AI-Based Automated STEM Evaluation System - Deployment and Infrastructure Plan

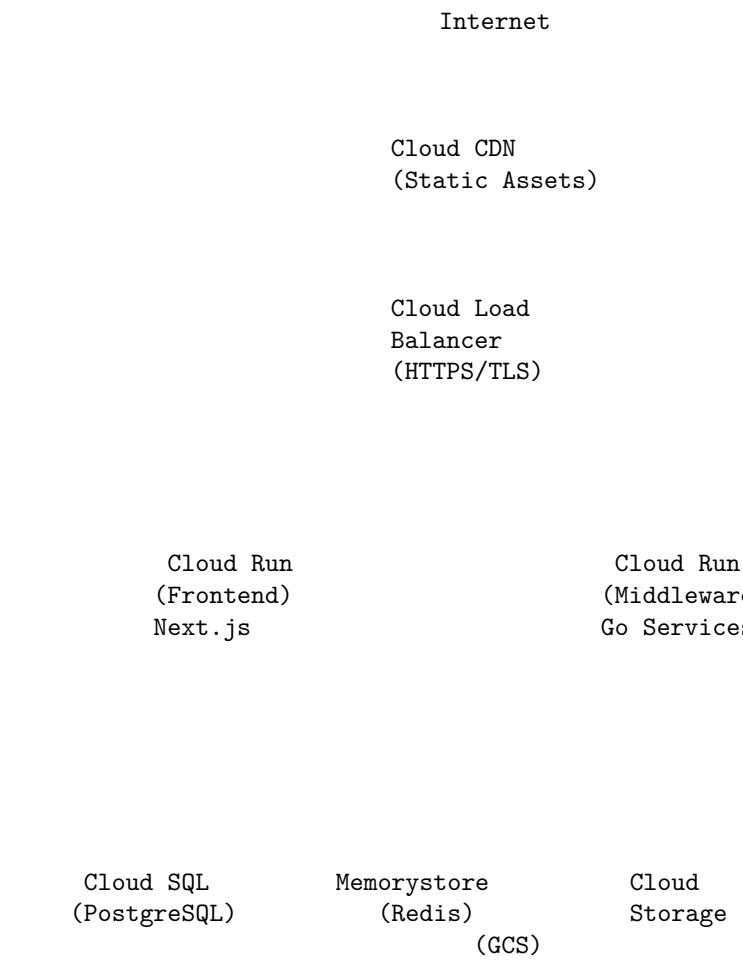
1. Infrastructure Overview

1.1 Cloud Platform

Provider: Google Cloud Platform (GCP)

Rationale: - Native Vertex AI integration - Competitive pricing - Strong compliance certifications (SOC 2, HIPAA, FERPA-ready) - Excellent uptime SLA - Integrated services (Cloud Run, Cloud SQL, GCS)

1.2 Deployment Architecture



1.3 Environments

Environment	Purpose	Auto-Deploy	Data
Development	Local development	No	Seeded test data
Staging	Testing and QA	Yes (on merge to develop)	Anonymized production copy
Production	Live system	Yes (manual approval)	Real user data

1.4 Domain Configuration

Primary Domain: stemblock.ai

Subdomains: - `www.stemblock.ai` - Main web application (Next.js frontend) - `api.stemblock.ai` - API backend (Go microservices) - `cdn.stemblock.ai` - Static assets (Cloud CDN) - `staging.stemblock.ai` - Staging environment - `docs.stemblock.ai` - API documentation (Swagger UI)

SSL/TLS: - Managed by Google Cloud Load Balancer - Auto-renewal certificates - TLS 1.3 minimum

2. Infrastructure Components

2.1 Frontend (Next.js)

Cloud Run Configuration **Service**: `frontend-service`

```yaml
service: frontend-service
region: us-central1
runtime: nodejs20

resources:
  cpu: 1
  memory: 512Mi

scaling:
  minInstances: 2
  maxInstances: 100
  targetConcurrency: 80

environment:
  - name: NODE_ENV
    value: production
  - name: NEXT_PUBLIC_API_URL
    value: https://api.stemblock.ai
```

```
  - name: NEXT_PUBLIC_WS_URL
    value: wss://api.stemblock.ai/ws

secrets:
  - name: GOOGLE_OAUTH_CLIENT_ID
    key: google-oauth-client-id
    version: latest
```

Deployment Strategy: - Docker container build from Next.js - Optimize build: Static generation for public pages - Image registry: Google Artifact Registry - Rollout: Blue-green deployment (Cloud Run native)

Build Process:

```
# Dockerfile for Next.js
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

FROM node:20-alpine AS runner
WORKDIR /app
ENV NODE_ENV production
COPY --from=builder /app/next.config.js ./
COPY --from=builder /app/public ./public
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./package.json

EXPOSE 3000
CMD ["npm", "start"]
```

CDN Configuration **Cloud CDN** for static assets: - Cache-Control: 1 year for immutable assets - Cache invalidation on deployment - Compression: Brotli + Gzip

2.2 Middleware (Go Services)

Cloud Run Services **API Gateway Service**:

```
service: api-gateway-service
region: us-central1
runtime: go1.21

resources:
  cpu: 1
```

```
  memory: 256Mi

scaling:
  minInstances: 2
  maxInstances: 50

environment:
  - name: PORT
    value: "8080"
  - name: ENV
    value: production
```

Individual Microservices: - `auth-service` - `user-service` - `assessment-service` - `evaluation-service` - `report-service` - `forum-service` - `notification-service`

Common Configuration:

```
resources:
  cpu: 0.5 to 1
  memory: 256Mi to 512Mi

scaling:
  minInstances: 1 (2 for critical services)
  maxInstances: 20
```

Build Process:

```
# Dockerfile for Go Services
FROM golang:1.21-alpine AS builder
WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main ./cmd/server

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /app/main .
EXPOSE 8080
CMD ["./main"]
```

Service Mesh (Optional Phase 2) For advanced networking: - **Istio** or **Cloud Service Mesh** - mTLS between services - Circuit breaking - Advanced traffic routing

2.3 Database (Cloud SQL)

PostgreSQL Configuration **Instance**: `stem-eval-db-prod`

```
Tier: db-custom-2-8192 (2 vCPUs, 8GB RAM)
Storage: SSD, 100GB (auto-resize enabled)
Version: PostgreSQL 15
High Availability: Enabled (regional HA)
Backups:
  - Automated daily backups (7-day retention)
  - Point-in-time recovery enabled
  - Backup window: 02:00-03:00 UTC
Maintenance: Sunday 03:00-04:00 UTC
Flags:
  - max_connections: 200
  - shared_buffers: 2GB
  - effective_cache_size: 6GB
  - work_mem: 10MB
```

Connection Pooling: PgBouncer (or Cloud SQL Proxy with pooling)

Read Replicas (Future): - For reporting queries - Reduce load on primary - Geographic distribution

Database Access **Cloud SQL Auth Proxy**: - Services connect via proxy (no public IP) - IAM-based authentication - Automatic connection encryption

Connection String:

```
postgresql://user:password@localhost:5432/stem_eval?sslmode=require
```

Migration Management: - Tool: `golang-migrate` - Migrations in Git repository - Applied via CI/CD pipeline - Rollback capability

2.4 Cache (Memorystore for Redis)

Instance: `stem-eval-redis-prod`

```
Tier: Basic (Standard for production)
Memory: 5GB
Version: Redis 7.0
Region: us-central1
High Availability: Enabled (Standard tier)
```

Usage: - Session storage - API rate limiting - Cached API responses - AI prompt cache - Real-time data (forum vote counts)

Connection: - Private IP within VPC - No public access - Encrypted in transit

2.5 Object Storage (Cloud Storage)

Buckets:

1. **User Uploads** (`prod-stem-eval-uploads`):
   - User-submitted files (images, code, documents)
   - Location: us-central1
   - Storage class: Standard
   - Versioning: Disabled
   - Lifecycle: None (retain indefinitely)
   - Access: Signed URLs only
2. **Processed Files** (`prod-stem-eval-processed`):
   - Thumbnails, processed images
   - Location: us-central1
   - Storage class: Standard
   - Lifecycle: Delete after 90 days if not accessed
   - Access: Public read (CDN)
3. **Reports** (`prod-stem-eval-reports`):
   - Generated PDF reports
   - Location: us-central1
   - Storage class: Standard
   - Lifecycle: Delete after 30 days
   - Access: Signed URLs
4. **Backups** (`prod-stem-eval-backups`):
   - Database backups, config backups
   - Location: us-central1
   - Storage class: Nearline (cost-effective for backups)
   - Lifecycle: Retain 90 days
   - Access: Private (admin only)

Security: - IAM-based access control - Encryption at rest (Google-managed keys) - Uniform bucket-level access - CORS configuration for uploads

2.6 Secrets Management

Google Secret Manager

Secrets: - `db-password` - `redis-password` - `jwt-private-key` - `jwt-public-key` - `vertex-ai-api-key` - `sendgrid-api-key` - `google-oauth-client-secret` - `encryption-key-primary` - `encryption-key-secondary` (for key rotation)

Access Control: - IAM roles per service - Principle of least privilege - Audit logging enabled

Rotation: - Quarterly rotation schedule - Automated rotation (where possible) - Zero-downtime rotation (support multiple active keys)

2.7 External Services

Vertex AI Project: Same GCP project **Models**: - Gemini 1.5 Pro (text analysis) - Gemini 1.5 Pro Vision (image analysis)

API Configuration: - Service account authentication - Quota management: Set budget alerts - Request caching: 24-hour cache for identical prompts

Cost Management: - Monthly budget: $5,000 (adjust based on usage) - Alerts at 50%, 80%, 100% - Usage monitoring dashboard

SendGrid Plan: Pro ($89.95/month, 100k emails)

Configuration: - API key stored in Secret Manager - Dedicated IP address (for better deliverability) - Email templates for transactional emails - Webhook for delivery status

Email Types: - Welcome emails - Password reset - Notification emails - Weekly digests

Google OAuth OAuth 2.0 Credentials: - Client ID and Secret in Secret Manager - Authorized redirect URIs: - `https://www.stemblock.ai/auth/callback` - `https://staging.stemblock.ai/auth/callback` (staging) - Scopes: `openid`, `email`, `profile`

3. Infrastructure as Code (Terraform)

3.1 Terraform Structure

```
/infrastructure
  /modules
     /cloud-run
     /cloud-sql
     /cloud-storage
     /redis
     /load-balancer
     /secret-manager
     /vpc
  /environments
     /dev
        main.tf
        variables.tf
        terraform.tfvars
     /staging
        main.tf
        variables.tf
        terraform.tfvars
     /prod
```

```
        main.tf
        variables.tf
        terraform.tfvars
  backend.tf
  versions.tf
```

3.2 Sample Terraform Configuration

Cloud Run Service (`modules/cloud-run/main.tf`):

```
resource "google_cloud_run_service" "service" {
  name     = var.service_name
  location = var.region

  template {
    spec {
      containers {
        image = var.image

        resources {
          limits = {
            cpu    = var.cpu
            memory = var.memory
          }
        }

        env {
          name  = "ENV"
          value = var.environment
        }

        dynamic "env" {
          for_each = var.secrets
          content {
            name = env.value.name
            value_from {
              secret_key_ref {
                name = env.value.secret_name
                key  = env.value.secret_key
              }
            }
          }
        }
      }

      service_account_name = var.service_account
```

```
    }

    metadata {
      annotations = {
        "autoscaling.knative.dev/minScale" = var.min_instances
        "autoscaling.knative.dev/maxScale" = var.max_instances
      }
    }
  }

  traffic {
    percent         = 100
    latest_revision = true
  }
}

resource "google_cloud_run_service_iam_member" "public_access" {
  count    = var.allow_unauthenticated ? 1 : 0
  service  = google_cloud_run_service.service.name
  location = google_cloud_run_service.service.location
  role     = "roles/run.invoker"
  member   = "allUsers"
}
```

Cloud SQL (`modules/cloud-sql/main.tf`):

```
resource "google_sql_database_instance" "main" {
  name             = var.instance_name
  database_version = "POSTGRES_15"
  region           = var.region

  settings {
    tier              = var.tier
    availability_type = "REGIONAL"
    disk_type         = "PD_SSD"
    disk_size         = var.disk_size
    disk_autoresize   = true

    backup_configuration {
      enabled                        = true
      start_time                     = "02:00"
      point_in_time_recovery_enabled = true
      transaction_log_retention_days = 7
      backup_retention_settings {
        retained_backups = 7
      }
    }
```

```
    ip_configuration {
      ipv4_enabled    = false
      private_network = var.vpc_id
    }

    database_flags {
      name  = "max_connections"
      value = "200"
    }
  }

  deletion_protection = true
}

resource "google_sql_database" "database" {
  name     = var.database_name
  instance = google_sql_database_instance.main.name
}

resource "google_sql_user" "user" {
  name     = var.db_user
  instance = google_sql_database_instance.main.name
  password = var.db_password
}
```

3.3 Terraform Backend

State Storage: GCS Bucket

```
terraform {
  backend "gcs" {
    bucket = "stem-eval-terraform-state"
    prefix = "prod"
  }
}
```

State Locking: Enabled (GCS native)

Best Practices: - Separate state files per environment - State file versioning enabled - Access restricted to DevOps team

4. CI/CD Pipeline

4.1 Pipeline Overview

Tool: GitHub Actions

Triggers: - Push to `main` branch → Deploy to production (with approval) - Push to `develop` branch → Deploy to staging (automatic) - Pull request → Run tests and build (no deploy)

4.2 CI Pipeline

Workflow: `.github/workflows/ci.yml`

```yaml
name: CI Pipeline

on:
  pull_request:
    branches: [main, develop]
  push:
    branches: [develop]

jobs:
  lint-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '20'
      - run: npm ci
        working-directory: ./frontend
      - run: npm run lint
        working-directory: ./frontend

  test-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '20'
      - run: npm ci
        working-directory: ./frontend
      - run: npm run test:ci
        working-directory: ./frontend

  lint-backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-go@v4
```

```yaml
      with:
        go-version: '1.21'
    - run: go mod download
      working-directory: ./services
    - run: golangci-lint run
      working-directory: ./services

test-backend:
  runs-on: ubuntu-latest
  services:
    postgres:
      image: postgres:15
      env:
        POSTGRES_PASSWORD: postgres
      options: >-
        --health-cmd pg_isready
        --health-interval 10s
        --health-timeout 5s
        --health-retries 5
  steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-go@v4
      with:
        go-version: '1.21'
    - run: go mod download
      working-directory: ./services
    - run: go test -v -race -coverprofile=coverage.out ./...
      working-directory: ./services
      env:
        DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test?sslmode=disable

security-scan:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v3
    - name: Run Trivy security scan
      uses: aquasecurity/trivy-action@master
      with:
        scan-type: 'fs'
        scan-ref: '.'
        format: 'sarif'
        output: 'trivy-results.sarif'
    - name: Upload Trivy results to GitHub Security
      uses: github/codeql-action/upload-sarif@v2
      with:
        sarif_file: 'trivy-results.sarif'
```

```yaml
  build-frontend:
    runs-on: ubuntu-latest
    needs: [lint-frontend, test-frontend]
    steps:
      - uses: actions/checkout@v3
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      - name: Build Docker image
        uses: docker/build-push-action@v4
        with:
          context: ./frontend
          push: false
          tags: frontend:${{ github.sha }}

  build-backend:
    runs-on: ubuntu-latest
    needs: [lint-backend, test-backend]
    steps:
      - uses: actions/checkout@v3
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      - name: Build Docker image
        uses: docker/build-push-action@v4
        with:
          context: ./services/api-gateway
          push: false
          tags: api-gateway:${{ github.sha }}
```

4.3 CD Pipeline (Staging)

Workflow: `.github/workflows/deploy-staging.yml`

```yaml
name: Deploy to Staging

on:
  push:
    branches: [develop]

jobs:
  deploy-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Authenticate to Google Cloud
```

```yaml
      uses: google-github-actions/auth@v1
      with:
        credentials_json: ${{ secrets.GCP_SA_KEY_STAGING }}

    - name: Set up Cloud SDK
      uses: google-github-actions/setup-gcloud@v1

    - name: Configure Docker for Artifact Registry
      run: gcloud auth configure-docker us-central1-docker.pkg.dev

    - name: Build and push Docker image
      run: |
        docker build -t us-central1-docker.pkg.dev/stem-eval-staging/frontend:${{ github.s
        docker push us-central1-docker.pkg.dev/stem-eval-staging/frontend:${{ github.sha }

    - name: Deploy to Cloud Run
      run: |
        gcloud run deploy frontend-service \
          --image us-central1-docker.pkg.dev/stem-eval-staging/frontend:${{ github.sha }}
          --region us-central1 \
          --platform managed \
          --allow-unauthenticated

  deploy-backend:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        service: [api-gateway, auth-service, user-service, assessment-service, evaluation-se
    steps:
      - uses: actions/checkout@v3

      - name: Authenticate to Google Cloud
        uses: google-github-actions/auth@v1
        with:
          credentials_json: ${{ secrets.GCP_SA_KEY_STAGING }}

      - name: Set up Cloud SDK
        uses: google-github-actions/setup-gcloud@v1

      - name: Configure Docker for Artifact Registry
        run: gcloud auth configure-docker us-central1-docker.pkg.dev

      - name: Build and push Docker image
        run: |
          docker build -t us-central1-docker.pkg.dev/stem-eval-staging/${{ matrix.service }}
          docker push us-central1-docker.pkg.dev/stem-eval-staging/${{ matrix.service }}:${{
```

```
      - name: Deploy to Cloud Run
        run: |
          gcloud run deploy ${{ matrix.service }} \
            --image us-central1-docker.pkg.dev/stem-eval-staging/${{ matrix.service }}:${{ g
            --region us-central1 \
            --platform managed \
            --no-allow-unauthenticated

  run-smoke-tests:
    runs-on: ubuntu-latest
    needs: [deploy-frontend, deploy-backend]
    steps:
      - uses: actions/checkout@v3
      - name: Run smoke tests
        run: |
          npm ci
          npm run test:smoke
        env:
          BASE_URL: https://staging.stemblock.ai
```

4.4 CD Pipeline (Production)

Workflow: `.github/workflows/deploy-production.yml`

```
name: Deploy to Production

on:
  push:
    branches: [main]

jobs:
  deploy-approval:
    runs-on: ubuntu-latest
    steps:
      - name: Wait for approval
        uses: trstringer/manual-approval@v1
        with:
          secret: ${{ github.TOKEN }}
          approvers: maryang,tech-lead
          minimum-approvals: 1
          issue-title: "Deploy to Production"

  deploy-frontend:
    runs-on: ubuntu-latest
    needs: deploy-approval
```

```
    steps:
      # Similar to staging, but with production credentials
      # and blue-green deployment strategy

  deploy-backend:
    runs-on: ubuntu-latest
    needs: deploy-approval
    strategy:
      matrix:
        service: [api-gateway, auth-service, ...]
    steps:
      # Similar to staging deployment

  run-smoke-tests:
    runs-on: ubuntu-latest
    needs: [deploy-frontend, deploy-backend]
    steps:
      - name: Run smoke tests
        run: npm run test:smoke
        env:
          BASE_URL: https://www.stemblock.ai

  notify-team:
    runs-on: ubuntu-latest
    needs: run-smoke-tests
    steps:
      - name: Send Slack notification
        uses: slackapi/slack-github-action@v1
        with:
          payload: |
            {
              "text": "  Production deployment successful! Version: ${{ github.sha }}"
            }
        env:
          SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
```

4.5 Database Migrations

Automated Migration Pipeline:

```
name: Run Database Migrations

on:
  workflow_dispatch:
    inputs:
      environment:
```

```
        description: 'Environment to run migrations'
        required: true
        type: choice
        options:
          - staging
          - production

jobs:
  migrate:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Authenticate to Google Cloud
        uses: google-github-actions/auth@v1
        with:
          credentials_json: ${{ secrets.GCP_SA_KEY }}

      - name: Set up Cloud SQL Proxy
        run: |
          wget https://dl.google.com/cloudsql/cloud_sql_proxy.linux.amd64 -O cloud_sql_proxy
          chmod +x cloud_sql_proxy
          ./cloud_sql_proxy -instances=${{ secrets.INSTANCE_CONNECTION_NAME }}=tcp:5432 &

      - name: Install golang-migrate
        run: |
          curl -L https://github.com/golang-migrate/migrate/releases/download/v4.15.2/migrat
          sudo mv migrate /usr/local/bin/

      - name: Run migrations
        run: |
          migrate -path ./migrations -database "${{ secrets.DATABASE_URL }}" up

      - name: Verify migrations
        run: |
          migrate -path ./migrations -database "${{ secrets.DATABASE_URL }}" version
```

Migration Safety: - Always test migrations in staging first - Use transactions where possible - Write rollback migrations - Backup database before major migrations - Monitor application after migration

5. Monitoring and Observability

5.1 Application Monitoring

Google Cloud Monitoring: - Automatic Cloud Run metrics (CPU, memory, requests) - Custom metrics (business KPIs) - Uptime checks - Alerting policies

Custom Metrics:

```
// In Go services
import "cloud.google.com/go/monitoring"

// Track AI API usage
aiAPICallsCounter.Inc()
aiAPICostGauge.Set(cost)

// Track submissions
submissionsCounter.Inc()
```

Dashboards: 1. **System Health**: CPU, memory, error rate, latency 2. **Business Metrics**: DAU, submissions, assessments 3. **AI Usage**: API calls, costs, response times 4. **Database**: Query performance, connection pool

5.2 Logging

Cloud Logging: - Structured JSON logs - Centralized log aggregation - Log-based metrics - Log retention: 30 days

Log Levels: - DEBUG: Development only - INFO: Important events - WARN: Potential issues - ERROR: Errors that don't crash the service - FATAL: Critical errors

Example Log Entry:

```
{
  "timestamp": "2025-10-18T10:30:00Z",
  "severity": "INFO",
  "service": "evaluation-service",
  "trace": "projects/stem-eval/traces/abc123",
  "message": "Submission evaluated",
  "user_id": "user-uuid",
  "submission_id": "sub-uuid",
  "ai_model": "gemini-pro",
  "processing_time_ms": 3500,
  "cost": 0.05
}
```

5.3 Tracing

Cloud Trace: - Distributed tracing across services - Automatic instrumentation (Cloud Run) - Manual span creation for important operations

OpenTelemetry (Alternative): - Vendor-neutral tracing - Compatible with Cloud Trace - More flexibility

5.4 Error Tracking

Sentry or **Cloud Error Reporting**: - Real-time error notifications - Error grouping and trends - Stack traces with source maps - User impact analysis

5.5 Alerting

Alert Policies:

1. **Critical (PagerDuty)**:
   - Service down (no successful requests in 5 minutes)
   - Error rate > 5%
   - Database connections exhausted
   - Disk space > 90%
2. **High (Slack)**:
   - Error rate > 1%
   - P95 latency > 1 second
   - AI API budget > 80%
   - Failed login attempts > 100/minute
3. **Medium (Email)**:
   - Error rate > 0.5%
   - Memory usage > 80%
   - Certificate expiration in 30 days

On-Call Rotation: - 24/7 on-call for critical alerts - Weekly rotation - Runbooks for common issues

6. Backup and Disaster Recovery

6.1 Database Backups

Automated Backups: - Daily automated backups at 02:00 UTC - Retention: 7 days (automated), 30 days (manual) - Point-in-time recovery: 7 days - Backup stored in same region (us-central1)

Manual Backups: - Before major migrations - Before major releases - Monthly full backups (retained 1 year)

Backup Testing: - Quarterly restore test - Document restore procedure - Measure RTO (Recovery Time Objective): 4 hours - Measure RPO (Recovery Point Objective): 1 hour

6.2 Disaster Recovery Plan

Scenarios: 1. Database failure 2. Region failure 3. Accidental data deletion 4. Ransomware attack 5. Service misconfiguration

Recovery Steps:

Database Failure: 1. Automatic failover to standby (HA enabled) 2. If standby fails, restore from backup 3. ETA: 15 minutes (failover) or 2 hours (restore)

Region Failure: 1. Redirect traffic to DR region (Phase 2) 2. Restore database from backup in DR region 3. Update DNS to point to DR region 4. ETA: 4-6 hours

Data Deletion: 1. Restore from point-in-time recovery 2. Merge with current data if needed 3. ETA: 2-4 hours

Disaster Recovery Testing: - Annual full DR test - Tabletop exercises quarterly - Document and refine procedures

6.3 Data Retention

Production Data: - Active users: Indefinitely - Inactive users (1 year): Archive to cold storage - Deleted users: 30-day grace period, then purge

Logs: - Application logs: 30 days - Audit logs: 1 year - Access logs: 90 days

Backups: - Automated DB backups: 7 days - Manual DB backups: 1 year - File backups (GCS): 90 days

7. Security and Compliance

7.1 Network Security

VPC Configuration:

```
VPC: stem-eval-vpc
Subnets:
  - Public subnet (10.0.0.0/24): Load balancers
  - Private subnet (10.0.1.0/24): Cloud Run services
  - Database subnet (10.0.2.0/24): Cloud SQL, Redis
```

Firewall Rules: - Default deny all ingress - Allow HTTPS (443) from internet to load balancer - Allow internal communication within VPC - No direct database access from internet

Cloud Armor (WAF): - DDoS protection - Rate limiting at edge - Geo-blocking (if needed) - Custom rules for application-specific threats

7.2 IAM and Service Accounts

Service Accounts: - `frontend-service@stem-eval.iam.gserviceaccount.com` - Permissions: Read secrets, write logs - `backend-services@stem-eval.iam.gserviceaccount.com` - Permissions: Database access, storage access, Vertex AI access - `ci-cd-deployer@stem-eval.iam.gservicea` - Permissions: Deploy Cloud Run, push images

Best Practices: - One service account per service (or group of similar services) - Principle of least privilege - No overly permissive roles (e.g., Owner, Editor) - Regular IAM audit

7.3 Compliance

FERPA: - Data encryption at rest and in transit - Access logs and audit trails - RBAC enforcement - Annual compliance review

COPPA (if applicable): - Age verification - Parental consent workflow - Limited data collection for children

SOC 2 Type II (Future): - Annual audit by external auditor - Controls for security, availability, confidentiality

8. Cost Management

8.1 Estimated Monthly Costs

Service	Configuration	Cost
Cloud Run (Frontend)	2-100 instances, 512Mi	$150
Cloud Run (Backend)	8 services, 1-20 instances	$400
Cloud SQL (PostgreSQL)	db-custom-2-8192, HA	$350
Memorystore (Redis)	5GB, Standard	$200
Cloud Storage	500GB storage, 1TB egress	$100
Cloud Load Balancer	1TB ingress	$50
Cloud CDN	500GB cache, 1TB egress	$80
Vertex AI	50k API calls/month	$500
SendGrid	Pro plan	$90
Total		**~$1,920/month**

Scaling Estimates: - 1,000 users: ~$2,000/month - 5,000 users: ~$5,000/month - 10,000 users: ~$10,000/month

8.2 Cost Optimization

Strategies: 1. **Right-sizing**: Adjust instance sizes based on actual usage 2. **Auto-scaling**: Scale down during low traffic periods 3. **Committed Use Discounts**: 1-year or 3-year commits for predictable workloads 4. **AI Cost Management**: - Prompt caching (24-hour cache) - Batch processing - Model selection (use cheaper models where appropriate) 5. **Storage Lifecycle Policies**: - Move old files to Nearline storage - Delete unused files 6. **Reserved IPs**: Use ephemeral IPs where possible 7. **Monitoring**: Set budget alerts, review monthly costs

8.3 Budget Alerts

```
resource "google_billing_budget" "monthly_budget" {
  billing_account = var.billing_account
  display_name    = "Monthly Budget"

  budget_filter {
    projects = ["projects/${var.project_id}"]
  }

  amount {
    specified_amount {
      currency_code = "USD"
      units         = "2000"
    }
  }

  threshold_rules {
    threshold_percent = 0.5
  }
  threshold_rules {
    threshold_percent = 0.8
  }
  threshold_rules {
    threshold_percent = 1.0
  }

  all_updates_rule {
    pubsub_topic = google_pubsub_topic.budget_alerts.id
  }
}
```

9. Performance Optimization

9.1 Frontend Optimization

- **Code Splitting**: Split by route, load on demand
- **Image Optimization**: next/image with WebP format
- **Lazy Loading**: Defer non-critical components
- **CDN**: Serve static assets from CDN
- **Compression**: Brotli for text, WebP for images
- **Caching**: Cache-Control headers for immutable assets
- **Prefetching**: Prefetch critical resources

Target Metrics: - First Contentful Paint: < 1.8s - Largest Contentful Paint: < 2.5s - Time to Interactive: < 3.8s - Cumulative Layout Shift: < 0.1

9.2 Backend Optimization

- **Connection Pooling**: Reuse database connections
- **Query Optimization**: Use indexes, avoid N+1 queries
- **Caching**: Redis for frequently accessed data
- **Pagination**: Limit result sets
- **Async Processing**: Background workers for heavy tasks
- **Vertical Scaling**: Increase CPU/memory for bottlenecks
- **Horizontal Scaling**: Add more instances

Target Metrics: - API P50 latency: $<$ 100ms - API P95 latency: $<$ 500ms - API P99 latency: $<$ 1000ms

9.3 Database Optimization

- **Indexes**: Strategic indexing on foreign keys and query fields
- **Query Caching**: Enable PostgreSQL query cache
- **Read Replicas**: Offload read queries (Phase 2)
- **Partitioning**: Partition large tables by date
- **Vacuum**: Regular VACUUM ANALYZE
- **Connection Pooling**: PgBouncer

10. Deployment Checklist

10.1 Pre-Launch Checklist

Infrastructure: - [ ] All Terraform configurations applied - [ ] Domains configured and DNS propagated - [ ] SSL certificates issued and valid - [ ] VPC and firewall rules configured - [ ] Database created and migrations applied - [ ] Redis instance provisioned - [ ] Storage buckets created with proper permissions

Services: - [ ] All services deployed to Cloud Run - [ ] Environment variables configured - [ ] Secrets stored in Secret Manager - [ ] Service accounts created with proper permissions - [ ] Health checks passing

CI/CD: - [ ] GitHub Actions workflows configured - [ ] Deployment pipelines tested - [ ] Rollback procedure documented - [ ] Manual approval configured for production

Monitoring: - [ ] Monitoring dashboards created - [ ] Alert policies configured - [ ] Error tracking integrated - [ ] Log aggregation working - [ ] On-call schedule defined

Security: - [ ] Security scan passed - [ ] Penetration test completed - [ ] Secrets rotated - [ ] HTTPS enforced - [ ] CORS configured - [ ] Rate limiting enabled

Compliance: - [ ] Privacy policy published - [ ] Terms of service published - [ ] FERPA compliance verified - [ ] Parental consent workflow tested

Backups: - [ ] Automated backups configured - [ ] Backup restore tested - [ ] Disaster recovery plan documented

Performance: - [ ] Load testing completed - [ ] Performance benchmarks met - [ ] CDN configured - [ ] Caching verified

Documentation: - [ ] Architecture documented - [ ] API documentation published - [ ] Deployment procedures documented - [ ] Runbooks created - [ ] User guides published

10.2 Launch Day Checklist

Pre-Launch (T-1 hour): - [ ] Final deployment to production - [ ] Smoke tests passed - [ ] Monitoring dashboards open - [ ] Team on standby - [ ] Communication channels ready (Slack, email)

Launch (T-0): - [ ] Announce launch internally - [ ] Monitor metrics closely - [ ] Watch for errors and alerts - [ ] Be ready to rollback if needed

Post-Launch (T+1 hour): - [ ] Verify core functionality - [ ] Check user sign-ups - [ ] Monitor performance metrics - [ ] Address any immediate issues

Post-Launch (T+24 hours): - [ ] Review logs for errors - [ ] Analyze performance metrics - [ ] Collect user feedback - [ ] Plan hotfixes if needed - [ ] Celebrate success!

10.3 Post-Launch Support

First Week: - Daily standup to discuss issues - Monitor metrics dashboard - Respond to user feedback - Deploy hotfixes as needed

First Month: - Weekly review meetings - Analyze usage patterns - Optimize costs - Plan feature enhancements

Ongoing: - Monthly performance reviews - Quarterly security audits - Regular dependency updates - Continuous improvement

---

Document Version: 1.0 **Last Updated**: 2025-10-18 **Domain**: stemblock.ai **API URL**: api.stemblock.ai