BLACKBOXAI

Generated on: 9/26/2025 9:19:49 AM

User

Aws project for automation for aviation company saffaran the reports of turbine engines jets using api, python, Teraform, kubernetes, docker, ec2, s3, IAM, REDIS, VPC, JENKINS, CLOUDWATCH, CLOUDTRAIL, GITHUB, SECURITY, MFA, etc.. Share step by step process and download the zip as repository github]

BLACKBOXAI

Below is a comprehensive, step-by-step guide to building an AWS-based automation project for an aviation company called "Saffaran." The project focuses on automating the generation and management of reports for turbine engine jets (e.g., performance metrics, maintenance logs, and failure predictions). We'll use a Python-based API to fetch/process data (simulating engine telemetry via a mock API), store reports in S3, cache data in Redis, deploy the application in a containerized environment (Docker + Kubernetes on EKS), provision infrastructure with Terraform, automate CI/CD with Jenkins, monitor with CloudWatch and CloudTrail, secure everything with IAM, VPC, MFA, and more. The entire setup will be version-controlled in a GitHub repository.

This is a production-ready blueprint, but it's simplified for clarity. You'll need an AWS account with billing enabled (free tier eligible for some components). Total setup time: 4-6 hours for basics, plus testing.

Important Notes:

- Security: Always enable MFA on your AWS root account and IAM users. Use least-privilege IAM policies. This guide includes security best practices.
- Costs: Expect ~$50-100/month for a small setup (EC2 t3.micro, EKS, etc.). Use AWS Cost Explorer to monitor.
- Assumptions: Engine data comes from a mock REST API (e.g., simulating sensor data for jets like Boeing 787 turbines). In reality, integrate with real aviation APIs (e.g., from GE Aviation or Boeing).
- GitHub Repo: I can't directly provide a downloadable ZIP (as I'm an AI assistant), but I'll guide you to create one. At the end, I'll provide a structure and sample files you can copy-paste into a new GitHub repo. Clone it locally and ZIP it if needed.

Project Architecture Overview

- Frontend/Backend: Python Flask API app (fetches engine data, generates PDF/CSV reports using libraries like ReportLab/Pandas).
- Data Flow: API → Python app (processes data) → Redis (caching) → S3 (report storage).
- Infrastructure: VPC for isolation → EC2 (Jenkins for CI/CD) → EKS (Kubernetes cluster for app deployment) → Docker (containerization).

- Automation: Terraform for IaC → Jenkins for pipelines → GitHub for source control.
- Monitoring/Security: CloudWatch (logs/metrics) → CloudTrail (audits) → IAM (roles/policies) → MFA (enforced).
- Scalability: Kubernetes auto-scales pods; S3 for durable storage.

High-level diagram (text-based):

```
 GitHub Repo → Jenkins (CI/CD) → Terraform (Infra) → VPC/EC2/EKS
Python API (Dockerized) → Fetches Engine Data → Redis Cache → Generates Reports → S3 Storage
Monitoring: CloudWatch + CloudTrail
Security: IAM + MFA
```

Prerequisites

- 1. AWS Account: Sign up at aws.amazon.com. Enable MFA on root account (via IAM console: Users → Your User → Security credentials → Assign MFA device).
- 2. Tools Installation (on your local machine, e.g., Ubuntu/Mac):

- AWS CLI: `curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip" && unzip awscliv2.zip && sudo ./aws/install` . - Configure: `aws configure` (use Access Key/Secret from IAM → Users → Security credentials). - Terraform: Download from terraform.io (v1.5+), add to PATH. - kubectl: For Kubernetes, `curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl" && sudo install kubectl /usr/local/bin/` . - Docker: Install from docker.com. - Python 3.10+: With pip. - Git: Installed. - Jenkins: We'll deploy it on EC2.

- 3. GitHub Account: Create a repo named "saffaran-engine-reports-automation".
- 4. Libraries: `pip install flask pandas reportlab redis boto3 docker kubernetes` .

Step-by-Step Setup Process

Step 1: Set Up Security Basics (IAM, MFA, CloudTrail)

- 1. Enable MFA:

- In AWS Console → IAM → Users → Select your user → Security credentials → Manage → Assign MFA device (use authenticator app like Google Authenticator). - Enforce for all IAM users: IAM → Dashboard → Activate MFA.

- 2. Create IAM Roles/Policies:

- Policy for EC2/Jenkins (e.g., `SaffaranEC2Policy`): Allows EC2:S3, EC2:CloudWatch, EKS access. - JSON (create in IAM → Policies → Create policy):

```
json
      {
          "Version": "2012-10-17",
          "Statement": [
```

```
            {
                "Effect": "Allow",
                "Action": [
                    "s3:GetObject,PutObject,ListBucket",
                    "cloudwatch:PutMetricData",
                    "eks:DescribeCluster"
                ],
                "Resource": ""
            }
        ]
    }
```

- Role for EC2: IAM → Roles → Create role → EC2 → Attach `SaffaranEC2Policy`. - User for Terraform/Jenkins: IAM → Users → Add user (programmatic access) → Attach policy above. Download keys.

- 3. Enable CloudTrail:

- Console → CloudTrail → Create trail → Name: `saffaran-trail` → Apply to all regions → S3 bucket: Create `saffaran-cloudtrail-logs` → Enable log file validation. - This audits all API calls for security.

Step 2: Set Up Networking (VPC with Terraform)

- 1. Create a directory: `mkdir saffaran-project && cd saffaran-project`.
- 2. Initialize Git: `git init && git remote add origin https://github.com/yourusername/saffaran-engine-reports-automation.git`.
- 3. Create `terraform/main.tf` for VPC (isolated network):

```
hcl
  provider "aws" {
    region = "us-east-1"  # Change as needed
  }

  resource "aws_vpc" "saffaran_vpc" {
    cidr_block = "10.0.0.0/16"
    tags = { Name = "saffaran-vpc" }
  }

  resource "aws_subnet" "public_subnet" {
    vpc_id     = aws_vpc.saffaran_vpc.id
    cidr_block = "10.0.1.0/24"
    availability_zone = "us-east-1a"
    tags = { Name = "public-subnet" }
  }

  resource "aws_internet_gateway" "igw" {
    vpc_id = aws_vpc.saffaran_vpc.id
  }

  resource "aws_route_table" "rt" {
    vpc_id = aws_vpc.saffaran_vpc.id
    route {
      cidr_block = "0.0.0.0/0"
      gateway_id = aws_internet_gateway.igw.id
```

```
  }
}

resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.rt.id
}


 # Security Group for EC2/EKS (allow SSH 22, HTTP 80/443, Jenkins 8080)
resource "aws_security_group" "saffaran_sg" {
  vpc_id = aws_vpc.saffaran_vpc.id
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Restrict to your IP in prod
  }
  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags = { Name = "saffaran-sg" }
}
```

- 4. Run Terraform: `terraform init && terraform plan && terraform apply` (type "yes"). This creates VPC, subnet, IGW, route table, and SG.

Step 3: Deploy Storage and Caching (S3, Redis)

- 1. S3 Bucket for Reports (via Terraform, add to `main.tf`):

```
hcl
  resource "aws_s3_bucket" "saffaran_reports" {
    bucket = "saffaran-engine-reports-${random_string.bucket_suffix.result}"  # Unique name
  }

  resource "aws_s3_bucket_public_access_block" "saffaran_reports" {
    bucket = aws_s3_bucket.saffaran_reports.id
    block_public_acls       = true
    block_public_policy     = true
    ignore_public_acls      = true
    restrict_public_buckets = true
  }
```

```
resource "random_string" "bucket_suffix" {
  length  = 8
  special = false
  upper   = false
}
```

- Re-run `terraform apply`. Bucket policy: IAM → Bucket policy → Add to allow only your IAM user.

- 2. Redis (via ElastiCache) (add to `main.tf`):

```
hcl
resource "aws_elasticache_subnet_group" "saffaran_redis" {
  name       = "saffaran-redis-group"
  subnet_ids = [aws_subnet.public_subnet.id]
}


resource "aws_elasticache_cluster" "saffaran_redis" {
  cluster_id           = "saffaran-redis"
  engine               = "redis"
  node_type            = "cache.t3.micro"
  num_cache_nodes      = 1
  parameter_group_name = "default.redis7"
  port                 = 6379
  subnet_group_name    = aws_elasticache_subnet_group.saffaran_redis.name
  security_group_ids   = [aws_security_group.saffaran_sg.id]
}
```

- Re-apply Terraform. Note Redis endpoint: `terraform output` (add output block if needed).

Step 4: Build the Python Application (API for Reports)

- 1. Create `app.py` (Flask API to fetch mock engine data, cache in Redis, generate/store report in S3):

```
python
from flask import Flask, jsonify
import redis
import boto3
import pandas as pd
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
import io
import requests  # For mock API

app = Flask(__name__)
r = redis.Redis(host='saffaran-redis.cache.amazonaws.com', port=6379, db=0)  # Replace with
actual endpoint
s3 = boto3.client('s3', aws_access_key_id='YOUR_KEY', aws_secret_access_key='YOUR_SECRET')
# Use IAM role in prod

@app.route('/generate_report/')
def generate_report(engine_id):
```

```
        # Fetch mock data from API (simulate turbine telemetry)
        response = requests.get(f'https://jsonplaceholder.typicode.com/posts/{engine_id}')  #
Replace with real aviation API
        data = response.json()

        # Cache in Redis
        r.setex(f"engine_{engine_id}", 3600, str(data))  # 1-hour TTL

        # Process to DataFrame
        df = pd.DataFrame([data])

        # Generate PDF report
        buffer = io.BytesIO()
        p = canvas.Canvas(buffer, pagesize=letter)
        p.drawString(100, 750, f"Turbine Engine Report for Jet {engine_id}")
        p.drawString(100, 700, f"Status: {data.get('title', 'OK')}")
        p.save()
        buffer.seek(0)

        # Upload to S3
        s3_bucket = 'saffaran-engine-reports-randomsuffix'  # From Terraform
        s3_key = f'reports/engine_{engine_id}.pdf'
        s3.put_object(Bucket=s3_bucket, Key=s3_key, Body=buffer)

        return jsonify({"status": "Report generated", "s3_url": f"s3://{s3_bucket}/{s3_key}"})


    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000)
```

- 2. Dockerfile (containerize the app):

```
dockerfile
  FROM python:3.10-slim
  WORKDIR /app
  COPY requirements.txt .
  RUN pip install -r requirements.txt
  COPY app.py .
  EXPOSE 5000
  CMD ["python", "app.py"]
```

- 3. requirements.txt:

```
flask==2.3.3
  redis==4.6.0
  boto3==1.28.0
  pandas==2.0.3
  reportlab==4.0.4
  requests==2.31.0
```

- 4. Test locally: `docker build -t saffaran-app . && docker run -p 5000:5000 saffaran-app`. Hit `http://localhost:5000/generate_report/1`.

Step 5: Deploy Infrastructure for App (EC2 for Jenkins, EKS for Kubernetes)

- 1. EC2 for Jenkins (add to `main.tf`):

```
hcl
   data "aws_ami" "amazon_linux" {
     most_recent = true
     owners      = ["amazon"]
     filter {
       name   = "name"
       values = ["amzn2-ami-hvm--x86_64-gp2"]
     }
   }

   resource "aws_instance" "jenkins_ec2" {
     ami           = data.aws_ami.amazon_linux.id
     instance_type = "t3.micro"
     subnet_id     = aws_subnet.public_subnet.id
     vpc_security_group_ids = [aws_security_group.saffaran_sg.id]
     iam_instance_profile = aws_iam_instance_profile.saffaran_profile.name  # Create profile
with EC2 role
     user_data = <<-EOF
       #!/bin/bash
       yum update -y
       yum install -y java-11-amazon-corretto docker git
       service docker start
       usermod -a -G docker ec2-user
       # Install Jenkins
       wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
       rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
       yum install -y jenkins
       systemctl start jenkins
       systemctl enable jenkins
     EOF
     tags = { Name = "saffaran-jenkins" }
   }

   # IAM Instance Profile (create role first as in Step 1)
   resource "aws_iam_instance_profile" "saffaran_profile" {
     name = "saffaran_profile"
     role = aws_iam_role.saffaran_ec2_role.name  # Reference your EC2 role
   }

   resource "aws_iam_role" "saffaran_ec2_role" {
     name = "saffaran_ec2_role"
     assume_role_policy = jsonencode({
       Version = "2012-10-17"
       Statement = [{
         Action = "sts:AssumeRole"
         Effect = "Allow"
         Principal = { Service = "ec2.amazonaws.com" }
       }]
```

```
    })
  }


   resource "aws_iam_role_policy_attachment" "saffaran_policy_attach" {
    role       = aws_iam_role.saffaran_ec2_role.name
    policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"  # Add more as
needed
  }
```

- Re-apply Terraform. SSH to EC2 (get public IP from console): `ssh -i your-key.pem ec2-user@IP`. - Get Jenkins initial password: `sudo cat /var/log/jenkins/jenkins.log | grep -i password`. - Access Jenkins: `http://EC2-IP:8080` → Unlock → Install suggested plugins (Docker, Git, Terraform, Kubernetes).

- 2. EKS for Kubernetes (add to `main.tf`):

```
hcl
  resource "aws_eks_cluster" "saffaran_eks" {
    name     = "saffaran-eks"
    role_arn = aws_iam_role.eks_role.arn  # Create EKS role with AmazonEKSClusterPolicy
    vpc_config {
      subnet_ids = [aws_subnet.public_subnet.id]
    }
  }


   # Node Group
  resource "aws_eks_node_group" "saffaran_nodes" {
    cluster_name    = aws_eks_cluster.saffaran_eks.name
    node_group_name = "saffaran-nodes"
    node_role_arn   = aws_iam_role.eks_node_role.arn  # Create with AmazonEKSWorkerNodePolicy,
etc.
    subnet_ids      = [aws_subnet.public_subnet.id]
    instance_types  = ["t3.small"]
    scaling_config {
      desired_size = 2
      max_size     = 3
      min_size     = 1
    }
  }
```

- Create required IAM roles for EKS (similar to EC2, attach `AmazonEKSClusterPolicy`, `AmazonEKSWorkerNodePolicy`, `AmazonEC2ContainerRegistryReadOnly`, `AmazonEKS_CNI_Policy`). - Re-apply Terraform. - Configure kubectl: `aws eks update-kubeconfig --region us-east-1 --name saffaran-eks`.

- 3. Deploy App to Kubernetes:

- Create `k8s-deployment.yaml`:

```
yaml
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: saffaran-app
    spec:
      replicas: 2
      selector:
        matchLabels:
          app: saffaran
      template:
        metadata:
          labels:
            app: saffaran
        spec:
          containers:
          - name: app
            image: your-dockerhub/saffaran-app:latest  # Push to ECR/DockerHub first
            ports:
            - containerPort: 5000
    ---
    apiVersion: v1
    kind: Service
    metadata:
      name: saffaran-service
    spec:
      selector:
        app: saffaran
      ports:
      - port: 80
        targetPort: 5000
      type: LoadBalancer  # Exposes via AWS ELB
```

- Build/push Docker image: On Jenkins EC2 or local: `docker tag saffaran-app:latest yourusername/saffaran-app:latest && docker push .` - Apply: `kubectl apply -f k8s-deployment.yaml .` - Get ELB URL: `kubectl get svc saffaran-service` → Access API via that URL.

Step 6: Set Up CI/CD with Jenkins and GitHub

- 1. Push code to GitHub: Create files (app.py, Dockerfile, etc.) in your local dir → `