DevOps Project Documentation: Resilient and Scalable Web Application Deployment on AWS

1) Overview

This project designs and implements a highly available, scalable, and secure web application architecture on AWS using VPC, EFS, EC2, Auto Scaling, ALB, and Route 53, aligned with AWS Well-Architected pillars to ensure fault tolerance, elasticity, and secure user access. The solution uses multi-AZ Auto Scaling behind an Application Load Balancer, shared storage with EFS as required, and intelligent DNS routing with Route 53 health checks and failover to improve resilience.[1] [2] [3] [4] [5]

2) Architecture Summary

- Multi-AZ VPC with public and private subnets per AZ for isolation and high availability.[4]
- Internet-facing Application Load Balancer in public subnets, targeting EC2 instances in private subnets via Target Groups.[2]
- EC2 Auto Scaling group spans at least two AZs to replace unhealthy instances automatically and scale to demand.[1] [2]
- EFS mounted by application instances for shared, scalable file storage across AZs if the app requires shared state (e.g., uploads, assets).[4]
- Route 53 provides DNS for the application domain, with ALB alias records and optional health checks and DNS failover to secondary stacks or regions for resilience.[6] [3]
- Security is enforced with layered controls: security groups, least-privilege IAM, encryption in transit and at rest, logging and monitoring consistent with the Well-Architected security pillar.[4]

3) Design Phase

3.1 Requirements

- High Availability: Multi-AZ across the compute and load-balancing tiers; health checks and automated recovery at each layer.[3] [2] [1] [4]
- Scalability: Dynamic scale-out/in using CloudWatch-driven Auto Scaling policies and ALB automatic scaling capacity.[7] [1]
- Security: Strong identity foundation, traceability, encryption, and defense-in-depth network controls (SGs, NACLs), aligned with Well-Architected Security pillar.[4]

- Resilience: Replace unhealthy instances, AZ-distributed capacity, optional DNS-level failover across stacks or regions using Route 53 health checks.[6] [3] [1]

3.2 Well-Architected Alignment

- Reliability: Multi-AZ distribution; automated recovery and health-based replacement via EC2 Auto Scaling; ALB health checks; DNS failover patterns.[2] [3] [1] [4]
- Performance Efficiency: Elastic horizontal scaling; ALB scales aggressively to traffic; optional sharding ELBs for extreme throughput with DNS-based distribution.[7] [4]
- Operational Excellence: Infrastructure as code and runbooks; monitoring and alarms; iterative improvement per Well-Architected guidance.[4]
- Cost Optimization: Scale to demand; terminate excess capacity automatically.[1] [4]
- Sustainability: Efficient scaling and shared services alignment per framework pillar.[4]

4) Target Architecture Diagram (Description)

- VPC with two or more AZs.
- Public subnets: ALB with security group allowing 80/443 from Internet, forwarding only to target group.
- Private subnets: EC2 Auto Scaling group instances with app security group allowing traffic only from ALB SG; instances mount EFS via mount targets in each AZ.
- NAT Gateways in public subnets for instance outbound access (patching, package repos).
- Route 53 public hosted zone with app record (e.g., app.example.com) as alias to ALB; optional health-checked records for active-passive or multi-Region failover.[3] [6]
- IAM roles for EC2/EFS access; KMS for encryption; CloudWatch alarms drive scaling; logs shipped to CloudWatch.

This topology enables multi-AZ availability, automatic instance healing, and scale-out based on metrics, while ALB and Route 53 handle traffic steering and health-aware routing.[2] [7] [3] [1]

5) Implementation Phase

The following step-by-step guide assumes use of IaC (CloudFormation/Terraform) with necessary variables, but can be executed via console/CLI.

5.1 Create Networking

- Create VPC (e.g., 10.0.0.0/16) with at least two AZs, each having:
  - Public subnet (for ALB, NAT Gateway).
  - Private subnet (for EC2 Auto Scaling instances).
- Configure Internet Gateway, route tables, NAT Gateways for private egress, and appropriate NACLs.
- Create security groups:
  - ALB SG: allow inbound 80/443 from 0.0.0.0/0; outbound to target ports.

    - App SG: allow inbound only from ALB SG on app port (e.g., 80/443); outbound to EFS mount ports (2049) and required egress.

Rationale: Multi-AZ VPC plus security layering underpins reliability and security pillars.[4]

5.2 Configure EFS

- Create EFS file system with mount targets in each AZ used by the Auto Scaling group.
- Enable encryption at rest; enforce TLS for in-transit where applicable.
- Create EFS security group allowing NFS(2049) from App SG.

Reasoning: EFS provides shared, scalable storage across AZs for stateful artifacts if required by the application.[4]

5.3 Build Custom AMI for Auto Scaling

- Start from a hardened base AMI; install app runtime, dependencies, CloudWatch agent, and EFS utilities.
- Configure systemd/user data to mount EFS at boot and perform app bootstrap.
- Bake image with Packer or Image Builder for faster, consistent scale-out and reduced configuration drift.

Benefit: Faster instance launch and consistent deployments aligned with operational excellence.[4]

5.4 Create Auto Scaling Group (ASG)

- Define Launch Template with AMI, instance type, IAM role, user data (mount EFS, register service), security group, and subnets (private, across AZs).
- Configure desired/min/max capacity (e.g., 2 min for HA), health checks, and replacement on instance failure.
- Attach scaling policies:
    - Target tracking (CPU%/RequestCountPerTarget) or step scaling via CloudWatch alarms.
- Register the ASG with an ALB Target Group using instance or IP targets.

Outcome: EC2 Auto Scaling detects unhealthy instances and replaces them while ensuring right-sized capacity to meet demand.[1] [2]

5.5 Deploy Application Load Balancer

- Create ALB in public subnets with ALB SG; attach listeners:
    - 80→redirect to 443; 443 with TLS cert (ACM).
- Configure health checks on the Target Group to a lightweight endpoint (e.g., /health) with proper thresholds and timeouts.
- Optional: For extreme scale, consider ALB sharding with multiple ALBs behind Route 53 records to distribute load via DNS, especially for very large workloads.[7]

Rationale: ALB provides high availability and automatic scaling of the load-balancing layer, routing to healthy targets only. [2] [7]

5.6 Integrate Route 53

- Create or use a public hosted zone for the application domain.
- Create an alias A/AAAA record to the ALB DNS name.
- Optional resilience:
  - Configure Route 53 health checks targeting ALB DNS or use alias "evaluate target health," and implement DNS failover policies (active-passive or weighted/latency with health checks) to a secondary stack or Region. [5] [6] [3]

Benefit: Route 53 can route only to healthy endpoints and fail over when primary becomes unhealthy, improving resilience beyond a single stack/Region. [5] [6] [3]

6) Testing and Optimization Phase

6.1 Functional Testing

- Verify VPC routing and SG rules (no direct Internet to instances; ALB-only ingress).
- Confirm ALB health checks pass and traffic reaches app.
- Mount and read/write EFS from instances where required.

References: ALB/Target Group health checks ensure only healthy instances serve traffic. [2]

6.2 Load and Scaling Tests

- Generate synthetic load (e.g., step/ramp, spike tests) to validate:
  - ASG scale-out/in policies trigger appropriately.
  - ALB scales to accommodate request rates; observe aggressive scale-out and conservative scale-in. [7]
- Validate replacement on instance failure by terminating an instance and observing automatic recovery and AZ rebalancing. [1] [2]

Rationale: EC2 Auto Scaling provides fault tolerance and maintains availability by redistributing instances across AZs and replacing unhealthy capacity. [1] [2]

6.3 Resilience and Failover Tests

- Simulate AZ impairment: disable targets in one AZ; confirm ALB routes to healthy AZ and ASG rebalances. [2] [1]
- If configured, test Route 53 DNS failover by failing primary health checks and verifying traffic shifts to secondary endpoint/stack/Region. [6] [3] [5]

References: Route 53 health checks and DNS failover allow routing only to healthy resources across locations. [3] [5] [6]

6.4 Optimization

- Tuning ALB Target Group health check path, intervals, and thresholds to balance speed vs. false positives.[2]
- Right-size ASG policies (target metrics, cooldowns) and instance types for performance/cost.
- Consider ALB sharding via Route 53 if sustained extremely high throughput or connection counts are required.[7]

7) Security Controls

- Identity: IAM roles and least-privilege policies for EC2/EFS; separate admin and service roles.[4]
- Network: Security groups enforcing ALB→App, App→EFS, no direct public access to instances.[4]
- Encryption: ACM-managed TLS at ALB; EFS encryption at rest and encryption in transit where applicable.[4]
- Traceability: CloudTrail and CloudWatch metrics/logs; load balancer access logs; alarms for health, errors, and scaling events.[4]
- Automation: Infrastructure as code, immutable AMIs, automated patching pipelines per operational excellence.[4]

These align with Well-Architected Security and Operational Excellence pillars.[4]

8) Operations and Monitoring

- Monitoring: CloudWatch metrics for ALB (RequestCount, TargetResponseTime, HTTPCode_ELB_5XX), ASG/EC2 (CPUUtilization, StatusCheckFailed), EFS throughput/IO metrics.[1] [2] [4]
- Alarms and Actions: Drive Auto Scaling via target tracking or step policies; alarm on unhealthy host count, high 5XXs, EFS burst credits, and DNS health check status.[3] [1] [2] [4]
- Runbooks: Document procedures for scaling adjustments, failover, rollback, and incident response consistent with Well-Architected practices.[4]

9) Deliverables

9.1 Architectural Diagrams and Design Documentation

- Diagram showing VPC, subnets, ALB, ASG/EC2, EFS, Route 53 with health checks, and IAM boundaries, including traffic flow and security boundaries.[3] [2] [4]
- Rationale for multi-AZ, Auto Scaling, ALB, and Route 53 failover patterns with reference to Well-Architected pillars.[3] [1] [4]

9.2 Implementation and Configuration Guide

- Step-by-step procedures outlined in Section 5 with parameter references and IaC snippets (templates not included here), including:
    - ASG settings (min/desired/max, health checks, scaling policies).[1] [2]
    - ALB listeners, target groups, and health checks.[2]
    - Route 53 alias records and optional health check–based DNS failover policies.[5] [6] [3]
    - EFS creation and mount configuration per AZ.[4]

9.3 Performance and Optimization Report

- Load test methodology and results showing:
    - Scale-out behavior and time-to-steady-state under defined traffic patterns.
    - ALB scaling responsiveness and any observed limits; consideration of sharding if needed.[7]
    - Instance failure and AZ impairment recovery timings with Auto Scaling and ALB.[1] [2]
    - DNS failover timings and health check thresholds if implemented.[6] [5] [3]

9.4 Project Presentation

- Slides describing:
    - Objectives and Well-Architected alignment across pillars.[4]
    - Final architecture, security model, and scaling strategy.[7] [2] [1]
    - Test results, bottlenecks found, and tuning decisions.[7] [2]
    - Challenges (e.g., health check tuning, bootstrap timing, EFS performance characteristics) and solutions adopted.[2] [7] [4]
    - Future improvements (multi-Region active-active, ALB sharding, CDN, WAF) with Route 53 strategies.[8] [3] [7]

10) Optional Enhancements

- Multi-Region patterns: Active-passive with Route 53 health-checked failover; or active-active with weighted/latency records and health checks.[8] [6] [3]
- ALB/NLB sharding via Route 53 for extreme scale, distributing traffic across multiple load balancers pointing to the same or partitioned target sets.[7]
- Additional pillars: Cost and sustainability dashboards; Well-Architected reviews and improvement plans.[4]

11) Key References Used

- EC2 Auto Scaling improves fault tolerance, availability, and cost by replacing unhealthy instances and scaling to demand across AZs.[1]
- Tutorial for setting up Auto Scaling with a load balancer and health checks.[2]
- ALB scaling characteristics and when to implement sharding; Route 53 integration for distributing across multiple LBs.[7]
- Route 53 health checks and DNS failover: monitor endpoints and route only to healthy resources; design active-passive failover.[5] [6] [3]
- AWS Well-Architected Framework pillars guiding design decisions across security, reliability, performance, cost, operations, and sustainability.[4]

⁂

1. https://docs.aws.amazon.com/autoscaling/ec2/userguide/auto-scaling-benefits.html
2. https://docs.aws.amazon.com/autoscaling/ec2/userguide/tutorial-ec2-auto-scaling-load-balancer.html
3. https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html
4. https://docs.aws.amazon.com/wellarchitected/latest/framework/the-pillars-of-the-framework.html
5. https://tutorialsdojo.com/elb-health-checks-vs-route-53-health-checks-for-target-health-monitoring/
6. https://www.stormit.cloud/blog/route-53-health-check/
7. https://aws.amazon.com/blogs/networking-and-content-delivery/scaling-strategies-for-elastic-load-balancing/
8. https://aws.amazon.com/blogs/architecture/category/networking-content-delivery/elastic-load-balancing/