PW SKILLS | DevOps and Cloud Computing

Securing Logs & Monitoring in CI/CD Pipelines

Objective

- Ensure sensitive data is not logged inadvertently.
- Implement encryption and RBAC policies to secure log access.
- Configure IAM roles in AWS to control log permissions.
- Set up log monitoring and anomaly detection with CloudWatch, Prometheus, and Grafana.
- Automate incident response using Terraform or Ansible.
- Integrate monitoring into CI/CD pipelines and track build metrics and logs.

PW SKILLS

Explaining the risks of logging personally identifiable information (PII) and sensitive data.

Let’s see

Logging personally identifiable information (PII) and sensitive data poses significant risks, both from a security and compliance standpoint. Here are the key risks:

- Data Breaches
- Regulatory Violations
- Internal Misuse
- Unintentional Exposure
- Long Retention Periods
- Audit and Incident Response Complexity

PW SKILLS

Discussing best practices for sanitizing logs (masking API keys, passwords, tokens).

Let’s discuss

Best Practices for Sanitizing Logs:

1. Avoid Logging Sensitive Data by Default
2. Mask Sensitive Fields
3. Use Structured Logging with Whitelisting
4. Implement Log Scrubbing Middleware
5. Environment-Based Logging Levels
6. Encrypt and Protect Logs
7. Regularly Audit Logs

PW SKILLS

Demonstrating configuring log filters to remove sensitive data before storing logs.

Let’s do it

Here’s a short example in Python using a custom log filter to remove sensitive data like passwords and API keys before storing logs:

```
import logging
import re

class SensitiveDataFilter(logging.Filter):
    def filter(self, record):
        # Mask common sensitive patterns
        record.msg = re.sub(r'("?(password|api_key|token)"?\s*[:=]\s*["\']?)[^"\']+',
                            r'\1****', str(record.msg))
        return True

logger = logging.getLogger("secureLogger")
logger.setLevel(logging.INFO)
handler = logging.FileHandler("app.log")
handler.addFilter(SensitiveDataFilter())
logger.addHandler(handler)

# Example usage
logger.info('User login with password="mySecret123" and api_key=ABCD1234')
```

PW SKILLS

Explaining why encrypting logs is critical to prevent unauthorized access.

Let’s see

- Encrypting logs is critical because it protects sensitive information from unauthorized access, especially if logs contain PII, passwords, tokens, or system details.
- Without encryption, attackers who gain access to storage or intercept data in transit can easily read and exploit the logs.
- Encryption ensures that even if logs are accessed, the data remains unreadable without proper decryption keys, helping maintain data confidentiality, integrity, and compliance with privacy regulations.

PW SKILLS

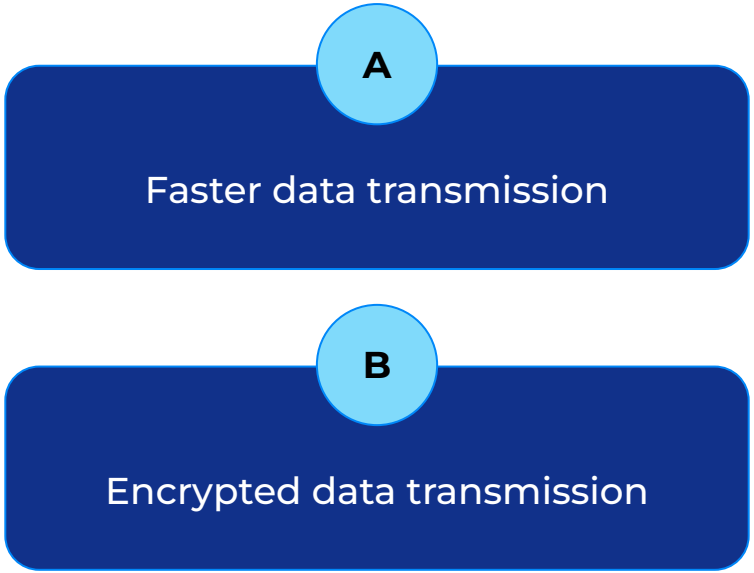
Discussing encryption at rest (S3, Elasticsearch, databases) and in transit (TLS, HTTPS).

Let’s discuss

- **Encryption at rest** (e.g., in S3, Elasticsearch, databases) protects stored log data from unauthorized access by encrypting it on disk.
- **Encryption in transit** (e.g., via TLS or HTTPS) secures data as it moves between systems, preventing interception or tampering during transmission.

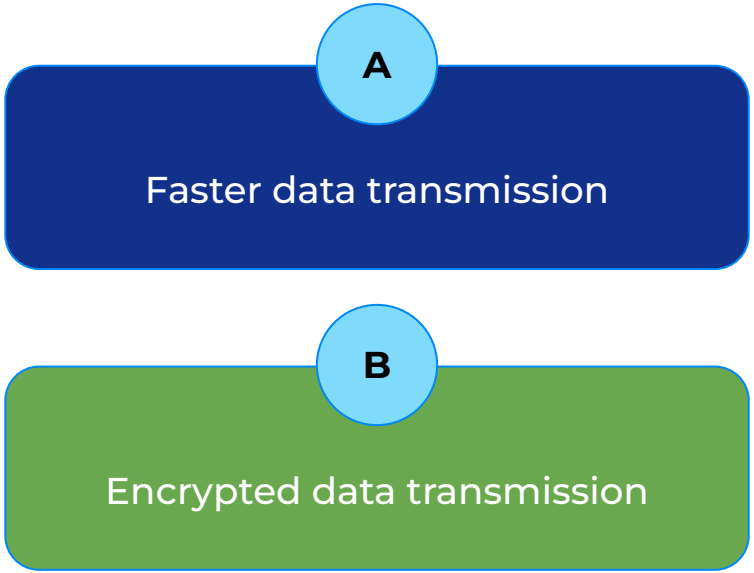
Pop Quiz

Q. What is the main advantage of using HTTPS over HTTP?



Pop Quiz

Q. What is the main advantage of using HTTPS over HTTP?



PW SKILLS

Introducing RBAC policies to limit access to logs based on user roles.

RBAC policies

Role-Based Access Control (RBAC) limits log access based on a user's role and responsibilities. For example:

- Admins may access all logs.
- Developers can view only application logs (excluding sensitive data).
- Auditors may access security logs but not modify them.

PW SKILLS

Explaining how IAM roles and policies control access to log data.

Let’s see

IAM roles and policies in cloud environments (like AWS, Azure, GCP) define who can access log data, what actions they can perform, and on which resources.

- IAM roles grant temporary permissions to users, applications, or services.
- IAM policies specify allowed or denied actions (e.g., s3:GetObject for S3 logs or logs:DescribeLogStreams for CloudWatch).

PW SKILLS

Demonstrating restricting access to CloudWatch logs using IAM policies.

Let’s do it

Here's a short example of an IAM policy that restricts access to specific CloudWatch log groups:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups",
        "logs:GetLogEvents",
        "logs:FilterLogEvents"
      ],
      "Resource": "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/my-app-log:*"
    }
  ]
}
```

PW SKILLS

Discussing auditing access logs to monitor log access patterns.

Let’s discuss

Auditing access logs involves tracking who accessed logs, when, and what actions they performed. This helps:

- Detect unauthorized or unusual access patterns
- Identify insider threats or privilege misuse
- Support compliance audits and forensic investigations

PW SKILLS

Explaining how anomaly detection works in log monitoring.

Let’s see

Anomaly detection in log monitoring uses rules or machine learning to identify log patterns that deviate from normal behavior. It works by:

- Learning baseline behavior
- Analyzing new logs
- Triggering alerts

PW SKILLS

Discussing CloudWatch Alarms and custom log filters for detecting suspicious activities.

Let’s discuss

CloudWatch Alarms:

- Monitor metrics like CPU usage, error rates, or login failures.
- Trigger actions (e.g., email, Lambda function, SNS) when thresholds are exceeded.
- Example: Alert if 5xx error rate exceeds 5% in a given time window.

Custom Log Filters:

Analyze CloudWatch Logs using pattern matching.

Search for suspicious behavior like:

- Repeated failed login attempts
- Unauthorized API calls
- Unexpected IP addresses

PW SKILLS

Demonstrating setting up an alert for critical log events (e.g., failed login attempts, API errors).

Let’s do it

1. Create a Log Group: Ensure logs (e.g., from Lambda or EC2) are sent to CloudWatch Logs.
2. Add a Metric Filter:

   Example filter for failed logins:

```
{ $.eventName = "ConsoleLogin" && $.errorMessage = "Failed" }
```

3. Assign a Metric Name: e.g., FailedLoginCount
4. Create a CloudWatch Alarm:
   - Set threshold (e.g., if FailedLoginCount ≥ 3 in 5 minutes)
   - Choose an action (e.g., send SNS notification)

Pop Quiz

Q. What is a custom log filter in CloudWatch Logs used for?

A

To encrypt logs before storing them

B

To search and match patterns in log data

Pop Quiz

Q. What is a custom log filter in CloudWatch Logs used for?

A

To encrypt logs before storing them

B

To search and match patterns in log data

PW SKILLS

Explaining automated incident response using Infrastructure as Code (IaC).

Let’s see

- Automated incident response using Infrastructure as Code (IaC) enables predefined actions (like isolating instances or revoking access) to be triggered automatically by alerts.
- Tools like Terraform or CloudFormation ensure fast, consistent, and error-free responses to security incidents, reducing manual intervention and downtime.

PW SKILLS

Discussing how Terraform or Ansible can provision resources automatically in response to alerts.

Let’s discuss

Terraform:

- Define infrastructure in code (e.g., security groups, EC2 quarantine VPC).
- Trigger via tools like AWS Lambda or CI/CD pipelines when alerts fire.
- Example: On detecting a threat, a Lambda function runs terraform apply to isolate a compromised instance.

Ansible:

- Automate tasks (e.g., blocking IPs, restarting services, deploying patches).
- Can be triggered by alert handlers (e.g., Ansible Tower with webhook from monitoring).
- Example: On repeated failed logins, Ansible runs a playbook to lock the user or adjust firewall rules.

PW SKILLS

Demonstrating an automated remediation workflow.

Let’s do it

Step 1: Detect Unauthorized Access

- Use AWS CloudWatch Logs with a custom filter:

```
{ $.errorCode = "*UnauthorizedOperation*" }
```

- Create a Metric Filter and CloudWatch Alarm to monitor this event.

Step 2: Trigger a Lambda Function

- The alarm sends a notification to SNS, which triggers a Lambda function.
- Lambda executes a remediation script (e.g., runs Terraform or calls Ansible).

Let’s do it

Step 3A: Run Terraform to Block Access

```
# Inside Lambda or automation server
cd /path/to/terraform/security_patch
terraform init
terraform apply -auto-approve
```

Step 3B: Run Ansible to Apply Patches

```
ansible-playbook block_access.yml --extra-vars "target_ip=192.0.2.1"
```

Outcome

- Unauthorized activity is detected, and access is automatically blocked or mitigated, with no manual intervention.

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

Explaining how monitoring should be part of infrastructure automation.

Let’s see

Monitoring should be integrated into infrastructure automation to ensure systems are observable from the start.

- It enables automatic deployment of monitoring tools, dashboards, and alerts through code, making the setup consistent and scalable.
- This helps detect issues early, supports auto-remediation, and provides valuable insights for performance, security, and cost optimization.

PW SKILLS

Discussing using Terraform to provision Prometheus & Grafana dashboards automatically.

Let’s discuss

Using Terraform, you can automatically provision Prometheus and Grafana dashboards as part of your infrastructure setup.

- Terraform can deploy Prometheus via Helm (in Kubernetes) or configure it on VMs, and use the Grafana provider to create data sources and import dashboards using JSON or HCL.
- This makes monitoring setup repeatable, version-controlled, and scalable, eliminating manual configuration and ensuring observability is built into your infrastructure from the start.

PW SKILLS

Demonstrating how Terraform can deploy and configure Prometheus exporters.

Let’s do it

Terraform can deploy and configure Prometheus exporters by provisioning the required infrastructure (like EC2 or Kubernetes), installing exporters (e.g., Node Exporter, Blackbox Exporter) using user data, provisioners, or Helm charts, and updating the Prometheus config to scrape these targets.

Example (Kubernetes - Helm):

```
resource "helm_release" "node_exporter" {
  name       = "prometheus-node-exporter"
  repository = "https://prometheus-community.github.io/helm-charts"
  chart      = "prometheus-node-exporter"
  namespace  = "monitoring"
}
```

PW SKILLS

Explaining the importance of monitoring CI/CD pipelines.

Let’s see

Monitoring CI/CD pipelines is crucial to identify inefficiencies and optimize performance:

- **Build Duration:** Long builds can signal codebase bloat, poor test design, or inefficient tooling, slowing down delivery and developer feedback loops.
- **Resource Consumption:** High CPU or memory usage may indicate misconfigured jobs or resource leaks, leading to slower builds, higher costs, or system instability.

PW SKILLS

Demonstrating how Prometheus scrapes CI/CD metrics and Grafana visualizes them.

Let’s do it

Prometheus scrapes CI/CD metrics by targeting endpoints exposed by tools like Jenkins, GitLab CI, or Tekton, which provide Prometheus-formatted metrics (e.g., at /metrics).

Example Prometheus config:

```
scrape_configs:
  - job_name: 'jenkins'
    static_configs:
      - targets: ['jenkins.example.com:8080']
```

Grafana then queries Prometheus and visualizes these metrics (e.g., build duration, success rate) using prebuilt or custom dashboards, enabling real-time CI/CD performance insights.

PW SKILLS

Explaining why centralizing build logs improves debugging and compliance.

Let’s see

Centralizing build logs improves debugging by making it easy to trace errors, compare builds, and analyze failures across environments in one place.

- It enhances compliance by securely storing logs with audit trails, retention policies, and access controls, ensuring accountability and meeting regulatory requirements.

PW SKILLS

Discussing how to collect logs from Jenkins/GitHub Actions and store them in Elasticsearch.

Let’s discuss

To collect logs from Jenkins or GitHub Actions and store them in Elasticsearch:

- For Jenkins, use plugins like Logstash or configure log forwarding with Filebeat, which ships logs to Elasticsearch.
- For GitHub Actions, export logs via GitHub’s API or third-party actions, then send them to Elasticsearch using tools like Logstash, Filebeat, or a custom script.

PW SKILLS

Demonstrating configuring a Jenkins pipeline to forward logs to ELK Stack.

Let’s do it

To configure a Jenkins pipeline to forward logs to the ELK Stack:

- Install Filebeat on the Jenkins server.
- Configure filebeat.yml to watch Jenkins log files:

```
filebeat.inputs:
  - type: log
    paths:
      - /var/log/jenkins/jenkins.log

output.elasticsearch:
  hosts: ["http://elasticsearch:9200"]
```

Let’s do it

- Enable and start Filebeat:

```
sudo systemctl enable filebeat
sudo systemctl start filebeat
```

PW SKILLS

Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this sessio
- Review your lecture notes
- Note down questions and queries regarding this session ar
  consult the teaching assistants

Thanks !