Project Title: Cloud Infrastructure Security Assessment Framework

Short Project Description:

The framework automates resource inventory collection, security analysis, compliance benchmarking, risk scoring, and report generation for cloud environments (AWS, Azure, GCP, etc).

Component	Role
Cloud Inventory Collector	Lists cloud resources (instances, buckets, VMs, databases, etc)
Configuration Analyzer	Evaluates security settings (e.g., S3 bucket policies, etc)
Compliance Benchmark Evaluator	Measures configs against benchmarks (CIS AWS, Azure Security Center, etc)
Risk Scorer	Assigns a security score to the environment
Reporting Engine	Generates findings and recommendations report

Component Details:

1. **Cloud Inventory Collector**:
   - Uses APIs (e.g., AWS boto3, Azure SDK, etc) to pull resource lists.
2. **Configuration Analyzer**:
   - Checks resources for:
     - Public exposure
     - Weak IAM permissions
     - Insecure storage
     - Etc
3. **Compliance Benchmark Evaluator**:
   - Runs checks against CIS, NIST standards, NIS 2, etc.
4. **Risk Scorer**:
   - Weighs vulnerabilities and calculates risk scores.
5. **Reporting Engine**:
   - Prepares structured recommendations.

Overall System Flow:

- Input: Cloud environment credentials
- Output: Cloud security audit report
- Focus on **configuration and compliance auditing**

Internal Functioning of Each Module:

1. Cloud Inventory Collector

- **How it works:**
  - Connects to the cloud provider’s API (AWS, Azure, GCP, etc).
  - Uses SDKs like **Boto3** (AWS), **Azure SDK**, or **Google Cloud SDK**, etc.
  - Enumerates all resources:
    - EC2 instances, S3 buckets, Lambda functions (AWS), etc
    - Storage accounts, VMs (Azure), etc
    - Compute Engine, Buckets (GCP), etc
- **Output:** A structured list of all resources with metadata (location, permissions, creation date, etc).

---

2. Configuration Analyzer

- **How it works:**
  - For each resource from the inventory:
    - Pulls configuration properties (ACLs, policies, public exposure settings, etc).
  - Checks for:
    - Open S3 buckets
    - IAM roles that allow *"Resource: ″ access
    - Storage with public-read or write
    - Etc
- **Techniques:**
  - Static rules (hardcoded)
  - Context-aware analysis (e.g., trust boundaries, etc)

---

3. Compliance Benchmark Evaluator

- **How it works:**
  - Maps resource configurations to compliance frameworks:
    - CIS AWS Foundations Benchmark
    - PCI-DSS
    - ISO 27001
    - NIS 2
    - HIPAA
    - Etc
  - Each check looks like:
    - "Ensure S3 buckets are not publicly accessible" (yes / no)
- **Result:** Pass/Fail per compliance item.

---

4. Risk Scorer

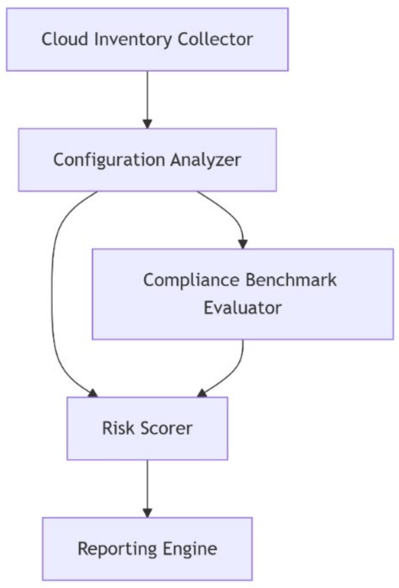
- **How it works:**
    - Assigns a **risk score** to each finding based on:
        - Severity (Critical, High, Medium, Low)
        - Resource sensitivity
        - Number of affected resources
    - Calculates overall "Cloud Environment Risk Score" (0-100%).

---

5. Reporting Engine

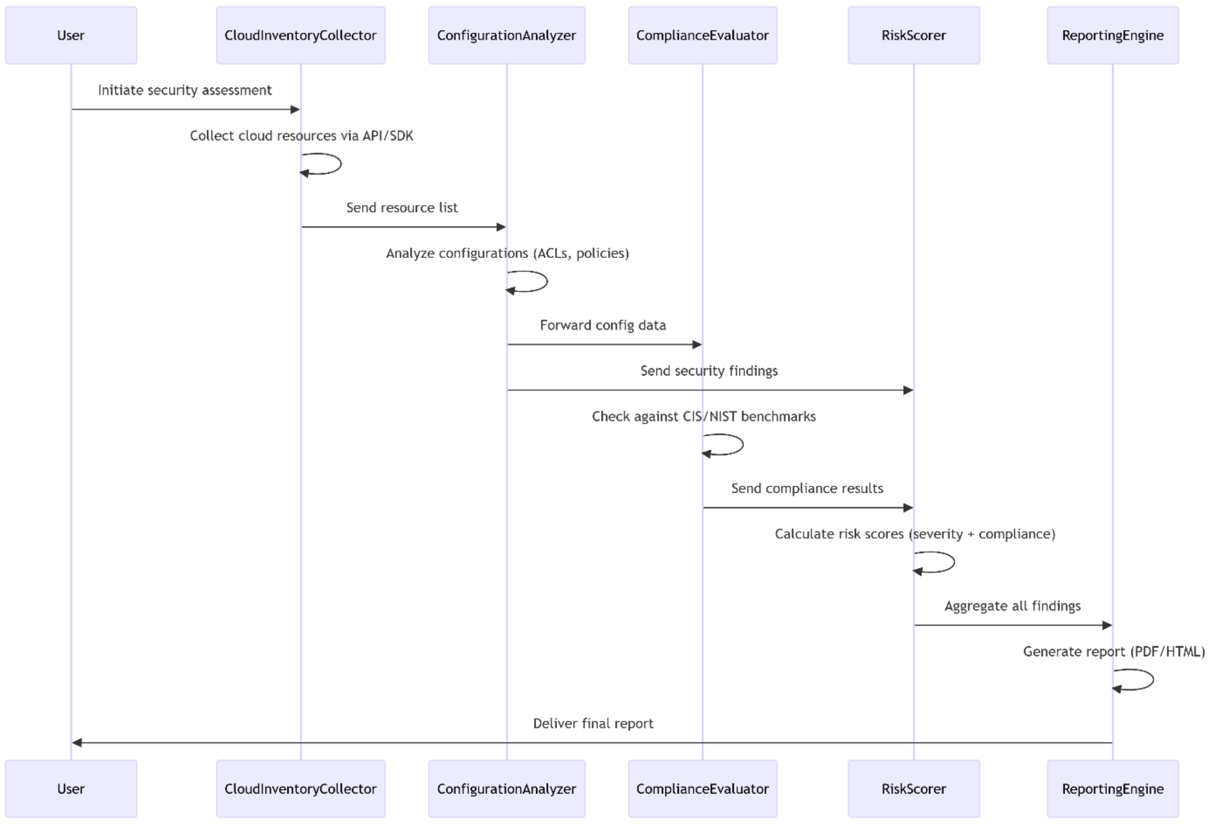
- **How it works:**
    - Aggregates results.
    - Organizes by severity and service (S3, EC2, IAM, etc).
    - Recommends specific remediations.
- **Format:** PDF/HTML reports with risk matrices, etc.

Component Diagram



- The **Cloud Inventory Collector** gathers resource metadata and sends it to the **Configuration Analyzer**.
- The **Configuration Analyzer** checks security settings and shares results with both the **Compliance Benchmark Evaluator** (for standards checks) and the **Risk Scorer** (for risk calculation).
- The **Risk Scorer** combines security and compliance data to compute a risk score, then forwards it to the **Reporting Engine**.

Sequence Diagram



- The **User** triggers the assessment, initiating resource collection by the **Cloud Inventory Collector**.
- The **Configuration Analyzer** evaluates security configurations and sends findings to the **Compliance Evaluator** and **Risk Scorer**.
- The **Compliance Evaluator** validates against benchmarks and shares results with the **Risk Scorer**.
- The **Risk Scorer** synthesizes data into a risk score, which the **Reporting Engine** formats into a user-ready report.

Detailed Project Description: Cloud Infrastructure Security Assessment Framework

A framework for auditing cloud infrastructure security configurations and compliance. The framework automates resource inventory collection, security analysis, compliance benchmarking, risk scoring, and report generation for cloud environments (AWS, Azure, GCP, etc).

---

1. System Overview

The framework assesses cloud security posture by analyzing resource configurations against security best practices and compliance standards (e.g., CIS, PCI-DSS, NIS 2, etc). This framework generates actionable reports highlighting risks and remediation steps.

Key Components

1. **Cloud Inventory Collector**
2. **Configuration Analyzer**
3. **Compliance Benchmark Evaluator**
4. **Risk Scorer**
5. **Reporting Engine**

---

2. Component Design & Implementation

2.1 Cloud Inventory Collector

Functionality:

- Discovers and catalogs cloud resources (e.g., S3 buckets, VMs, databases, etc).

Implementation Steps (e.g.):

1. **Cloud Provider Integration**:
    - **AWS**: Use `Boto3` SDK to call APIs like `DescribeInstances`, `ListBuckets`.

    - **Azure**: Use `Azure SDK for Python` to query resources via Azure Resource Graph.
    - **GCP**: Use `Google Cloud Client Library` for Compute Engine and Storage APIs.
2. **Authentication**:
    - Securely manage credentials using environment variables or vaults (e.g., AWS IAM roles, Azure Managed Identity, etc).
3. **Data Structuring (e.g.)**:
    - Store inventory in JSON format with metadata:
      ```json
      {
        "resource_id": "s3-bucket-123",
        "type": "AWS::S3::Bucket",
        "region": "us-east-1",
        "public_access": true,
        "creation_date": "2023-01-01"
      }
      ```
4. **Challenges (e.g.)**:
    - **Rate Limiting**: Implement retries with exponential backoff.
    - **Multi-Account Support**: Use AWS Organizations or Azure Management Groups.

Output:

- Structured inventory of cloud resources.

2.2 Configuration Analyzer

Functionality:

- Identifies misconfigurations (e.g., public S3 buckets, overly permissive IAM policies, etc).

Implementation Steps (e.g.):

1. **Rule Definition**:

    - Define static security rules in YAML (e.g.):

      ```yaml
      rules:
        - id: "S3_PUBLIC_ACCESS"
          description: "Check for publicly accessible S3 buckets"
          condition: "resource.public_access == true"
          severity: "High"
      ```

    - Use regex or policy language (e.g., Open Policy Agent, etc) for complex checks.

2. **Analysis Workflow**:
    - For each resource, evaluate rules against its configuration.
3. **Example Checks**:
    - **AWS**:
        - S3 bucket ACL allows `Everyone` access.
        - IAM policies with `"Effect": "Allow"` and `"Resource": "*"`.
    - **Azure**:
        - Storage accounts with public blob containers.
        - VMs with open SSH/RDP ports.

Output:

- List of security findings (resource ID, rule violated, severity).

2.3 Compliance Benchmark Evaluator

Functionality:

- Validates configurations against compliance frameworks (CIS, NIST, NIS 2, etc).

Implementation Steps (e.g.):

1. **Compliance Mapping**:
   - Map CIS AWS Benchmark controls to resource checks (e.g.):
     ```
     cis_checks = {
       "CIS-1.3": {
         "description": "Ensure S3 buckets are not publicly accessible",
         "rule_id": "S3_PUBLIC_ACCESS"
       }
     }
     ```
2. **Evaluation**:
   - Compare Configuration Analyzer findings against compliance controls.
3. **Result Format**:
   - Pass/fail status per compliance item.

Output:

- Compliance report (framework, control ID, status).

2.4 Risk Scorer

Functionality:

- Calculates risk scores for the environment.

Implementation Steps (e.g.):

1. **Severity Weights**:
   - Assign weights: Critical=10, High=5, Medium=3, Low=1.
2. **Scoring Formula**:
   - `Risk Score = Σ (Severity Weight × Resource Sensitivity) / Total Resources`
   - **Resource Sensitivity**: Use tags (e.g., `env:prod` doubles weight).
3. **Normalization**:
   - Convert to 0–100 scale.

Output:

- Numeric risk score and prioritized findings.

2.5 Reporting Engine

Functionality:

- Generates human-readable and machine-readable reports.

Implementation Steps (e.g.):

1. **Template Design**:
   - **HTML**: Use Jinja2 for dynamic content (e.g., tables, charts, etc).
   - **PDF**: Convert HTML to PDF with `WeasyPrint` or `wkhtmltopdf`.
   - **SARIF**: Export for CI/CD integration.
2. **Content**:
   - **Executive Summary**: Risk score, top findings.
   - **Detailed Findings**: Resource, violation, remediation.
   - **Compliance Summary**: Pass/fail status per framework.

Output:

- Final report in HTML/PDF/SARIF formats.

---

3. Technology Stack (e.g.)

- **Cloud SDKs**: Boto3 (AWS), Azure SDK, Google Cloud Client Library, etc.
- **Rule Engine**: Open Policy Agent (OPA) or custom YAML/JSON rules, etc.
- **Reporting**: Jinja2, WeasyPrint, SARIF SDK, etc.
- **Data Storage**: SQLite (for small-scale) or PostgreSQL (enterprise), etc.

---

4. Evaluation & Validation (e.g.)

1. **Accuracy Testing**:
   - Deploy test environments with known vulnerabilities (e.g., public S3 buckets, etc).
   - Verify if the tool detects all misconfigurations.
2. **Compliance Validation**:
   - Compare results against AWS Security Hub or Azure Security Center.
3. **Performance**:
   - Measure time to scan environments with 1000+ resources.

---

5. Development Roadmap

1. **Phase 1**: Build AWS inventory collector and basic Configuration Analyzer.
2. **Phase 2**: Integrate CIS benchmarks and risk scoring.
3. **Phase 3**: Add multi-cloud support (Azure, GCP).
4. **Phase 4**: Develop reporting engine and test.

---

6. Challenges & Mitigations (optional)

- **Complex Policies**: Use OPA for declarative policy checks.
- **Credential Security**: Leverage cloud-native secret managers (AWS Secrets Manager, Azure Key Vault, etc).
- **Scalability**: Implement parallel API calls and pagination.

---

7. Glossary

- **CIS**: Center for Internet Security
- **IAM**: Identity and Access Management

- **SARIF**: Static Analysis Results Interchange Format
- **OPA**: Open Policy Agent
