avoid them: Alert fatigue, monitoring irrelevant metrics, lack of contextual data.
- Monitoring as Code: Using **infrastructure-as-code** principles to define monitoring configurations.

Key Monitoring Metrics

Infrastructure: CPU/RAM/disk utilization

- **Application:** Error rates, latency
- **Network:** Bandwidth, packet loss
- **Business:** Transactions per second
- **Security:** Failed login attempts
- **Custom:** Domain-specific KPIs

CPU Utilization (%) = (1 - Idle Time / Total Time) × 100.

Application Metrics: Error rates (e.g., HTTP 500s), request latency.

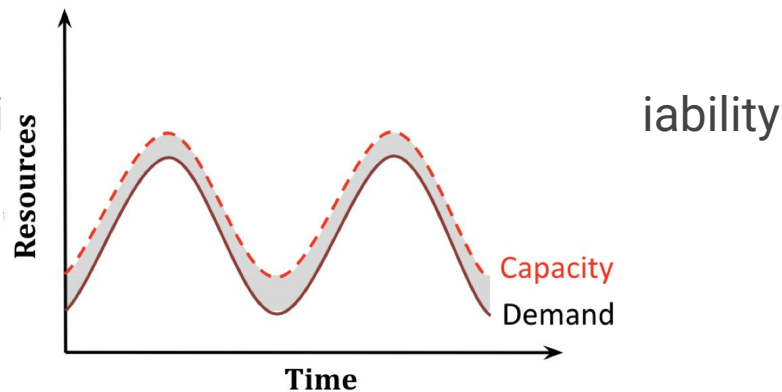
Business Metrics: User logins, transactions/minute.

Monitoring Architectures

- Push Model: Agents send metrics
- Pull Model: Server scrapes targets
- Centralized: Single management point
- Decentralized: Distributed collectors
- Hybrid: Combines push/pull
- Agentless: Uses APIs/SDKs

Resource Management Overview

- Understand the principles and importance of managing cloud resources effectively.
- Resource management involves provisioning, monitoring, and optimizing compute, storage, network, and database resources in cloud environments.
- Ensures cost efficiency, performance optimization, and scalability.
- Balances performance with cost.
- Critical for scalability of cloud applications.



Types of Cloud Resources

Each resource type has specific use cases and cost implications.
Understanding resource types is foundational to effective management.

- **Compute:**
Virtual machines (e.g., EC2, Azure VMs), containers (e.g., Kubernetes, Docker).
- **Storage:**
Block storage (e.g., EBS), object storage (e.g., S3, Blob Storage).
- **Network:**
Virtual private clouds (VPCs), load balancers, CDN.
- **Database:**
Managed databases (e.g., RDS, DynamoDB, Azure SQL).

Resource Allocation Strategies

- **Static**

Fixed resource provisioning based on predicted demand.

Allocation:

- **Dynamic**

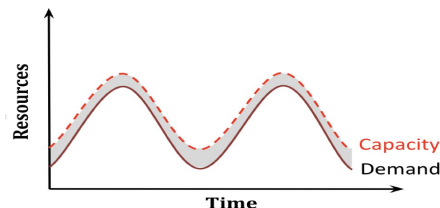
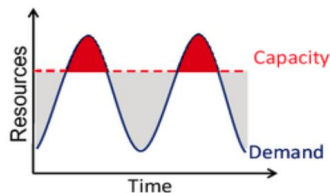
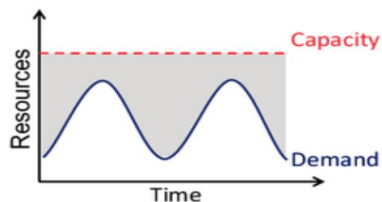
Adjusting resources in real-time based on workload (e.g., auto-scaling).

Allocation:

- **Load**

Distributing workloads across resources for optimal performance.

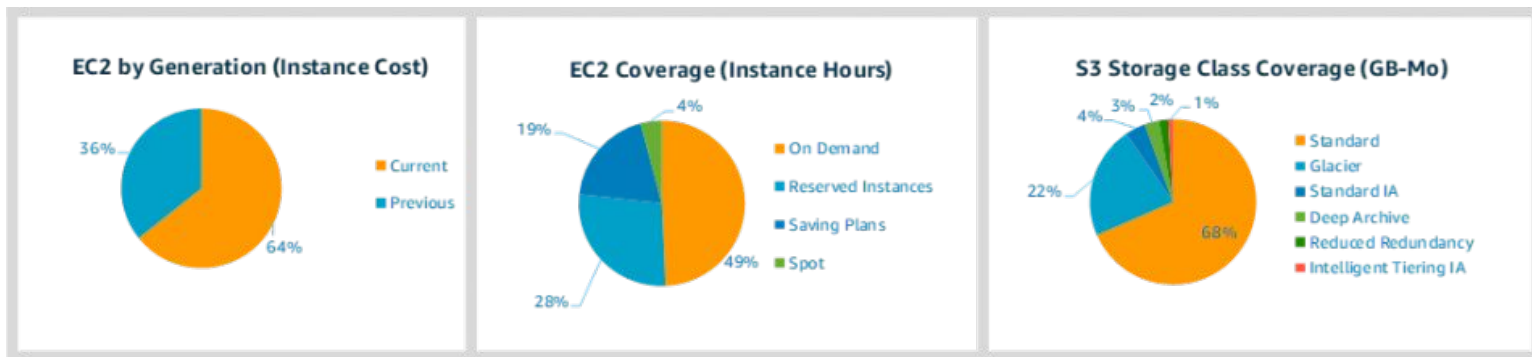
Balancing:



Dynamic allocation improves efficiency but requires careful configuration. Load balancing enhances reliability and user experience.

Cost Management in Resource Allocation

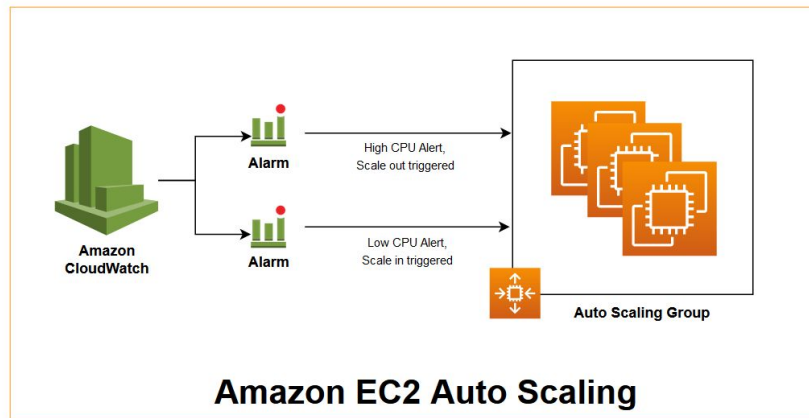
- Cost management is integral to resource management.
- Regular analysis prevents budget overruns.
- Tools: AWS Cost Explorer, Azure Cost Management, Google Cloud Billing.
- Techniques: Budget alerts, cost forecasting, and identifying underutilized resources.
- Reserved Instances vs. On-Demand: Trade-offs in cost and flexibility.



Auto-Scaling and Elasticity

- Auto-scaling: Automatically adjusting compute resources based on demand.
- Components: Scaling policies, triggers (e.g., CPU utilization, request rates), and cooldown periods.
- Examples: AWS Auto Scaling, Azure Scale Sets, GCP Managed Instance Groups.
- Ensures variable workloads efficiently.
- Requires careful over- or under-scaling.

ε



le
id

Auto-Scaling Fundamentals

Dynamic adjustment of resources based on demand.




Types:

- Reactive: Rules-based (e.g., CPU > 70% → add VM).
- Predictive: ML-driven (e.g., seasonal traffic patterns).

Scaling Policies:

- Step Scaling: Add 2 VMs if CPU > 75%,
4 VMs if > 85%.
- Target Tracking: Maintain 60% CPU utilization.

Auto-Scaling

Autoscaling Capabilities				 Google Cloud
Scaling policies	Simple scaling	yes	yes	no
	Target tracking	yes	no	yes
	Scheduled	yes	yes	yes
Spot instances	Usable in cluster	yes	yes	yes
	Mix on-demand / spot	yes	yes	no
Mix of VM types		yes	no	no

Reactive: Rules-based scaling

Predictive: ML-driven scaling

Horizontal: Add more instances

Vertical: Upgrade instance size

Cool-Down: Prevent rapid fluctuations

Policy Types: Step/target tracking

Load Balancing Metrics

- **Throughput:** Requests/second capacity
- **Latency:** End-to-end response time
- **Concurrency:** Active connections
- **Capacity Planning:** Peak vs. average
- **Queue Theory & Optimal Workers**

Load Balancing Algorithms

- Round Robin: Cyclic distribution
- Least Connections: Busy-aware
- IP Hash: Session persistence
- Weighted: Priority-based routing
- Geolocation: Regional routing
- Least Response Time: Performance-based

Throughput: Requests/sec

= (Workers × Avg. Request Handling Rate) - Overhead.

Case Study: How Twitter uses LB to handle 500M tweets/day.

Cost Optimization Strategies

- Right-Sizing: Match workload needs
- Spot Instances: Discounted capacity
- Reserved Instances: Long-term savings
- Scheduling: Start/stop non-production
- Tagging: Track resource ownership
- Governance: Enforce spending policies

Resource Monitoring and Optimization

- Continuous monitoring is essential for optimization.
- Rightsizing involves matching resource capacity to workload needs.
- **Monitoring** **Tools:**
AWS CloudWatch, Azure Monitor, Google Cloud Operations Suite.
- **Techniques:**
Identifying idle resources, rightsizing instances, and optimizing storage.
- **Optimization** **Goals:**
Reduce waste, improve performance, and lower costs.

Challenges in Resource Management

- **Over-provisioning:**
Allocating more resources than needed, increasing costs.
- **Under-provisioning:**
Insufficient resources leading to performance issues.
- **Complexity:**
Managing resources across hybrid or multi-cloud environments.
- **Noisy Neighbor:** Shared resources
- **Cold Start:** Scaling delays
- **Metric Overload:** Alert fatigue
- **Tool Sprawl:** Integration complexity

Resource Lifecycle Management

- Lifecycle management ensures resources are used efficiently.
- IaC enables repeatable and automated resource management.
- **Stages:**
Provisioning, utilization, monitoring, and deprovisioning.
- **Tools:**
Infrastructure as Code (IaC) for provisioning (e.g., Terraform, CloudFormation).
- **Importance of Cleanup:**
Avoiding orphaned resources to reduce costs.

Best Practices for Resource Management

- ❑ Use IaC for consistent provisioning.
- ❑ Implement tagging strategies for cost tracking.
- ❑ Regularly review and optimize resource usage.
- ❑ Leverage auto-scaling and load balancing for elasticity.
- ❑ Set up budget alerts to control costs.
- ❑ Best practices balance performance, cost, and scalability.
- ❑ Automation and monitoring are critical for efficiency.

Hands-On Activity: Configuring Auto-Scaling

Task:

Configure an auto-scaling group for a sample web application on AWS.

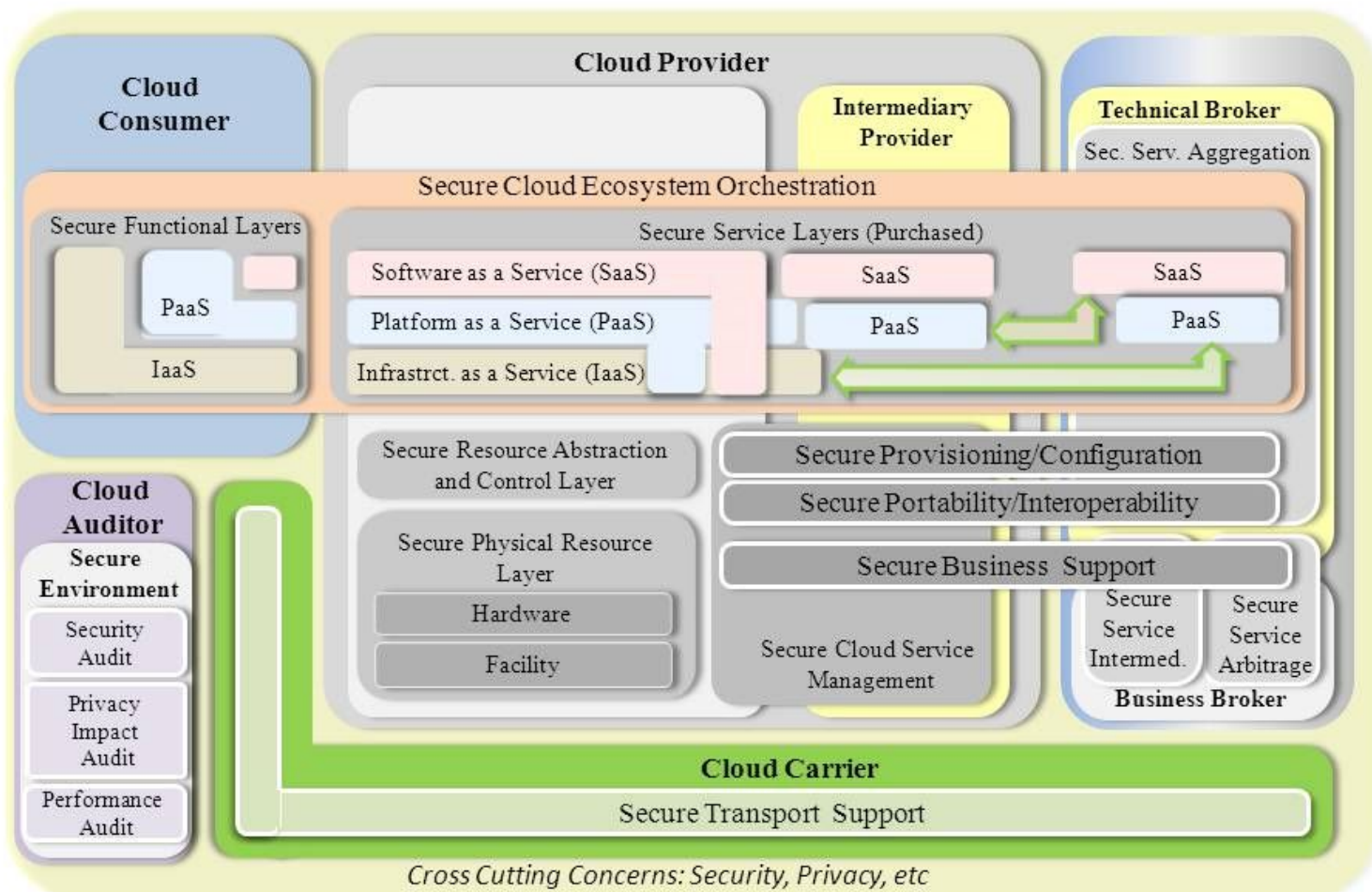
Steps:

- Create an EC2 instance template.
- Set up an auto-scaling group with CPU-based scaling policies.
- Simulate load to trigger scaling events.

Tools:

AWS Management Console or CLI.

NIST Cloud Computing Security Reference Architecture



IT Security in Cloud Infrastructure

- Understand the importance and challenges of securing cloud environments.
- **Definition:**
Cloud security encompasses policies, technologies, and controls to protect data, applications, and infrastructure in cloud environments.
- **Importance:**
Protects against data breaches, ensures compliance, and maintains user trust.
- Cloud security is a shared responsibility between providers and users.
- Evolving threats require proactive security measures.

Shared Responsibility Model

Divides security responsibilities between cloud provider and customer.

Understanding the model prevents security gaps.

Customers must secure their workloads effectively.

Provider Responsibilities: Physical security, network infrastructure, hypervisor.

Customer Responsibilities: Data protection, identity management, application security, and configurations.

Examples:

AWS, Azure, and GCP shared responsibility models.

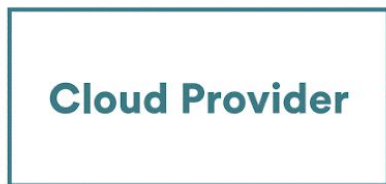
Shared Responsibility Model



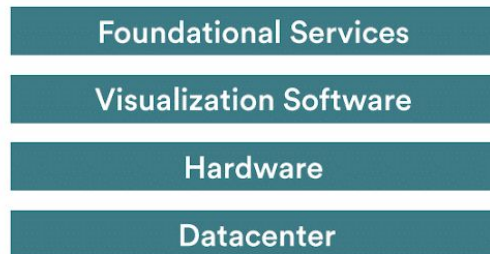
**Responsible for Security
'in' the Cloud**



SHARED RESPONSIBILITY MODEL



**Responsible for Security
'of' the Cloud**



Identity and Access Management (IAM)

Controls who can access cloud resources and what actions they can perform.

Components:

Users, groups, roles, policies, and multi-factor authentication (MFA).

Tools:

AWS IAM, Azure Active Directory, Google Cloud IAM.

Key Points:

- Principle of least privilege minimizes risk.
- MFA enhances account security.

Data Encryption

- Encryption protects sensitive data from unauthorized access.
- Key management is critical for compliance.
- **Types:** Encryption at rest (e.g., AES-256) and in transit (e.g., TLS/SSL).
- **Tools:**
AWS Key Management Service (KMS), Azure Key Vault, Google Cloud KMS.
- **Key Management:**
Generating, storing, and rotating encryption keys securely.

Network Security

- Secure network configurations prevent unauthorized access.
- Regular audits ensure compliance with security policies.
- **Techniques:**
Virtual Private Clouds (VPCs), security groups, network ACLs, and firewalls.
- **Tools:**
AWS VPC, Azure Virtual Network, Google Cloud VPC.
- **Best Practices:**
Network segmentation, restricting inbound/outbound traffic.

Threat Detection and Response

- Proactive threat detection reduces incident impact.
- Automated responses improve efficiency.
- Real-time monitoring, anomaly detection, and incident response workflows.
- Incident Response: Identifying, containing, and mitigating threats.

- **Useful**

Tools:

AWS GuardDuty, Azure Sentinel, Google Cloud Security Command Center.

Compliance and Governance

- Compliance ensures legal and regulatory adherence.
- Automated tools simplify compliance management.
- Auditing, reporting, and maintaining compliance documentation.
- Tools: AWS Config, Azure Policy, Google Cloud Security Compliance.
- Popular Standards: GDPR, HIPAA, SOC, PCI-DSS.
 - GDPR
 - HIPAA
 - SOC
 - PCI-DSS

Security Automation

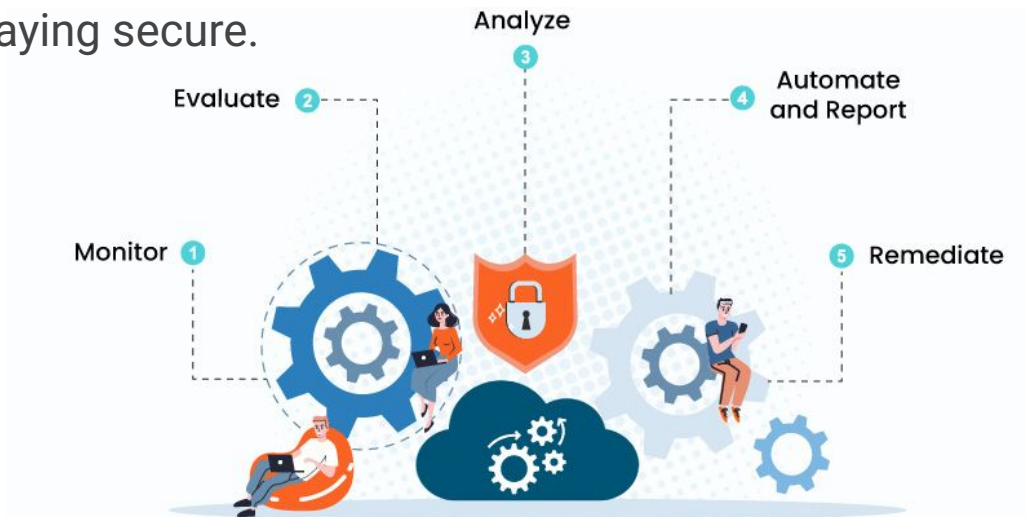
- Automated patch management, vulnerability scanning, and configuration checks.
- Reduces human error, improves response time.
- Automation enhances scalability of security processes.
- Regular updates are critical for staying secure.

Automation Tools

AWS Systems Manager,

Azure Automation,

Google Cloud Security Scanner.



Best Practices for Cloud Security

- ❑ Implement least privilege access with IAM.
- ❑ Encrypt all sensitive data at rest and in transit.
- ❑ Regularly update and patch systems.
- ❑ Monitor and log all activities for auditing.
- ❑ Conduct periodic security assessments and penetration testing.
- ❑ Proactive security measures prevent breaches.
- ❑ Layered security enhances resilience.

Environment Variables

- Key-value pairs used to configure applications dynamically.
- Decouple sensitive data (e.g., API keys, database credentials) from source code.
- Environment variables improve security by avoiding hardcoded values.
- Facilitate environment-specific configurations without code changes.
- Support flexible deployment across environments (e.g., dev, prod).
- Enable separation of configuration from code, enhancing portability & security.

Environment Variables: Use Cases

Storing sensitive data:

API keys, database credentials, and authentication tokens.

Configuration settings:

Database URLs, logging levels, and feature flags.

Environment-specific settings:

Different configurations for development, staging, and production.

Environment Variables in Cloud Platforms

- Each platform offers tools for managing variables securely.
- Integration with application runtimes varies by platform.
- Secure storage prevents unauthorized access.
- Automated rotation enhances security.
- AWS: Parameter Store, Secrets Manager.
- Azure: App Configuration, Key Vault.
- Google Cloud: Secret Manager, Cloud Functions environment variables.

Thank you
