Technical Design Document: Enterprise Hybrid Architecture

Target SLA: 99.9% | Hybrid Cloud | Disaster Recovery Enabled

Version: 1.0 | Status: Draft

Table of Contents

1. Infrastructure Design

1.1 Network Segmentation & DMZ

The architecture utilizes a dual-tier network design across On-Premise and Cloud (AWS).
- **On-Prem:** VLAN segmentation separates the Application Layer from the Data Layer. A DMZ hosts the Load Balancer and API Gateway to filter external traffic (Citizen Portal/External Systems).
- **Cloud:** A multi-tier VPC design is implemented. Public subnets host NAT Gateways and ALBs; Private subnets host the Application Layer (EKS/Fargate); Isolated subnets house the Data Layer (RDS/ElastiCache).

1.2 Container Orchestration

The workload is containerized using Docker and managed via a hybrid Kubernetes strategy:
- **Cloud:** Amazon EKS (Managed Kubernetes) for automated scaling and lifecycle management.
- **On-Prem:** Red Hat OpenShift or Rancher-managed Kubernetes for consistency with the cloud environment.
- **Deployment:** Applications (Django/FastAPI) and Background workers (Celery) are deployed as separate Deployment objects with Horizontal Pod Autoscalers (HPA).

1.3 Database & Storage Strategy

- **Database:** PostgreSQL (RDS in Cloud) with Multi-AZ deployment for failover. Synchronous replication is used within the primary region, while asynchronous replication is used for the DR site.
- **Caching:** Redis (Amazon ElastiCache) Cluster Mode enabled for low-latency session and data caching.
- **Storage:** Amazon S3 for cloud-native document storage with Cross-Region Replication (CRR). On-prem storage utilizes an S3-compatible object store (e.g., MinIO or NetApp) for local access and sync.

1.4 Load Balancing & Firewall

A Global Server Load Balancer (GSLB) routes traffic between On-Prem and Cloud based on latency/availability. Local Traffic Managers (LTM/ALB) distribute requests across pods.
- **Firewalls:** AWS WAF is deployed at the edge to mitigate SQLi and XSS. Security Groups and NACLs provide defense-in-depth within the VPC.

2. High Availability (HA) Design

The system target is a 99.9% SLA, which requires a redundant, self-healing architecture.

- **Stateless Applications:** All Django/FastAPI nodes are stateless. State is managed in Redis/DB. If a node fails, the Load Balancer health check removes it, and K8s spawns a replacement.
- **Multi-AZ Strategy:** Cloud resources are distributed across at least three Availability Zones. A failure of one AZ does not impact service availability.
- **Database High Availability:** RDS Multi-AZ ensures a standby instance is ready to take over within seconds. DNS flip is handled automatically by AWS.
- **Redis HA:** ElastiCache clusters operate with primary and replica nodes across AZs with automated failover.
- **Clustering:** Celery workers are clustered and utilize a distributed broker (Redis) ensuring that background jobs are retry-capable and balanced.

3. Disaster Recovery (DR) Design

3.1 DR Parameters

Based on the diagram's dedicated DR Site, the following parameters are defined:

- **RPO (Recovery Point Objective):** < 15 minutes (ensured by asynchronous database replication).
- **RTO (Recovery Time Objective):** < 4 hours (automated failover orchestration).
- **Strategy:** Active-Passive (Warm Standby). The DR site maintains a minimal pilot light for the Application layer and full read-replicas for data.

3.2 Failover & Recovery

- **Data Sync:** Cross-region replication for S3 and RDS Async replication to the DR region.
- **Failover Process:** Upon detection of primary region/site failure, the 'Failover Orchestration' system triggers: 1) Promotion of RDS Read-Replica to Primary. 2) Update of DNS records (Route53/Internal GSLB). 3) Scaling up of standby Celery and Backend pods.
- **Testing:** Formal DR drills performed semi-annually including a full workload flip to the DR site.

4. Security Architecture

4.1 Identity & Access Management

- **IAM Model:** Using AWS IAM Roles for Service Accounts (IRSA) in Kubernetes to ensure pods have the least privilege required to access AWS resources (S3/RDS).
- **RBAC:** Fine-grained Role-Based Access Control implemented within the application and the K8s cluster.

4.2 Data Protection

- **Encryption in Transit:** TLS 1.3 for all public and internal communications (Service Mesh/mTLS recommended).
- **Encryption at Rest:** AES-256 encryption using AWS KMS for RDS, S3, and EBS volumes.
- **Secrets Management:** Integration with AWS Secrets Manager or HashiCorp Vault to prevent hardcoded credentials.

4.3 Zero Trust & Compliance

The architecture assumes no perimeter-based trust. All connections require authentication and authorization. Audit logs are aggregated by the SOC (Security Operations Center) via CloudTrail and VPC Flow Logs.

5. CI/CD Pipeline Design

The diagram indicates GitHub based CI/CD integrated across environments.

- **Branching Strategy:** GitFlow model (Feature -> Develop -> Main).
- **Pipeline:** GitHub Actions triggers on push. Phases include: Build -> Lint/Test -> Security Scan (Trivy/Snyk) -> Image Push to ECR -> Deploy.
- **Deployment Strategy:** Rolling updates for dev; Canary or Blue-Green for production to minimize risk during releases.
- **IaC:** Terraform manages all cloud infrastructure (VPC, RDS, EKS). Ansible handles on-prem OS-level configuration.

6. Monitoring & Observability

A 'Unified Monitoring' approach as seen in the diagram ensures situational awareness across Hybrid and DR sites.

- **Metrics:** Prometheus for time-series data; Grafana for visualization dashboards.
- **Logs:** ELK Stack (Elasticsearch, Logstash, Kibana) for centralized log aggregation and searching.
- **Tracing:** Jaeger or AWS X-Ray for distributed tracing across Django, FastAPI, and Celery jobs to identify bottlenecks.
- **Alerting:** PagerDuty integration for critical threshold breaches (e.g., CPU > 80%, Request Error Rate > 1%).
- **SLA Reporting:** Automated uptime checks via Synthetics (Route 53 or Datadog) to verify the 99.9% target.

7. Capacity Planning

Assumption: 1,000 Concurrent Users with a growth model to 10k over 5 years.

- **Compute:** 10 pods of Backend (0.5 vCPU, 1GB RAM each) per 500 concurrent users. Initial cloud footprint: 2x m5.large instances per AZ.
- **Database:** 1x db.m5.xlarge for RDS (4 vCPU, 16GB RAM) to handle concurrent connections/IOPS.
- **Storage:** Initial 1TB, scaling dynamically at ~200GB/year.
- **Scaling Strategy:** Horizontal Scaling (HPA) for application pods based on CPU/RAM metrics. Database vertical scaling (RDS instance resizing) for larger load spikes.

8. Cost Estimation (Cloud Context)

Monthly estimated cost for Cloud Environment (Excludes On-Prem licenses):

- **Compute (EKS/EC2):** ~$1,200 (6 nodes + Cluster management)
- **Database (RDS Multi-AZ):** ~$800
- **Storage (S3 + Data Transfer):** ~$400
- **Network (NAT, WAF, ALB):** ~$350
- **DR Replication Cost:** ~$600 (Storage/Data Transfer/Standby nodes)
- **Total Estimated:** ~$3,350 / Month

9. Document Integrity & Constraints

- Must support 5-year scalability (Horizontal/Vertical hybrid).
- Must meet 99.9% SLA (Redundant components).
- Must support Audit compliance (ELK/CloudTrail logged to SOC).
- Must ensure data protection (Encryption + Multi-AZ backups).
- Must support hybrid connectivity (AWS Direct Connect / Site-to-Site VPN).
- No Single Point of Failure (Everything Multi-AZ/Clustered).