GoPhish Technical Architecture

Table of Contents

- Overview
- Application Architecture
- AWS Architecture
- Azure Architecture
- Component Details
- Security Architecture
- High Availability Architecture
- Network Architecture
- Database Architecture
- Scaling Considerations

Overview

This document provides comprehensive technical architecture guidance for deploying GoPhish in various environments, with specific focus on AWS and Azure cloud platforms. The architectures presented are designed to be secure, scalable, and maintainable.

Architecture Principles

1. **Security First:** All architectures prioritize security and isolation
2. **High Availability:** Critical components are redundant
3. **Scalability:** Horizontal scaling where appropriate
4. **Cost-Efficiency:** Balance performance with cost
5. **Maintainability:** Simple operations and monitoring

Application Architecture

System Components

```
graph TB
    subgraph "External Users"
        Admin[Security Admin]
        Target[Phishing Targets]
    end

    subgraph "Gophish Application"
        subgraph "Admin Layer"
            AdminUI[Admin Web UI]
            API[REST API]
            Auth[Authentication]
        end

        subgraph "Phishing Layer"
            PhishServer[Phishing Server]
            Landing[Landing Pages]
            Tracking[Tracking Engine]
        end

        subgraph "Processing Layer"
            Worker[Background Worker]
            Mailer[Email Sender]
            IMAP[IMAP Monitor]
        end

        subgraph "Data Layer"
            DB[(Database)]
            Sessions[Session Store]
        end
    end

    subgraph "External Services"
        SMTP[SMTP Server]
        Webhook[Webhook Endpoints]
        ImapServer[IMAP Server]
    end

    Admin -->|HTTPS :3333| AdminUI
    Admin -->|HTTPS :3333| API
    Target -->|HTTP/HTTPS :80/443| PhishServer

    AdminUI --> Auth
    API --> Auth
    Auth --> Sessions
```

```
    AdminUI --> DB
    API --> DB
    PhishServer --> DB
    Landing --> DB
    Tracking --> DB

    Worker --> DB
    Mailer --> SMTP
    IMAP --> ImapServer

    PhishServer --> Landing
    PhishServer --> Tracking

    Worker --> Mailer
    Worker --> IMAP

    API --> Webhook

    style Admin fill:#e1f5ff
    style Target fill:#fff3e0
    style DB fill:#f3e5f5
    style SMTP fill:#e8f5e9
    style AdminUI fill:#bbdefb
    style PhishServer fill:#ffccbc
```

Data Flow Diagram

```
sequenceDiagram
    participant Admin as Security Admin
    participant UI as Admin Interface
    participant API as Gophish API
    participant DB as Database
    participant Worker as Background Worker
    participant SMTP as SMTP Server
    participant Target as Target User
    participant Phish as Phish Server

    Admin->>UI: Create Campaign
    UI->>API: POST /api/campaigns
    API->>DB: Store Campaign
    API->>Worker: Queue Campaign
```

```
    Worker->>DB: Get Targets
    loop For Each Target
        Worker->>SMTP: Send Email
        SMTP->>Target: Phishing Email
    end

    Target->>Target: Opens Email
    Target->>Phish: GET /track?rid=xxx (tracking pixel)
    Phish->>DB: Log "Email Opened" Event

    Target->>Phish: GET /?rid=xxx (click link)
    Phish->>DB: Log "Link Clicked" Event
    Phish->>Target: Render Landing Page

    Target->>Phish: POST / (submit credentials)
    Phish->>DB: Log "Data Submitted" Event
    Phish->>Target: Redirect to legitimate site

    Admin->>UI: View Results
    UI->>API: GET /api/campaigns/1/results
    API->>DB: Query Results
    DB->>API: Return Results
    API->>UI: Display Results
    UI->>Admin: Show Dashboard
```

AWS Architecture

Single Region Deployment (Recommended for Most Use Cases)

```
graph TB
    subgraph "AWS Cloud - us-east-1"
        subgraph "VPC - 10.0.0.0/16"
            subgraph "Public Subnet 1 - AZ1 - 10.0.1.0/24"
                ALB1[Application Load Balancer]
                NAT1[NAT Gateway]
            end

            subgraph "Public Subnet 2 - AZ2 - 10.0.2.0/24"
                ALB2[ALB - Standby]
                NAT2[NAT Gateway]
            end
```

```
    subgraph "Private Subnet 1 - AZ1 - 10.0.10.0/24"
        Admin1[Admin Server<br/>EC2 t3.medium]
        Phish1[Phish Server<br/>EC2 t3.small]
    end

    subgraph "Private Subnet 2 - AZ2 - 10.0.11.0/24"
        Admin2[Admin Server<br/>EC2 t3.medium]
        Phish2[Phish Server<br/>EC2 t3.small]
    end

    subgraph "Database Subnet 1 - AZ1 - 10.0.20.0/24"
        RDS1[(RDS MySQL<br/>Primary)]
    end

    subgraph "Database Subnet 2 - AZ2 - 10.0.21.0/24"
        RDS2[(RDS MySQL<br/>Standby)]
    end
end

R53[Route 53<br/>DNS]
S3[S3 Bucket<br/>Static Assets]
SES[Amazon SES<br/>Email Sending]
CW[CloudWatch<br/>Monitoring]
Secrets[Secrets Manager<br/>Credentials]

R53 -->|DNS Resolution| ALB1
ALB1 --> Admin1
ALB1 --> Admin2
ALB1 --> Phish1
ALB1 --> Phish2

Admin1 --> RDS1
Admin2 --> RDS1
Phish1 --> RDS1
Phish2 --> RDS1

RDS1 -.->|Replication| RDS2

Admin1 --> S3
Admin2 --> S3
Admin1 --> SES
Admin2 --> SES
Admin1 --> Secrets
```

```
        Admin2 --> Secrets
        Phish1 --> Secrets
        Phish2 --> Secrets

        Admin1 --> CW
        Admin2 --> CW
        Phish1 --> CW
        Phish2 --> CW
        RDS1 --> CW

        Admin1 --> NAT1
        Admin2 --> NAT2
    end

    Internet((Internet))
    AdminUser[Security Admin]
    TargetUser[Phishing Target]

    Internet --> R53
    AdminUser -.->|HTTPS :443| Internet
    TargetUser -.->|HTTP/HTTPS| Internet

    style RDS1 fill:#f9d5e5
    style RDS2 fill:#f9d5e5
    style SES fill:#eeeeee
    style S3 fill:#eeeeee
    style Admin1 fill:#bbdefb
    style Admin2 fill:#bbdefb
    style Phish1 fill:#ffccbc
    style Phish2 fill:#ffccbc
```

AWS Architecture - Detailed Component Description

Networking:

- **VPC:** Isolated virtual network (10.0.0.0/16)
- **Public Subnets:** Host ALB and NAT Gateways (10.0.1.0/24, 10.0.2.0/24)
- **Private Subnets:** Host application servers (10.0.10.0/24, 10.0.11.0/24)
- **Database Subnets:** Host RDS instances (10.0.20.0/24, 10.0.21.0/24)
- **Multi-AZ:** High availability across 2 Availability Zones

Compute:

- **Admin Servers:** t3.medium EC2 instances (2 vCPU, 4GB RAM)

- **Phish Servers:** t3.small EC2 instances (2 vCPU, 2GB RAM)
- **Auto Scaling:** Optional Auto Scaling Groups for dynamic scaling
- **AMI:** Custom AMI with Gophish pre-installed

Load Balancing:

- **Application Load Balancer:** Layer 7 load balancing
  - Listener 1: Port 443 → Admin Servers (Target Group 1)
  - Listener 2: Port 80/443 → Phish Servers (Target Group 2)
- **Health Checks:** HTTP health checks every 30 seconds
- **Sticky Sessions:** Session affinity for admin interface

Database:

- **RDS MySQL:** Managed database service
  - Instance: db.t3.medium (2 vCPU, 4GB RAM)
  - Storage: 100GB GP3 SSD
  - Multi-AZ: Automatic failover to standby
  - Backup: Daily automated backups, 7-day retention
  - Encryption: At rest and in transit

Email:

- **Amazon SES:** Managed email sending service
  - High deliverability
  - Bounce and complaint handling
  - DKIM/SPF configuration
  - Sending limits: 50,000 emails/day (adjustable)

Storage:

- **S3 Bucket:** Static assets, templates, exports
  - Encryption: Server-side encryption (SSE-S3)
  - Versioning: Enabled
  - Lifecycle: Expire old exports after 90 days

Security:

- **Secrets Manager:** Store database credentials, API keys
- **Security Groups:**
  - ALB: Allow 80, 443 from 0.0.0.0/0
  - Admin: Allow 3333 from ALB only
  - Phish: Allow 80 from ALB only
  - RDS: Allow 3306 from Admin/Phish only
- **IAM Roles:** EC2 instances use IAM roles (no keys)

- **Network ACLs:** Additional network layer security

Monitoring:

- **CloudWatch:** Metrics, logs, alarms
  - CPU, memory, disk, network metrics
  - Application logs
  - RDS performance insights
  - Custom metrics (campaigns, emails sent)
- **CloudWatch Alarms:** Alert on anomalies
- **SNS:** Notifications to administrators

DNS:

- **Route 53:** DNS management
  - A record: gophish.example.com → ALB
  - A record: phish.example.com → ALB
  - Health checks: Monitor endpoint availability

AWS Architecture - Small/Medium Deployment (Cost-Optimized)

```
graph TB
    subgraph "AWS Cloud - us-east-1"
        subgraph "VPC - 10.0.0.0/16"
            subgraph "Public Subnet - 10.0.1.0/24"
                ALB[Application Load Balancer]
            end

            subgraph "Private Subnet - 10.0.10.0/24"
                EC2[Gophish All-in-One<br/>EC2 t3.small<br/>Admin + Phish
            end
        end

        R53[Route 53]
        SES[Amazon SES]
        CW[CloudWatch]

        R53 --> ALB
        ALB --> EC2
        EC2 --> SES
        EC2 --> CW
    end
```

```
    Internet((Internet))
    Users[Users]

    Internet --> R53
    Users --> Internet

    style EC2 fill:#90caf9
```

Specifications:

- **Single EC2 instance:** t3.small (2 vCPU, 2GB RAM)
- **SQLite database:** Local storage
- **Cost:** ~$20-30/month
- **Suitable for:** Up to 2,000 users

AWS Architecture - Enterprise HA Deployment

```
graph TB
    subgraph "AWS Cloud"
        subgraph "Region 1 - us-east-1 PRIMARY"
            R53_1[Route 53<br/>Failover Primary]

            subgraph "VPC 1"
                ALB1[ALB]
                ASG1[Auto Scaling Group<br/>Admin Servers 2-10]
                ASG2[Auto Scaling Group<br/>Phish Servers 2-10]
                RDS1[(RDS Multi-AZ<br/>Primary)]
                ElastiCache1[ElastiCache Redis<br/>Session Store]
            end
        end

        subgraph "Region 2 - us-west-2 DR"
            R53_2[Route 53<br/>Failover Secondary]

            subgraph "VPC 2"
                ALB2[ALB]
                ASG3[Auto Scaling Group<br/>Admin Servers]
                ASG4[Auto Scaling Group<br/>Phish Servers]
                RDS2[(RDS Read Replica<br/>Promoted on Failover)]
                ElastiCache2[ElastiCache Redis]
            end
        end
```

```
        S3[S3 Bucket<br/>Cross-Region Replication]

        RDS1 -.->|Cross-Region Replication| RDS2
        ElastiCache1 -.->|Global Datastore| ElastiCache2

        R53_1 --> ALB1
        R53_2 --> ALB2
        ALB1 --> ASG1
        ALB1 --> ASG2
        ALB2 --> ASG3
        ALB2 --> ASG4

        ASG1 --> RDS1
        ASG2 --> RDS1
        ASG1 --> ElastiCache1
        ASG3 --> RDS2
        ASG4 --> RDS2
        ASG3 --> ElastiCache2

        ASG1 --> S3
        ASG3 --> S3
    end

    Internet((Internet))
    Internet --> R53_1

    style RDS1 fill:#f48fb1
    style RDS2 fill:#f48fb1
    style ElastiCache1 fill:#ce93d8
    style ElastiCache2 fill:#ce93d8
```

Features:

- **Multi-Region:** Primary and DR regions
- **Auto Scaling:** Dynamic scaling based on load
- **Session Store:** Redis for distributed sessions
- **Cross-Region Replication:** Database and S3
- **Automatic Failover:** Route 53 health check based
- **Cost:** ~$1,000-3,000/month
- **Suitable for:** 50,000+ users, mission-critical deployments

---

Azure Architecture

Single Region Deployment (Recommended)

```
graph TB
    subgraph "Azure Cloud - East US"
        subgraph "Resource Group - gophish-prod"
            subgraph "Virtual Network - 10.0.0.0/16"
                subgraph "Subnet - Application - 10.0.1.0/24"
                    AGW[Application Gateway<br/>WAF Enabled]
                end

                subgraph "Subnet - Admin - 10.0.10.0/24"
                    VMSS1[VM Scale Set<br/>Admin Servers<br/>Standard_B2s
                end

                subgraph "Subnet - Phish - 10.0.11.0/24"
                    VMSS2[VM Scale Set<br/>Phish Servers<br/>Standard_B2s
                end

                subgraph "Subnet - Database - 10.0.20.0/24"
                    MySQL[(Azure Database<br/>for MySQL<br/>Flexible Serv
                end
            end

            DNS[Azure DNS<br/>gophish.example.com]
            Storage[Azure Blob Storage<br/>Static Assets]
            KV[Key Vault<br/>Secrets]
            Monitor[Azure Monitor<br/>Log Analytics]
            NSG[Network Security Groups]
        end

        SendGrid[SendGrid<br/>Email Service]
    end

    Internet((Internet))
    AdminUser[Security Admin]
    TargetUser[Phishing Target]

    Internet --> DNS
    DNS --> AGW
    AGW --> VMSS1
```

```
AGW --> VMSS2

VMSS1 --> MySQL
VMSS2 --> MySQL
VMSS1 --> Storage
VMSS1 --> KV
VMSS2 --> KV
VMSS1 --> SendGrid

VMSS1 --> Monitor
VMSS2 --> Monitor
MySQL --> Monitor

NSG -.->|Firewall Rules| VMSS1
NSG -.->|Firewall Rules| VMSS2
NSG -.->|Firewall Rules| MySQL

AdminUser --> Internet
TargetUser --> Internet

style MySQL fill:#f9d5e5
style Storage fill:#e1bee7
style VMSS1 fill:#bbdefb
style VMSS2 fill:#ffccbc
```

Azure Architecture - Detailed Component Description

Networking:

- **Virtual Network:** Isolated network (10.0.0.0/16)
- **Subnets:**
  - Application Gateway: 10.0.1.0/24
  - Admin Servers: 10.0.10.0/24
  - Phish Servers: 10.0.11.0/24
  - Database: 10.0.20.0/24
- **Network Security Groups:** Subnet-level firewall rules
- **Azure Firewall:** Optional for enhanced security

Compute:

- **VM Scale Sets:** Auto-scaling virtual machine groups
  - Admin VMSS: Standard_B2s (2 vCPU, 4GB RAM)
  - Phish VMSS: Standard_B2s (2 vCPU, 4GB RAM)

    - Min instances: 2
    - Max instances: 10
    - Scale based on CPU/memory/custom metrics

Load Balancing:

- **Application Gateway:** Layer 7 load balancer
    - WAF enabled (Web Application Firewall)
    - SSL termination
    - Path-based routing:
        - /admin → Admin VMSS
        - /* → Phish VMSS
    - Health probes
    - Session affinity (cookie-based)

Database:

- **Azure Database for MySQL:** Managed PaaS database
    - Flexible Server tier
    - General Purpose: 2 vCores, 8GB RAM
    - Storage: 100GB SSD
    - Backup: Automated daily backups, 7-day retention
    - High Availability: Zone-redundant HA
    - Encryption: At rest and in transit
    - Firewall: Allow Azure services only

Email:

- **SendGrid:** Integrated email service
    - Azure Marketplace integration
    - DKIM/SPF configuration
    - Bounce and complaint handling
    - Alternative: Use custom SMTP

Storage:

- **Azure Blob Storage:** Object storage
    - Hot tier for active data
    - Container: gophish-assets (private)
    - Encryption: Microsoft-managed keys
    - Lifecycle management: Archive old data

Security:

- **Key Vault:** Secure secret storage

    - Database credentials
    - API keys
    - Certificates
    - Managed identities for VM access (no credentials)
- **NSGs:** Network security rules
    - App Gateway: Allow 80, 443 from Internet
    - Admin VMSS: Allow 3333 from App Gateway
    - Phish VMSS: Allow 80 from App Gateway
    - MySQL: Allow 3306 from VMSS only
- **Azure AD:** Identity and access management
- **RBAC:** Role-based access control for Azure resources

Monitoring:

- **Azure Monitor:** Comprehensive monitoring
    - Metrics: CPU, memory, disk, network
    - Logs: Application and system logs
    - Alerts: Automated alerts on thresholds
    - Dashboards: Custom monitoring dashboards
- **Application Insights:** Application performance monitoring
- **Log Analytics:** Centralized log management

DNS:

- **Azure DNS:** DNS hosting
    - A record: gophish.example.com → Application Gateway IP
    - CNAME: phish.example.com → gophish.example.com
    - TTL: 300 seconds

Backup & DR:

- **Azure Backup:** VM and database backups
- **Geo-Redundant Storage:** Cross-region replication
- **Azure Site Recovery:** Disaster recovery orchestration

Azure Architecture - Small/Medium Deployment (Cost-Optimized)

```
graph TB
    subgraph "Azure Cloud - East US"
        subgraph "Resource Group"
            VNet[Virtual Network]
            VM[Single VM<br/>Standard_B2s<br/>All-in-One Server]
            MySQL[(Azure Database<br/>for MySQL<br/>Basic Tier)]
```

```
            NSG[Network Security Group]
        end

        DNS[Azure DNS]

        DNS --> VM
        VM --> MySQL
        NSG --> VM
    end

    Internet((Internet))
    Internet --> DNS

    style VM fill:#90caf9
    style MySQL fill:#f9d5e5
```

Specifications:

- **Single VM:** Standard_B2s (2 vCPU, 4GB RAM)
- **Database:** Azure Database for MySQL - Basic tier
- **Cost:** ~$50-80/month
- **Suitable for:** Up to 2,000 users

Azure Architecture - Enterprise HA Deployment

```
graph TB
    subgraph "Azure"
        TM[Traffic Manager<br/>Priority Routing]

        subgraph "Primary Region - East US"
            subgraph "Availability Zone 1"
                AGW1[App Gateway]
                VMSS1[Admin VMSS]
                VMSS2[Phish VMSS]
            end

            subgraph "Availability Zone 2"
                AGW1_AZ2[App Gateway]
                VMSS1_AZ2[Admin VMSS]
                VMSS2_AZ2[Phish VMSS]
            end
```

```
        MySQL1[(Azure MySQL<br/>Zone-Redundant HA)]
        Redis1[Azure Cache<br/>for Redis]
    end

    subgraph "DR Region - West US"
        AGW2[App Gateway]
        VMSS3[Admin VMSS]
        VMSS4[Phish VMSS]
        MySQL2[(MySQL<br/>Read Replica)]
        Redis2[Redis Replica]
    end

    Storage[Geo-Redundant<br/>Storage Account]

    TM --> AGW1
    TM -.->|Failover| AGW2

    AGW1 --> VMSS1
    AGW1 --> VMSS2
    AGW1_AZ2 --> VMSS1_AZ2
    AGW1_AZ2 --> VMSS2_AZ2

    VMSS1 --> MySQL1
    VMSS2 --> MySQL1
    VMSS1 --> Redis1

    AGW2 --> VMSS3
    AGW2 --> VMSS4
    VMSS3 --> MySQL2
    VMSS4 --> MySQL2
    VMSS3 --> Redis2

    MySQL1 -.->|Geo-Replication| MySQL2
    Redis1 -.->|Geo-Replication| Redis2

    VMSS1 --> Storage
    VMSS3 --> Storage
end

style MySQL1 fill:#f48fb1
style MySQL2 fill:#f48fb1
style Redis1 fill:#ce93d8
style Redis2 fill:#ce93d8
```

Features:

- **Multi-Region:** Primary (East US) and DR (West US)
- **Availability Zones:** Within primary region
- **Traffic Manager:** Global load balancing and failover
- **Zone-Redundant HA:** Database and cache
- **Geo-Replication:** Cross-region data replication
- **Cost:** ~$1,500-4,000/month
- **Suitable for:** 50,000+ users, enterprise deployments

Component Details

Admin Server Component

Purpose: Provides administrative interface and API

Specifications:

- **CPU:** 2 cores minimum, 4 cores recommended
- **Memory:** 4GB minimum, 8GB recommended
- **Storage:** 20GB OS + database storage
- **Network:** Private subnet, load balancer access only

Software:

- **OS:** Ubuntu 20.04 LTS or Amazon Linux 2
- **Runtime:** Gophish binary (no dependencies)
- **Service:** systemd service for auto-start
- **Reverse Proxy:** Optional Nginx for additional features

Configuration:

```
{
  "admin_server": {
    "listen_url": "0.0.0.0:3333",
    "use_tls": true,
    "cert_path": "/etc/gophish/cert.pem",
    "key_path": "/etc/gophish/key.pem"
  },
  "phish_server": {
    "listen_url": "0.0.0.0:80"
  },
```

```
    "db_name": "mysql",
    "db_path": "user:pass@tcp(rds-endpoint:3306)/gophish"
}
```

Health Check:

- **Endpoint:** `GET /health`
- **Interval:** 30 seconds
- **Timeout:** 5 seconds
- **Healthy Threshold:** 2 consecutive successes
- **Unhealthy Threshold:** 3 consecutive failures

Phish Server Component

Purpose: Serves landing pages and tracks interactions

Specifications:

- **CPU:** 1-2 cores
- **Memory:** 2GB minimum, 4GB recommended
- **Storage:** 10GB OS
- **Network:** Public subnet (via load balancer)

Software:

- **OS:** Ubuntu 20.04 LTS or Amazon Linux 2
- **Runtime:** Gophish binary (phish mode only)

Configuration:

```
./gophish --mode phish --config /etc/gophish/config.json
```

Scaling:

- Horizontal scaling: Add more phish servers behind load balancer
- Stateless: No session state, fully scalable
- Connection draining: 300 seconds for graceful shutdown

Database Component

Schema Overview:

Core Tables:

1. **users** - Admin users and authentication
2. **campaigns** - Campaign definitions
3. **templates** - Email templates
4. **pages** - Landing pages
5. **groups** - Target groups
6. **targets** - Individual recipients
7. **results** - Campaign results per recipient
8. **events** - Detailed event log
9. **smtp** - SMTP sending profiles

Performance Tuning:

MySQL Configuration:

```
[mysqld]
innodb_buffer_pool_size = 2G
innodb_log_file_size = 512M
max_connections = 500
query_cache_size = 64M
```

Indexes:

- `campaigns(user_id, created_date)`
- `results(campaign_id, status)`
- `events(campaign_id, time)`
- `targets(group_id, email)`

Backup Strategy:

- **Automated Backups:** Daily at 2 AM UTC
- **Retention:** 7 days point-in-time recovery
- **Backup Validation:** Weekly restore test
- **Export Backups:** Monthly full export to S3/Blob Storage

Security Architecture

Network Security Architecture

```
graph TB
    subgraph "Internet Boundary"
        Internet((Internet))
        WAF[Web Application Firewall]
    end

    subgraph "DMZ - Public Subnet"
        LB[Load Balancer<br/>SSL Termination]
        Bastion[Bastion Host<br/>Jump Box]
    end

    subgraph "Application Tier - Private Subnet"
        Admin[Admin Servers]
        Phish[Phish Servers]
    end

    subgraph "Data Tier - Private Subnet"
        DB[(Database)]
        Cache[(Cache)]
    end

    subgraph "Management"
        VPN[VPN Gateway]
        Monitoring[Monitoring]
    end

    Internet --> WAF
    WAF --> LB
    LB --> Admin
    LB --> Phish
    Admin --> DB
    Phish --> DB
    Admin --> Cache

    VPN -.->|SSH| Bastion
    Bastion -.->|SSH| Admin
    Bastion -.->|SSH| Phish

    Admin --> Monitoring
    Phish --> Monitoring
    DB --> Monitoring

    style Internet fill:#ffcdd2
    style WAF fill:#fff9c4
```

```
    style DB fill:#c8e6c9
    style Cache fill:#c8e6c9
```

Security Layers

Layer 1: Network Security

- VPC/VNet isolation
- Private subnets for application and database
- Security groups/NSGs with least privilege
- No direct Internet access to application servers
- NAT Gateway for outbound traffic only

Layer 2: Application Security

- TLS 1.2+ for all connections
- Certificate-based authentication
- API key rotation
- Rate limiting
- Input validation and sanitization

Layer 3: Data Security

- Encryption at rest (database and storage)
- Encryption in transit (TLS)
- Secure password hashing (bcrypt)
- Database firewall rules
- Secrets management (AWS Secrets Manager / Azure Key Vault)

Layer 4: Access Control

- Role-based access control (RBAC)
- Multi-factor authentication (planned)
- Principle of least privilege
- Audit logging
- Session management

Layer 5: Monitoring & Response

- Security event logging
- Anomaly detection
- Intrusion detection
- Automated alerting
- Incident response procedures

Security Group / NSG Rules

Admin Server Security Group:

```
Inbound:
- Allow TCP 3333 from Load Balancer Security Group
- Allow TCP 22 from Bastion Host (management only)

Outbound:
- Allow TCP 3306 to Database Security Group
- Allow TCP 587 to 0.0.0.0/0 (SMTP)
- Allow TCP 443 to 0.0.0.0/0 (webhooks, updates)
```

Phish Server Security Group:

```
Inbound:
- Allow TCP 80 from Load Balancer Security Group
- Allow TCP 22 from Bastion Host (management only)

Outbound:
- Allow TCP 3306 to Database Security Group
- Allow TCP 443 to 0.0.0.0/0 (webhooks)
```

Database Security Group:

```
Inbound:
- Allow TCP 3306 from Admin Server Security Group
- Allow TCP 3306 from Phish Server Security Group

Outbound:
- Deny all (database shouldn't initiate outbound)
```

Load Balancer Security Group:

```
Inbound:
- Allow TCP 443 from 0.0.0.0/0 (admin interface)
- Allow TCP 80 from 0.0.0.0/0 (phishing pages)
- Allow TCP 443 from 0.0.0.0/0 (phishing pages, optional)

Outbound:
- Allow TCP 3333 to Admin Server Security Group
- Allow TCP 80 to Phish Server Security Group
```

High Availability Architecture

HA Requirements

Availability Targets:

- **Uptime:** 99.9% (8.76 hours downtime/year)
- **RTO (Recovery Time Objective):** < 15 minutes
- **RPO (Recovery Point Objective):** < 5 minutes

HA Components

Application Tier HA:

- Multiple instances behind load balancer
- Health checks with automatic failover
- Auto-scaling for capacity
- Graceful connection draining

Database Tier HA:

- Multi-AZ deployment (primary + standby)
- Automatic failover (< 2 minutes)
- Read replicas for scaling
- Point-in-time recovery

Network Tier HA:

- Multi-AZ load balancer
- Multiple NAT Gateways
- Redundant VPN connections

Failure Scenarios & Response

```
graph TB
    subgraph "Failure Detection"
        HC[Health Check<br/>Failure]
        Monitor[Monitoring<br/>Alert]
    end

    subgraph "Automated Response"
```

```
    LB[Load Balancer<br/>Removes Instance]
    ASG[Auto Scaling<br/>Launches Replacement]
    DB_Failover[Database<br/>Automatic Failover]
end

subgraph "Manual Response"
    Alert[Alert<br/>Operations Team]
    Investigate[Investigate<br/>Root Cause]
    Remediate[Remediate<br/>Issue]
end

HC --> LB
HC --> ASG
Monitor --> Alert

LB --> |Instance Unhealthy| ASG
ASG --> |Launch New Instance| HC

Monitor --> DB_Failover
DB_Failover --> Alert

Alert --> Investigate
Investigate --> Remediate

style HC fill:#ffcdd2
style LB fill:#fff9c4
style ASG fill:#c8e6c9
```

Scenario 1: Single Server Failure

- **Detection:** Health check fails after 3 attempts (~90 seconds)
- **Response:** Load balancer stops routing traffic
- **Recovery:** Auto-scaling launches new instance (~5 minutes)
- **Impact:** No user impact, traffic served by remaining instances

Scenario 2: Database Failure

- **Detection:** Database health check fails
- **Response:** Automatic failover to standby (AWS RDS / Azure MySQL)
- **Recovery:** < 2 minutes
- **Impact:** Brief connection errors, applications reconnect automatically

Scenario 3: Availability Zone Failure

- **Detection:** All instances in AZ fail health checks

- **Response:** Traffic routed to instances in other AZ
- **Recovery:** Immediate
- **Impact:** No user impact if multi-AZ deployed

Scenario 4: Region Failure (DR)

- **Detection:** Regional health check fails
- **Response:** Manual or automatic failover to DR region
- **Recovery:** 15-30 minutes (depending on automation)
- **Impact:** Brief outage during DNS propagation

Network Architecture

DNS Configuration

Primary Domain: gophish.example.com

```
Type: A
Name: gophish.example.com
Value: [Load Balancer IP/DNS]
TTL: 300
```

Phishing Domain: phish.example.com

```
Type: CNAME
Name: phish.example.com
Value: gophish.example.com
TTL: 300
```

Email Domain:

```
Type: MX
Name: example.com
Value: 10 mail.example.com
TTL: 3600

Type: TXT (SPF)
Name: example.com
Value: "v=spf1 include:_spf.google.com ~all"

Type: TXT (DKIM)
Name: default._domainkey.example.com
```

```
Value: "v=DKIM1; k=rsa; p=[public key]"

Type: TXT (DMARC)
Name: _dmarc.example.com
Value: "v=DMARC1; p=quarantine; rua=mailto:dmarc@example.com"
```

SSL/TLS Configuration

Certificate Sources:

- Let's Encrypt (free, automated)
- AWS Certificate Manager (AWS deployments)
- Azure Key Vault (Azure deployments)
- Commercial CA (extended validation)

TLS Best Practices:

- TLS 1.2 minimum, TLS 1.3 preferred
- Strong cipher suites only
- Perfect Forward Secrecy (PFS)
- HTTP Strict Transport Security (HSTS)
- Certificate pinning (optional)

Database Architecture

Database Sizing

Small Deployment (< 2,000 users):

- SQLite3: Local file, 1-2GB
- No separate database server needed
- Backup: File-level backups

Medium Deployment (2,000-10,000 users):

- MySQL: db.t3.medium (2 vCPU, 4GB RAM)
- Storage: 100GB SSD
- Connections: 200 max
- Single AZ acceptable

Large Deployment (10,000-50,000 users):

- MySQL: db.r5.large (2 vCPU, 16GB RAM)
- Storage: 500GB SSD
- Connections: 500 max
- Multi-AZ required
- Read replicas: 1-2

Enterprise Deployment (50,000+ users):

- MySQL: db.r5.xlarge+ (4+ vCPU, 32+ GB RAM)
- Storage: 1TB+ SSD
- Connections: 1000 max
- Multi-AZ required
- Read replicas: 2-4
- Database clustering (Aurora MySQL)

Database Maintenance

Regular Tasks:

- **Daily:** Automated backups
- **Weekly:** Backup validation, log review
- **Monthly:** Performance tuning, index optimization
- **Quarterly:** Capacity planning, upgrade planning
- **Annually:** DR test, security audit

Performance Monitoring:

- Query execution time
- Slow query log
- Connection pool usage
- Disk I/O
- Replication lag
- Table locks

Scaling Considerations

Horizontal Scaling

Application Tier:

- Add more admin/phish server instances

- Configure auto-scaling based on:
  - CPU utilization (> 70%)
  - Memory utilization (> 80%)
  - Request count (> 1000/min per instance)
  - Custom metrics (campaigns running)

Example Auto-Scaling Policy (AWS):

```
Target Tracking:
  Metric: Average CPU Utilization
  Target: 70%
  Min Instances: 2
  Max Instances: 10
  Cooldown: 300 seconds
```

Vertical Scaling

When to Scale Up:

- Consistently high resource usage
- Database performance degradation
- Increased storage requirements

Scaling Strategy:

1. Monitor performance metrics
2. Identify bottleneck (CPU, memory, I/O)
3. Schedule maintenance window
4. Upgrade instance size
5. Validate performance improvement

Performance Optimization

Application Level:

- Database query optimization
- Connection pooling
- Caching frequently accessed data
- Asynchronous processing (email sending)
- Load balancer session affinity

Database Level:

- Proper indexing
- Query optimization
- Read replicas for reporting
- Partitioning large tables
- Archive old campaign data

Network Level:

- CDN for static assets (optional)
- Compression (gzip)
- Connection keep-alive
- Optimal MTU settings

Cost Estimation

AWS Cost Estimates

Small Deployment:

- EC2 (1x t3.small): $15/month
- RDS MySQL (db.t3.micro): $15/month
- Load Balancer: $20/month
- Data Transfer: $10/month
- **Total: ~$60/month**

Medium Deployment:

- EC2 (2x t3.medium): $60/month
- RDS MySQL (db.t3.medium): $60/month
- Load Balancer: $20/month
- S3, CloudWatch: $20/month
- Data Transfer: $30/month
- **Total: ~$190/month**

Large HA Deployment:

- EC2 (4x t3.large): $240/month
- RDS MySQL Multi-AZ (db.r5.large): $320/month
- Load Balancer: $40/month
- ElastiCache Redis: $50/month
- S3, CloudWatch: $50/month
- Data Transfer: $100/month

- **Total: ~$800/month**

Azure Cost Estimates

Small Deployment:

- VM (1x Standard_B2s): $30/month
- MySQL Basic: $25/month
- Load Balancer: $20/month
- **Total: ~$75/month**

Medium Deployment:

- VM (2x Standard_B2ms): $90/month
- MySQL General Purpose: $130/month
- Application Gateway: $150/month
- Storage, Monitoring: $30/month
- **Total: ~$400/month**

Large HA Deployment:

- VM Scale Sets (4-10 instances): $400/month
- MySQL Zone-Redundant HA: $450/month
- Application Gateway: $150/month
- Redis Cache: $100/month
- Storage, Monitoring: $100/month
- **Total: ~$1,200/month**

Deployment Checklist

Pre-Deployment

- [ ] Architecture design reviewed and approved
- [ ] Sizing calculations completed
- [ ] Cost estimation and budget approval
- [ ] DNS domains registered
- [ ] SSL certificates obtained
- [ ] Cloud accounts configured
- [ ] Network design documented
- [ ] Security requirements defined
- [ ] Compliance requirements identified

- [ ] Monitoring plan created
- [ ] Backup and DR plan documented

Deployment

- [ ] VPC/VNet created with subnets
- [ ] Security groups/NSGs configured
- [ ] Database instance deployed
- [ ] Database initialized and secured
- [ ] Application servers deployed
- [ ] Gophish installed and configured
- [ ] Load balancer configured
- [ ] DNS records created
- [ ] SSL certificates installed
- [ ] Monitoring configured
- [ ] Logging configured
- [ ] Backups configured
- [ ] Auto-scaling configured (if applicable)

Post-Deployment

- [ ] Smoke tests passed
- [ ] Load testing completed
- [ ] Security scan completed
- [ ] Backup tested and validated
- [ ] DR procedure tested
- [ ] Documentation updated
- [ ] Team trained on operations
- [ ] Monitoring alerts configured
- [ ] Runbooks created
- [ ] Go-live approval obtained

Conclusion

This technical architecture document provides comprehensive guidance for deploying Gophish in secure, scalable, and highly available configurations across AWS and Azure platforms. The architectures can be adapted based on specific organizational requirements, scale, and budget constraints.

Key takeaways:

- Start with simpler architectures and scale as needed
- Security should be built in from the start
- High availability requires investment but provides resilience
- Regular monitoring and maintenance are essential
- Cost optimization is an ongoing process

Document Version: 1.0
Last Updated: 2025-11-05
Author: Technical Architecture Team