**DEPLOY A SCALABLE WEB APPLICATION USING AWS INFRASTRUCTURE VIA BOTO3**

**OBJECTIVE**

Automate the deployment of a PHP user-registration web application on AWS using Python's Boto3 SDK. This script orchestrates the complete infrastructure lifecycle—from networking to compute, database provisioning, load balancing, and notifications—ensuring a secure, scalable, and highly available architecture without manual console interactions.

**Why This Project?**

1. **Infrastructure as Code**: Full end-to-end automation with Boto3 provides repeatable deployments, version control of infrastructure, and faster iteration.
2. **Scalability & Performance**: Auto Scaling Groups dynamically adjust instance counts based on real-time load, ensuring consistent performance under varying traffic.
3. **High Availability & Fault Tolerance**: Multi-AZ RDS and redundant subnets prevent single points of failure and maintain uptime during zone disruptions.
4. **Security Best Practices**: Least-privilege security groups, private RDS tier, and optional bastion host enforce strong network segmentation and controlled access.
5. **Comprehensive Coverage**: The script spans VPC creation, subnet/routing configuration, EC2 provisioning, database setup, load balancing, and notification wiring in a single codebase.

**1. CONFIGURATION SECTION**

# -------------------- CONFIGURATION --------------------

REGION = 'ap-south-1'

AMI\_ID = 'ami-0e35ddab05955cf57'

KEY\_NAME = 'akashsridhar'

ZIP\_URL = 'https://raw.githubusercontent.com/AKASH8525/new/main/php\_registration\_app.zip'

DB\_NAME = 'userdb'

DB\_USER = 'admin'

DB\_PASSWORD = 'Admin12345'

SNS\_EMAIL = 'akashak052004@gmail.com'

SNS\_TOPIC\_NAME = 'asg-notifications'

**Explanation:**

* REGION: AWS Region (e.g., Mumbai).
* AMI\_ID: Amazon Linux/Ubuntu image ID.
* KEY\_NAME: SSH key pair name.
* ZIP\_URL: PHP application source archive.
* DB\_\*: RDS MySQL database credentials.
* SNS\_\*: Email notification configuration.

**2. INITIALIZATION: LOGGING & AWS CLIENTS**

# -------------------- SETUP LOGGING --------------------

logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

# -------------------- AWS CLIENTS --------------------

ec2 = boto3.client('ec2', region\_name=REGION)

rds = boto3.client('rds', region\_name=REGION)

elb = boto3.client('elbv2', region\_name=REGION)

asg\_client = boto3.client('autoscaling', region\_name=REGION)

sns\_client = boto3.client('sns', region\_name=REGION)

# No separate CloudWatch client is needed with target tracking

**Explanation:**

Sets up logging and AWS service clients:

* **Logging Configuration:** Establishes structured log formatting and level.
* **Boto3 Clients**: Creates clients for EC2, RDS, ELBv2, Auto Scaling, and SNS, all scoped to the specified region.

**3. Resource Tracking Dictionary**

# -------------------- RESOURCE TRACKING --------------------

resources = {

    'vpc': None,

    'subnets': [],  # [0-1]: Public, [2-3]: Private (EC2), [4-5]: Private (RDS)

    'igw': None,

    'nat\_gw': None,

    'security\_groups': [],  # [0]: ALB-SG, [1]: EC2-SG, [2]: RDS-SG, [3]: Bastion-SG

    'rds': None,

    'launch\_template': None,

    'target\_group': None,

    'alb': None,

    'asg': None,

    'bastion\_instance': None,

    'sns\_topic\_arn': None

}

**Explanation:**

Prepares a central registry for deployed resources:

* **Resource Dictionary:** Initializes keys for every AWS resource (VPC, subnets, SGs, RDS, ALB, ASG, Bastion, SNS).
* **Usage:** Enables inter-module communication and simplifies cleanup.

**4. Networking: create\_network()**

def create\_network():

    try:

        logging.info("Creating VPC...")

        vpc = ec2.create\_vpc(CidrBlock='10.0.0.0/16')

        resources['vpc'] = vpc['Vpc']['VpcId']

        ec2.modify\_vpc\_attribute(VpcId=resources['vpc'], EnableDnsSupport={'Value': True})

        ec2.modify\_vpc\_attribute(VpcId=resources['vpc'], EnableDnsHostnames={'Value': True})

        ec2.create\_tags(Resources=[resources['vpc']], Tags=[{'Key': 'Name', 'Value': 'Main-VPC'}])

        logging.info(f"VPC created with ID: {resources['vpc']}")

        logging.info("Creating Internet Gateway...")

        igw = ec2.create\_internet\_gateway()

        resources['igw'] = igw['InternetGateway']['InternetGatewayId']

        ec2.attach\_internet\_gateway(InternetGatewayId=resources['igw'], VpcId=resources['vpc'])

        logging.info(f"Internet Gateway created with ID: {resources['igw']}")

        logging.info("Creating Subnets...")

        subnets = [

            ('10.0.1.0/24', f'{REGION}a', 'public-1'),

            ('10.0.2.0/24', f'{REGION}b', 'public-2'),

            ('10.0.3.0/24', f'{REGION}a', 'private-ec2-1'),

            ('10.0.4.0/24', f'{REGION}b', 'private-ec2-2'),

            ('10.0.5.0/24', f'{REGION}a', 'private-rds-1'),

            ('10.0.6.0/24', f'{REGION}b', 'private-rds-2')

        ]

        for cidr, az, name in subnets:

            subnet = ec2.create\_subnet(

                CidrBlock=cidr,

                VpcId=resources['vpc'],

                AvailabilityZone=az,

                TagSpecifications=[{

                    'ResourceType': 'subnet',

                    'Tags': [{'Key': 'Name', 'Value': name}]

                }]

            )

            resources['subnets'].append(subnet['Subnet']['SubnetId'])

            logging.info(f"Subnet {name} created with ID: {subnet['Subnet']['SubnetId']}")

        logging.info("Creating NAT Gateway...")

        eip = ec2.allocate\_address(Domain='vpc')

        nat\_gw = ec2.create\_nat\_gateway(

            SubnetId=resources['subnets'][0],

            AllocationId=eip['AllocationId'],

            TagSpecifications=[{

                'ResourceType': 'natgateway',

                'Tags': [{'Key': 'Name', 'Value': 'Main-NAT'}]

            }]

        )

        resources['nat\_gw'] = nat\_gw['NatGateway']['NatGatewayId']

        ec2.get\_waiter('nat\_gateway\_available').wait(NatGatewayIds=[resources['nat\_gw']])

        logging.info(f"NAT Gateway created with ID: {resources['nat\_gw']}")

        logging.info("Configuring Route Tables...")

        public\_rt = ec2.create\_route\_table(VpcId=resources['vpc'])['RouteTable']['RouteTableId']

        ec2.create\_route(RouteTableId=public\_rt, DestinationCidrBlock='0.0.0.0/0', GatewayId=resources['igw'])

        ec2.associate\_route\_table(RouteTableId=public\_rt, SubnetId=resources['subnets'][0])

        ec2.associate\_route\_table(RouteTableId=public\_rt, SubnetId=resources['subnets'][1])

        private\_rt = ec2.create\_route\_table(VpcId=resources['vpc'])['RouteTable']['RouteTableId']

        ec2.create\_route(RouteTableId=private\_rt, DestinationCidrBlock='0.0.0.0/0', NatGatewayId=resources['nat\_gw'])

        for subnet in resources['subnets'][2:4]:

            ec2.associate\_route\_table(RouteTableId=private\_rt, SubnetId=subnet)

        logging.info("Network setup complete!")

    except Exception as e:

        logging.error(f"Network creation failed: {str(e)}")

        exit(1)

**Explanation:**

Builds the virtual network foundation:

* **VPC Setup:** Creates a /16 VPC, enables DNS support/hostnames, and tags it for easy identification.
* **Internet Gateway:** Attaches an IGW to allow public subnet egress and ingress.
* **Subnets**: Provisions six subnets across two AZs:
  + 2 public subnets (for NAT GW and ALB)
  + 2 private subnets (for application EC2 instances)
  + 2 isolated private subnets (for RDS)
* **NAT Gateway:** Allocates an Elastic IP and deploys a NAT GW in a public subnet to grant private instances outbound internet access.
* **Route Tables:** Configures separate route tables:
  + Public route table directs 0.0.0.0/0 traffic to the IGW for public subnets.
  + Private route table directs 0.0.0.0/0 traffic to the NAT GW for private EC2 subnets.

**5. Security Groups: configure\_security\_groups()**

def create\_security\_groups():

    try:

        logging.info("Creating Security Groups...")

        alb\_sg = ec2.create\_security\_group(

            Description='ALB Security Group', GroupName='ALB-SG', VpcId=resources['vpc'],

            TagSpecifications=[{'ResourceType': 'security-group', 'Tags': [{'Key': 'Name', 'Value': 'ALB-SG'}]}]

        )['GroupId']

        ec2.authorize\_security\_group\_ingress(

            GroupId=alb\_sg,

            IpPermissions=[{'IpProtocol': 'tcp', 'FromPort': 80, 'ToPort': 80, 'IpRanges': [{'CidrIp': '0.0.0.0/0'}]}]

        )

        ec2\_sg = ec2.create\_security\_group(

            Description='EC2 Security Group', GroupName='EC2-SG', VpcId=resources['vpc'],

            TagSpecifications=[{'ResourceType': 'security-group', 'Tags': [{'Key': 'Name', 'Value': 'EC2-SG'}]}]

        )['GroupId']

        bastion\_sg = ec2.create\_security\_group(

            Description='Bastion Host Security Group', GroupName='Bastion-SG', VpcId=resources['vpc'],

            TagSpecifications=[{'ResourceType': 'security-group', 'Tags': [{'Key': 'Name', 'Value': 'Bastion-SG'}]}]

        )['GroupId']

        ec2.authorize\_security\_group\_ingress(

            GroupId=bastion\_sg,

            IpPermissions=[{'IpProtocol': 'tcp', 'FromPort': 22, 'ToPort': 22, 'IpRanges': [{'CidrIp': '0.0.0.0/0'}]}]

        )

        ec2.authorize\_security\_group\_ingress(

            GroupId=ec2\_sg,

            IpPermissions=[{'IpProtocol': 'tcp', 'FromPort': 80, 'ToPort': 80, 'UserIdGroupPairs': [{'GroupId': alb\_sg}]}]

        )

        ec2.authorize\_security\_group\_ingress(

            GroupId=ec2\_sg,

            IpPermissions=[{'IpProtocol': 'tcp', 'FromPort': 22, 'ToPort': 22, 'UserIdGroupPairs': [{'GroupId': bastion\_sg}]}]  # Allow SSH from Bastion

        )

        rds\_sg = ec2.create\_security\_group(

            Description='RDS Security Group', GroupName='RDS-SG', VpcId=resources['vpc'],

            TagSpecifications=[{'ResourceType': 'security-group', 'Tags': [{'Key': 'Name', 'Value': 'RDS-SG'}]}]

        )['GroupId']

        ec2.authorize\_security\_group\_ingress(

            GroupId=rds\_sg,

            IpPermissions=[{'IpProtocol': 'tcp', 'FromPort': 3306, 'ToPort': 3306, 'UserIdGroupPairs': [{'GroupId': ec2\_sg}]}]

        )

        resources['security\_groups'].extend([alb\_sg, ec2\_sg, rds\_sg, bastion\_sg])

        logging.info("All Security Groups created!")

    except Exception as e:

        logging.error(f"Security group creation failed: {str(e)}")

        exit(1)

**Explanation:**

Implements network security segmentation:

* **ALB Security Group:** Allows inbound HTTP on port 80 from any IPv4 address.
* **EC2 Security Group:** Allows HTTP from the ALB SG and SSH from the Bastion SG only.
* **Bastion Security Group**: Allows SSH from anywhere to serve as a hardened jump host.
* **RDS Security Group:** Restricts MySQL (port 3306) access to only the EC2 SG.
* **Security Group Tagging & Tracking**: Each SG ID is stored for use in EC2, RDS, and application deployment.

**6. RDS Provisioning: create\_rds\_instance()**

def create\_rds():

    try:

        logging.info("Creating RDS Instance...")

        rds.create\_db\_subnet\_group(

            DBSubnetGroupName='rds-subnet-group',

            DBSubnetGroupDescription='RDS subnet group',

            SubnetIds=resources['subnets'][4:6]

        )

        rds.create\_db\_instance(

            DBInstanceIdentifier='web-db',

            AllocatedStorage=20,

            DBInstanceClass='db.t3.micro',

            Engine='mysql',

            MasterUsername=DB\_USER,

            MasterUserPassword=DB\_PASSWORD,

            VpcSecurityGroupIds=[resources['security\_groups'][2]],

            DBSubnetGroupName='rds-subnet-group',

            MultiAZ=True,  # Multi-AZ enabled for high availability

            PubliclyAccessible=False,

            DBName=DB\_NAME

        )

        resources['rds'] = 'web-db'

        rds.get\_waiter('db\_instance\_available').wait(DBInstanceIdentifier='web-db')

        return rds.describe\_db\_instances(DBInstanceIdentifier='web-db')['DBInstances'][0]['Endpoint']['Address']

    except Exception as e:

        logging.error(f"RDS creation failed: {str(e)}")

        exit(1)

**Explanation:**

Provisions the database layer:

* **DB Subnet Group:** Groups the two RDS-dedicated subnets to ensure Multi-AZ placement.
* **RDS Instance:** Launches a Multi-AZ MySQL instance with 20 GB storage, private-only access, and attached to the RDS SG.
* **Endpoint Discovery:** Waits for instance readiness and retrieves the endpoint for application connectivity.
* Enforces best practices with subnet isolation and no public access.

**7. Launch Template: create\_launch\_template(db\_endpoint)**

def create\_launch\_template(db\_endpoint):

    try:

        logging.info("Creating Launch Template...")

        user\_data = f'''#!/bin/bash

sudo apt-get update -y

sudo apt-get install -y apache2 php libapache2-mod-php php-mysql unzip wget mysql-client

sudo systemctl start apache2

sudo chown -R www-data:www-data /var/www/html

cd /var/www/html

sudo rm -f index.html

sudo wget {ZIP\_URL} -O app.zip

sudo unzip -o app.zip

sudo rm app.zip

sudo bash -c "cat > /var/www/html/dbconfig.php" <<EOL

<?php

\$host = "{db\_endpoint}";

\$db  = "{DB\_NAME}";

\$user = "{DB\_USER}";

\$pass = "{DB\_PASSWORD}";

\$conn = new mysqli(\$host, \$user, \$pass, \$db);

if (\$conn->connect\_error) {{

    die("Connection failed: " . \$conn->connect\_error);

}}

?>

EOL

counter=0

until sudo mysql -h {db\_endpoint} -u {DB\_USER} -p{DB\_PASSWORD} -e "USE {DB\_NAME};" 2>/dev/null; do

    if [ $counter -ge 10 ]; then exit 1; fi

    sleep 30

    ((counter++))

done

sudo mysql -h {db\_endpoint} -u {DB\_USER} -p{DB\_PASSWORD} -D {DB\_NAME} -e "

CREATE TABLE IF NOT EXISTS users (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(100),

    email VARCHAR(100) UNIQUE,

    password VARCHAR(255),

    created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);" 2>/dev/null

sudo systemctl restart apache2

'''

        ec2.create\_launch\_template(

            LaunchTemplateName='php-app-lt',

            LaunchTemplateData={

                'ImageId': AMI\_ID,

                'InstanceType': 't2.micro',

                'KeyName': KEY\_NAME,

                'SecurityGroupIds': [resources['security\_groups'][1]],

                'UserData': base64.b64encode(user\_data.encode()).decode(),

                'TagSpecifications': [{

                    'ResourceType': 'instance',

                    'Tags': [{'Key': 'Name', 'Value': 'PHP-App'}]

                }]

            }

        )

        resources['launch\_template'] = 'php-app-lt'

        logging.info("Launch Template created.")

    except Exception as e:

        logging.error(f"Launch Template creation failed: {str(e)}")

**Explanation:**

Bootstraps and standardizes application servers:

* **User Data Script**: Installs Apache, PHP, required extensions, and MySQL client. Downloads the PHP app archive, configures dbconfig.php with the dynamic db\_endpoint, and initializes the users table.
* **Launch Template Definition:** Captures AMI, instance type, key pair, SG, and user **data in a reusable template for Auto Scaling.**

**8. Load Balancer: create\_alb()**

def create\_alb():

    try:

        logging.info("Creating ALB Resources...")

        tg = elb.create\_target\_group(

            Name='php-tg', Protocol='HTTP', Port=80, VpcId=resources['vpc'],

            HealthCheckProtocol='HTTP', HealthCheckPath='/register.php',

            HealthyThresholdCount=2, UnhealthyThresholdCount=2

        )

        resources['target\_group'] = tg['TargetGroups'][0]['TargetGroupArn']

        alb = elb.create\_load\_balancer(

            Name='php-alb', Subnets=resources['subnets'][0:2],

            SecurityGroups=[resources['security\_groups'][0]], Scheme='internet-facing'

        )['LoadBalancers'][0]

        resources['alb'] = alb['LoadBalancerArn']

        elb.get\_waiter('load\_balancer\_available').wait(LoadBalancerArns=[resources['alb']])

        elb.create\_listener(

            LoadBalancerArn=resources['alb'], Protocol='HTTP', Port=80,

            DefaultActions=[{'Type': 'forward', 'TargetGroupArn': resources['target\_group']}]

        )

        logging.info("ALB setup complete.")

    except Exception as e:

        logging.error(f"ALB creation failed: {str(e)}")

**Explanation:**

Distributes incoming traffic:

* **Target Group:** Defines the backend pool on port 80 and health checks at /register.php.
* **Application Load Balancer:** Deploys an internet-facing ALB across public subnets with the ALB SG.
* **Listener:** Routes HTTP traffic on port 80 to the target group.

**9. Notifications: create\_sns\_topic()**

def create\_sns\_topic():

    try:

        logging.info(f"Creating SNS Topic: {SNS\_TOPIC\_NAME}")

        topic = sns\_client.create\_topic(Name=SNS\_TOPIC\_NAME)

        resources['sns\_topic\_arn'] = topic['TopicArn']

        logging.info(f"SNS Topic created with ARN: {resources['sns\_topic\_arn']}")

        logging.info(f"Subscribing email {SNS\_EMAIL} to SNS Topic...")

        subscription = sns\_client.subscribe(

            TopicArn=resources['sns\_topic\_arn'],

            Protocol='email',

            Endpoint=SNS\_EMAIL

        )

        logging.info(f"Subscription initiated. Please check your email ({SNS\_EMAIL}) to confirm the subscription.")

    except Exception as e:

        logging.error(f"SNS Topic/Subscription error: {str(e)}")

        exit(1)

**Explanation:**

Enables notification workflows:

* **Topic Creation:** Creates an SNS topic for Auto Scaling and other event notifications.
* **Subscription:** Subscribes the configured email endpoint, requiring user confirmation to activate.

**10. Auto Scaling Group: create\_asg()**

def create\_asg():

    try:

        logging.info("Creating Auto Scaling Group...")

        asg\_client.create\_auto\_scaling\_group(

            AutoScalingGroupName='php-asg',

            LaunchTemplate={'LaunchTemplateName': resources['launch\_template'], 'Version': '$Latest'},

            MinSize=2,             # One instance in each private subnet

            MaxSize=4,             # Maximum capacity

            DesiredCapacity=2,     # Start with 2 instances

            VPCZoneIdentifier=','.join(resources['subnets'][2:4]),

            TargetGroupARNs=[resources['target\_group']],

            HealthCheckType='ELB',

            HealthCheckGracePeriod=300

        )

        resources['asg'] = 'php-asg'

        # Configure SNS notifications for ASG events

        asg\_client.put\_notification\_configuration(

            AutoScalingGroupName=resources['asg'],

            TopicARN=resources['sns\_topic\_arn'],

            NotificationTypes=[

                'autoscaling:EC2\_INSTANCE\_LAUNCH',

                'autoscaling:EC2\_INSTANCE\_LAUNCH\_ERROR',

                'autoscaling:EC2\_INSTANCE\_TERMINATE',

                'autoscaling:EC2\_INSTANCE\_TERMINATE\_ERROR'

            ]

        )

        logging.info("ASG created with SNS notifications.")

        # Add a target tracking scaling policy to adjust capacity based on average CPU utilization.

        asg\_client.put\_scaling\_policy(

            AutoScalingGroupName=resources['asg'],

            PolicyName='TargetTrackingPolicy',

            PolicyType='TargetTrackingScaling',

            TargetTrackingConfiguration={

                'PredefinedMetricSpecification': {

                    'PredefinedMetricType': 'ASGAverageCPUUtilization'

                },

                'TargetValue': 50.0,  # Adjust target CPU utilization percentage as needed

                'DisableScaleIn': False

            }

        )

        logging.info("Target tracking scaling policy created.")

    except Exception as e:

        logging.error(f"ASG creation failed: {str(e)}")

        exit(1)

**Explanation:**

Automates scale-out and scale-in of app servers:

* **Auto Scaling Group**: Configures min/max/desired counts (2–4 instances) in private EC2 subnets, attached to the launch template and target group.
* **Health Checks:** Utilizes ELB health checks with a grace period to ensure instances serve traffic only when healthy.
* **SNS Notifications:** Hooks ASG events (launch/terminate) into the SNS topic.
* **Target Tracking Policy**: Maintains average CPU utilization (~50%) by dynamically adjusting instance count.

**11. Bastion Host: create\_bastion()**

def create\_bastion():

    try:

        logging.info("Launching Bastion Host (explicitly assigning public IP)...")

        instance = ec2.run\_instances(

            ImageId=AMI\_ID,

            InstanceType='t2.micro',

            KeyName=KEY\_NAME,

            MinCount=1,

            MaxCount=1,

            NetworkInterfaces=[{

                'DeviceIndex': 0,

                'SubnetId': resources['subnets'][0],

                'AssociatePublicIpAddress': True,

                'Groups': [resources['security\_groups'][3]]

            }],

            TagSpecifications=[{

                'ResourceType': 'instance',

                'Tags': [{'Key': 'Name', 'Value': 'Bastion-Host'}]

            }]

        )

        instance\_id = instance['Instances'][0]['InstanceId']

        resources['bastion\_instance'] = instance\_id

        ec2.get\_waiter('instance\_running').wait(InstanceIds=[instance\_id])

        response = ec2.describe\_instances(InstanceIds=[instance\_id])

        public\_ip = response['Reservations'][0]['Instances'][0].get('PublicIpAddress', None)

        if public\_ip is None:

            logging.warning("Bastion instance did not get a public IP despite explicit assignment.")

        logging.info(f"Bastion Host running: {instance\_id} with Public IP: {public\_ip}")

        return public\_ip

    except Exception as e:

        logging.error(f"Bastion creation failed: {str(e)}")

        exit(1)

**Explanation:**

Facilitates secure admin access:

* **Instance Provisioning:** Launches a small bastion host in a public subnet with SSH enabled.
* **Public IP Assignment:** Attaches a public IP for secure SSH access to reach private resources via the bastion.

**12. Orchestration: main()**

def main():

    try:

        create\_network()

        create\_security\_groups()

        db\_endpoint = create\_rds()

        create\_launch\_template(db\_endpoint)

        create\_alb()

        create\_sns\_topic()

        create\_asg()

        bastion\_ip = create\_bastion()

        alb\_dns = elb.describe\_load\_balancers(LoadBalancerArns=[resources['alb']])['LoadBalancers'][0]['DNSName']

        logging.info(f"\n\nDEPLOYMENT SUCCESSFUL!")

        logging.info(f"Access your registration page at: http://{alb\_dns}/register.php")

        logging.info(f"Bastion Host Public IP (for SSH): {bastion\_ip}")

        logging.info("NOTE: Check your email and confirm the SNS subscription to receive Auto Scaling notifications.")

    except Exception as e:

        logging.error(f"Deployment failed: {str(e)}")

        exit(1)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Explanation:**

Drives the full deployment in sequence:

1. **Configuration → Initialization → Resource Tracking → Network → Security → RDS → Launch Template → ALB → SNS → ASG → Bastion**
2. Retrieves and logs the ALB DNS and bastion public IP for user access.
3. Handles exceptions centrally to prevent partial deployments and ensure clean failure modes.

**Conclusion**

This Boto3-based deployment script provides a fully automated, repeatable, and modular infrastructure setup for a PHP registration application on AWS. By defining discrete functions for each layer—networking, security, compute, database, load balancing, scaling, and notifications—it enables:

* **Consistent Deployments:** Infrastructure as code ensures that every run produces the same architecture without manual drift.
* **Scalability:** Auto Scaling and load balancing dynamically adapt to traffic, maintaining performance.
* **High Availability:** Multi-AZ database and redundant subnets minimize outage risks.
* **Security:** Segmented networking, least-privilege access, and bastion hosts safeguard resources.
* **Maintainability:** Modular functions and resource tracking simplify updates, debugging, and teardown.

Together, these elements form a robust, production-ready foundation that can be extended with additional components (e.g., CloudWatch alarms, IAM roles, CI/CD integration) to meet evolving requirements.

*Maintained by: Akash K*

*Github Repo : https://github.com/AKASH8525/AKASH\_*[*PROJECTS*](https://github.com/AKASH8525/AKASH_PROJECTS.git)*.git*