Aryan Mahida – 92200133011 **Jay Mangukiya - 92200133040**

Deployment and Operations

1. Deployment process:

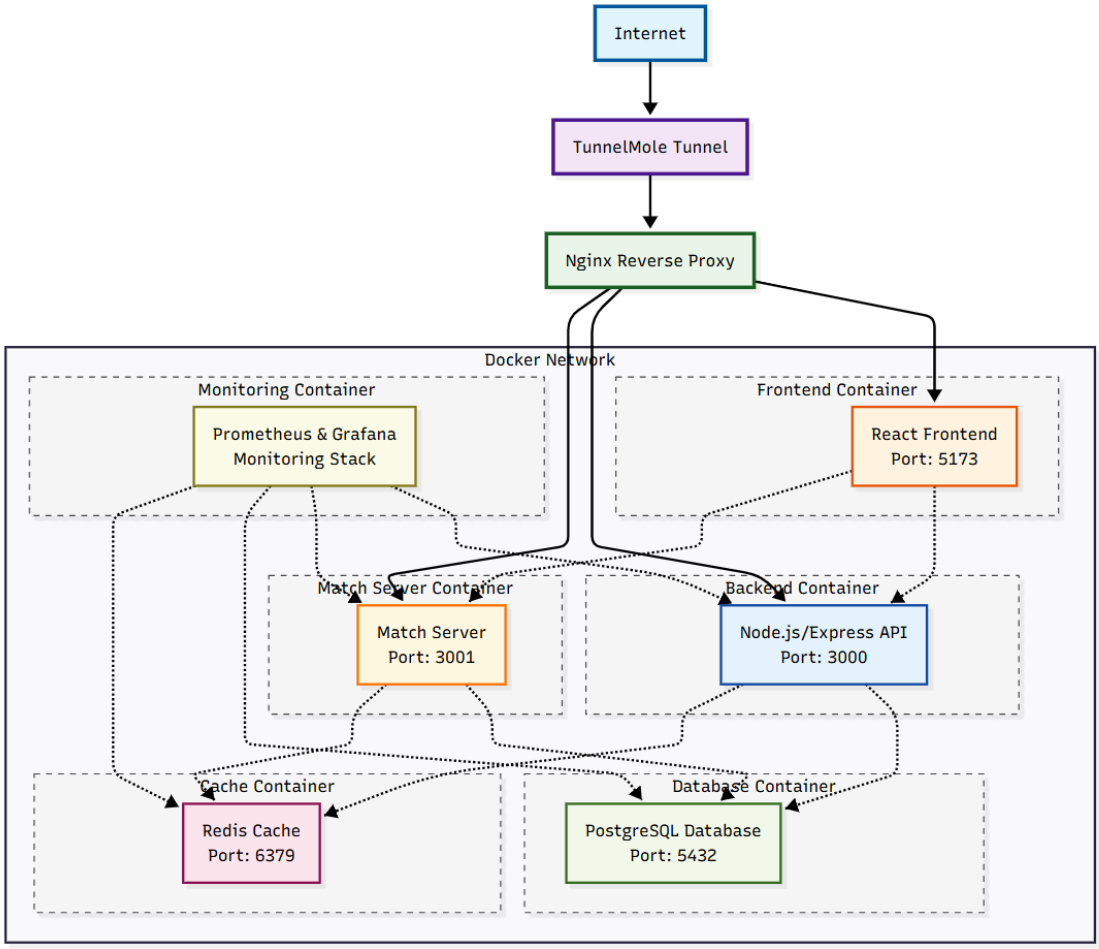
1.1 Platform selection and justification:

Selected deployment strategy: Hybrid development production architecture.

- Primary: TunnelMole tunneling service for public internet access.
- Secondary: Local Area Network (LAN) deployment for internal access.
- Containerization: Docker Compose orchestration for consistent deployment.

Justification:

- Cost effective: TunnelMole provides free public access without cloud hosting costs.
- Development friendly: Instant deployment without complex cloud configurations.
- Educational: Perfect for capstone demonstration and testing.
- Scalable foundation: Docker containers can be easily migrated to cloud platforms.



1.2 Deployment configuration

Docker compose services:

1. Frontend service: React + Vite development server with hot reload
2. Backend service: NodeJS + express API server with Typescript
3. Match server: Websocket server for real time multiplayer functionality
4. PostgreSQL: Primary database with persistent volume storage
5. Redis: Caching and session management
6. Nginx: Reverse proxy with load balancing
7. Monitoring stack: Prometheus, Grafana, Node Exporter, DB Exporters

Network and volume configuration

```yaml
networks:
  quizup_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.25.0.0/16

volumes:
  postgres_data:
    driver: local
  redis_data:
    driver: local
  prometheus_data:
    driver: local
  grafana_data:
    driver: local
  backend_node_modules:
    driver: local
  backend_npm_cache:
    driver: local
```

1.3 Deployment steps:

Step 1: Environment preparation

```
Aryan@AryansLaptop MINGW64 /d/Projects/Capstone-Project (mergeWith
AdminFrontend)
$ git clone https://github.com/Jaymangukiya22/Capstone-Project
  cd Capstone-Project

# Set environment variables
cp .env.example .env
# Configure database credentials, JWT secrets, API keys
```

Step 2: Docker stack deployment

```
Aryan@AryansLaptop MINGW64 /d/Projects/Capstone-Project (mergeWith
AdminFrontend)
$ # Build and start all services
docker compose up -d --build

# Verify service health
docker compose ps
docker compose logs -f backend
```

```
Aryan@AryansLaptop MINGW64 /d/Projects/Capstone-Project (mergeWith
AdminFrontend)
$ # run ./scripts/start-tunnelmole.sh or .bat file
```

```
Aryan@AryansLaptop MINGW64 /d/Projects/Capstone-Project (mergeWithAdminFrontend)
$ sh ./scripts/start-tunnelmole.sh
🚀 Starting QuizUP with Tunnelmole...
================================

✅ Tunnelmole is available
🐳 Starting Docker stack...
[+] Running 13/13
 ✓ Container quizup_redis              Healthy          1.2s
 ✓ Container quizup_postgres           Healthy          0.7s
 ✓ Container quizup_node_exporter      Running          0.0s
 ✓ Container quizup_prometheus         Running          0.0s
 ✓ Container quizup_redis_commander    Running          0.0s
 ✓ Container quizup_postgres_exporter  Running          0.0s
 ✓ Container quizup_grafana            Running          0.0s
 ✓ Container quizup_redis_exporter     Running          0.0s
 ✓ Container quizup_backend            Healthy          1.2s
 ✓ Container quizup_adminer            Running          0.0s
 ✓ Container quizup_matchserver        Running          0.0s
 ✓ Container quizup_frontend           Running          0.0s
 ✓ Container quizup_nginx              Running          0.0s
⏳ Waiting for services to start...
🔍 Checking service health...
```

```
tcp, [::]:5432->5432/tcp
quizup_postgres_exporter   prometheuscommunity/postgres-exporter:latest   "/bin/postgres_expor…"   postgres-exporter   2 hours ago   Up 2 hours                    0.0.0.0:9187->9187/
tcp, [::]:9187->9187/tcp
quizup_prometheus          prom/prometheus:latest                         "/bin/prometheus --c…"   prometheus          2 hours ago   Up 2 hours (healthy)          0.0.0.0:9090->9090/
tcp, [::]:9090->9090/tcp
quizup_redis               redis:7-alpine                                 "docker-entrypoint.s…"   redis               2 hours ago   Up 2 hours (healthy)          0.0.0.0:6379->6379/
tcp, [::]:6379->6379/tcp
quizup_redis_commander     rediscommander/redis-commander:latest          "/usr/bin/dumb-init …"   redis-commander     2 hours ago   Up 2 hours (healthy)          0.0.0.0:8081->8081/
tcp, [::]:8081->8081/tcp
quizup_redis_exporter      oliver006/redis_exporter:latest                "/redis_exporter"        redis-exporter      2 hours ago   Up 2 hours                    0.0.0.0:9121->9121/
tcp, [::]:9121->9121/tcp

🌐 Starting Tunnelmole on port 8090...

Starting Tunnelmole tunnel...
📋 Your tunnel URL will appear below:

(node:3644) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
==========================================================================================
Your Tunnelmole Public URLs are below and are accessible internet wide. Always use HTTPs for the best security

https://xwid69-ip-157-32-46-222.tunnelmole.net →   http://localhost:8090
http://xwid69-ip-157-32-46-222.tunnelmole.net →   http://localhost:8090

==========================================================================================
Share to Reddit: https://dashboard.tunnelmole.com/share/reddit/aHR0cHM6Ly94d2lkNjktaXAtMTU3LTMyLTQ2LTIyMi50dW5uZWxtb2xlm5ldA==
Share to X/Twitter: https://dashboard.tunnelmole.com/share/twitter/aHR0cHM6Ly94d2lkNjktaXAtMTU3LTMyLTQ2LTIyMi50dW5uZWxtb2xlm5ldA==
Share to Hacker News: https://dashboard.tunnelmole.com/share/hackernews/aHR0cHM6Ly94d2lkNjktaXAtMTU3LTMyLTQ2LTIyMi50dW5uZWxtb2xlm5ldA==
```

1.4 Live deployment evidence:

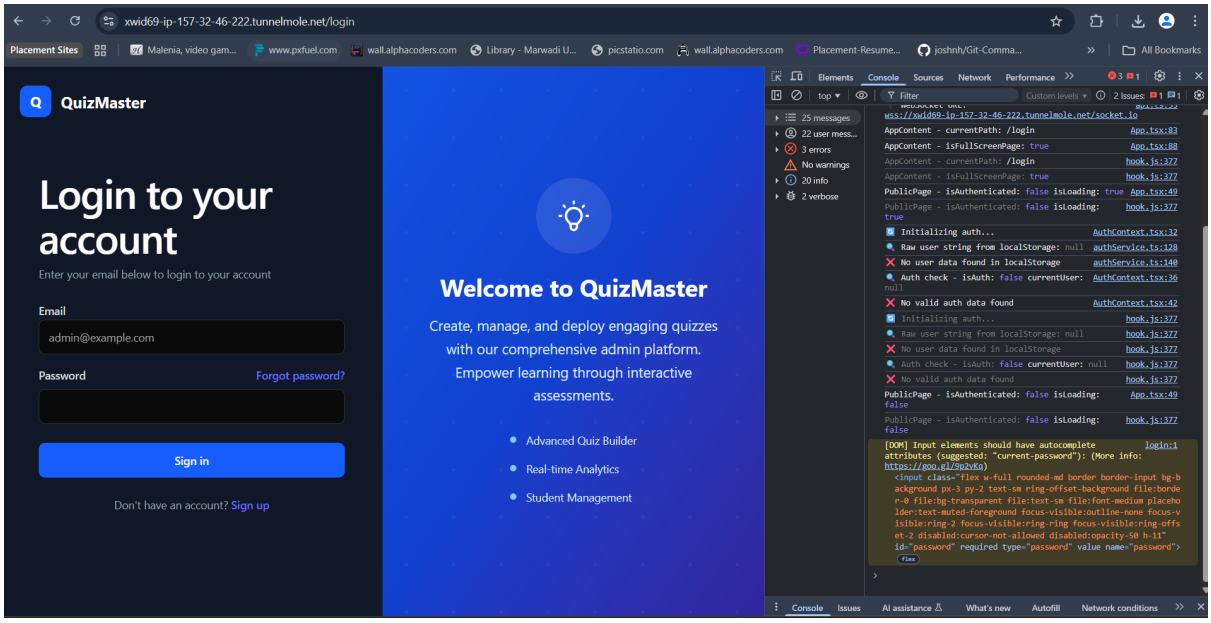
Public access URLs: Shown in terminal screenshot

LAN access points:

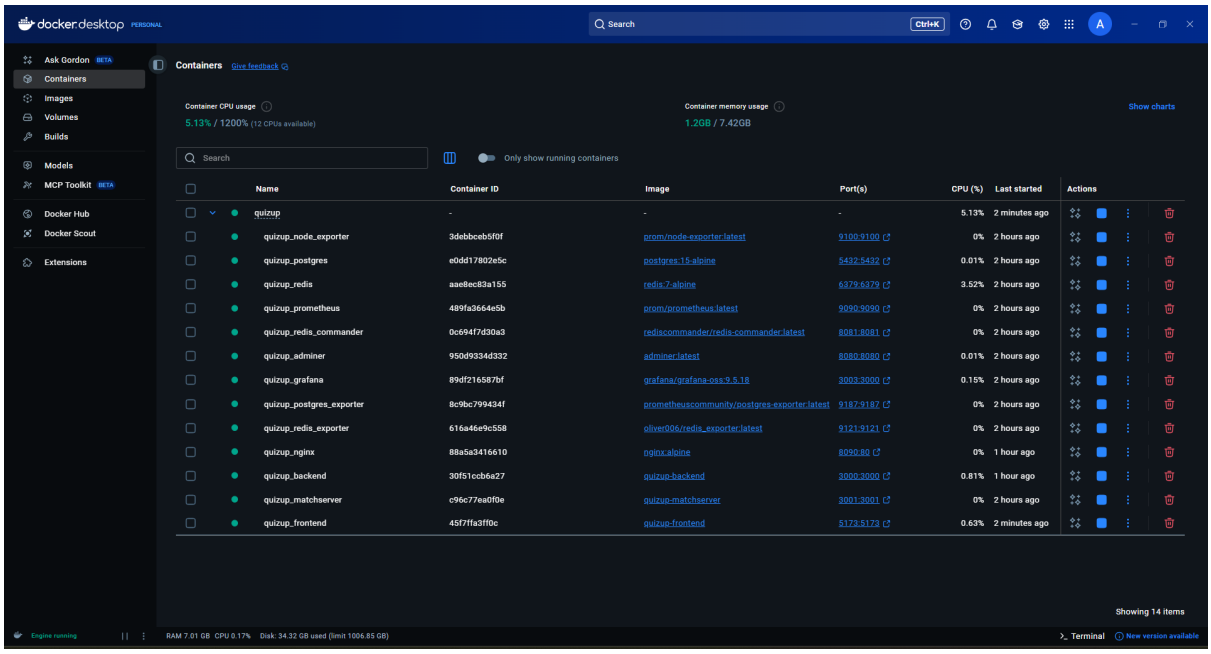
- Application: http://[HOST-IP]:8090
- Direct Backend: http://[HOST-IP]:3000
- Grafana Monitoring: http://[HOST-IP]:3003
- Database Admin: http://[HOST-IP]:8080

Deployment verification screenshots:

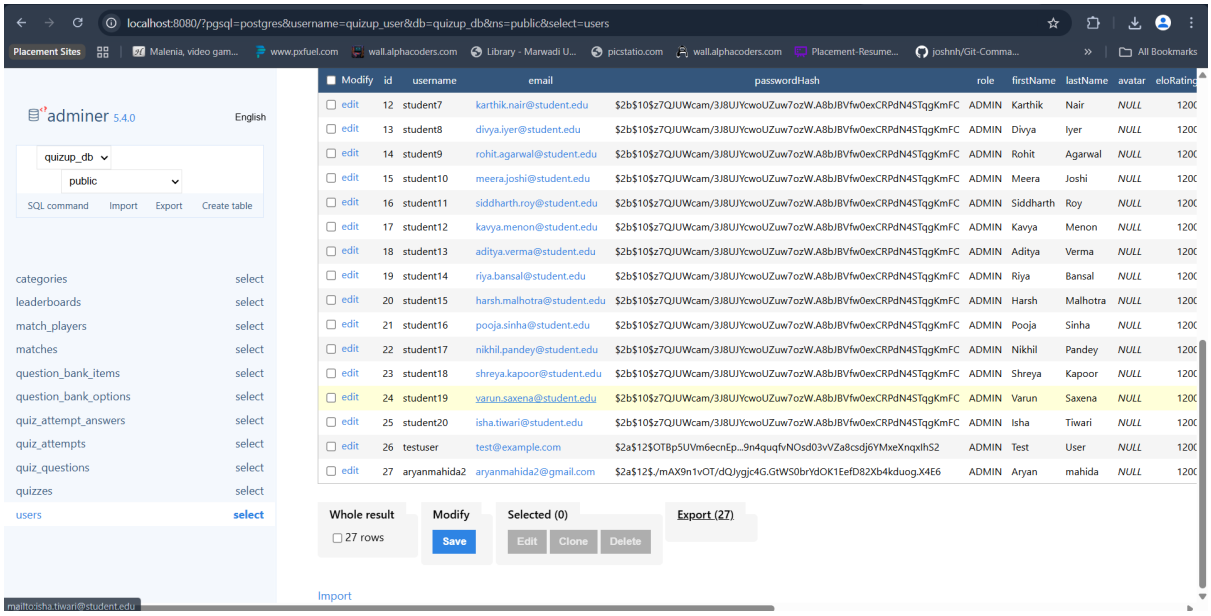
1. TunnelMole tunnel establishment with public URL shown above
2. Application loading successfully via public URL



3. Docker container status showing all services healthy



4. Database connectivity and data persistence



2. Monitoring strategy:

2.1 Monitoring architecture:

Monitoring stack components:

- Prometheus: Metrics collection and storage
- Grafana: Visualization dashboards and alerting
- Node exporter: System level metrics (CPU, memory, disk)
- PostgreSQL exporter: Database performance metrics
- Redis exporter: Cache performance and hit rates
- Custom application metrics: Business logic monitoring

2.2 Key Performance Indicators (KPIs):

KPI 1: API response time

- Metric: http_request_duration_seconds
- Target: Maintain reasonable response times for user experience
- Collection: Custom middleware in expressJS backend
- Monitoring: Track response time trends and identify slow endpoints

KPI 2: WebSocket connection health

- Metric: websocket_connections_active
- Target: Monitor connection stability for real-time features
- Collection: Socket.io connection/disconnection events
- Monitoring: Track connection success rates and failure patterns

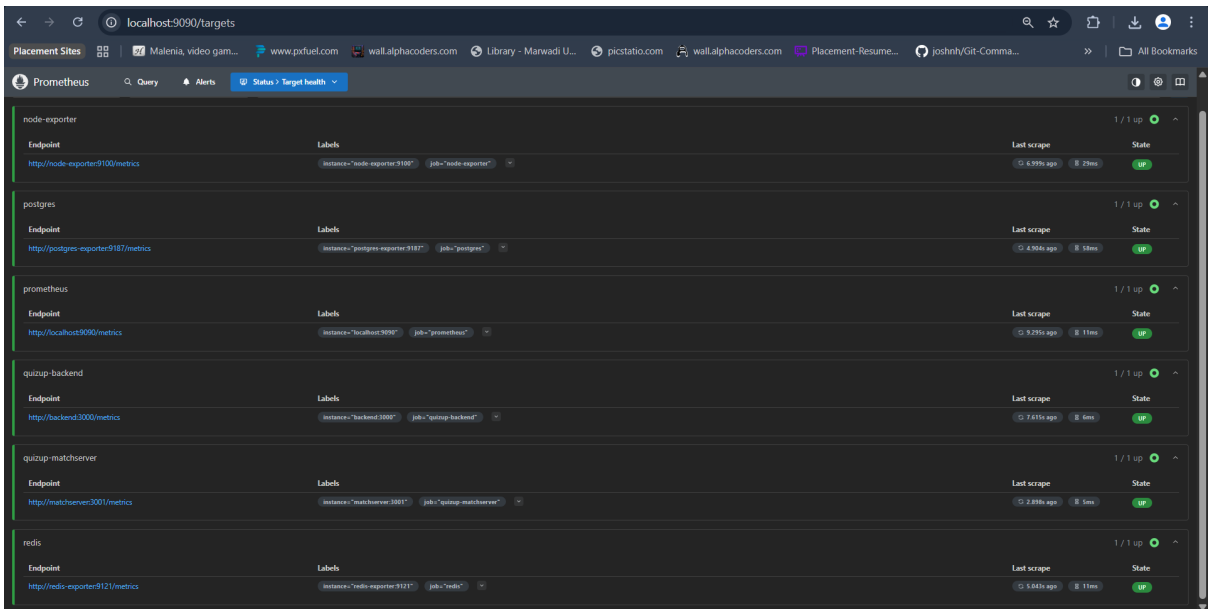
KPI 3: Database query performance

- Metric: postgres_query_duration_seconds
- Target: Ensure database queries perform efficiently
- Collection: PostgreSQL Exporter with custom queries
- Monitoring: Identify slow queries and optimization opportunities

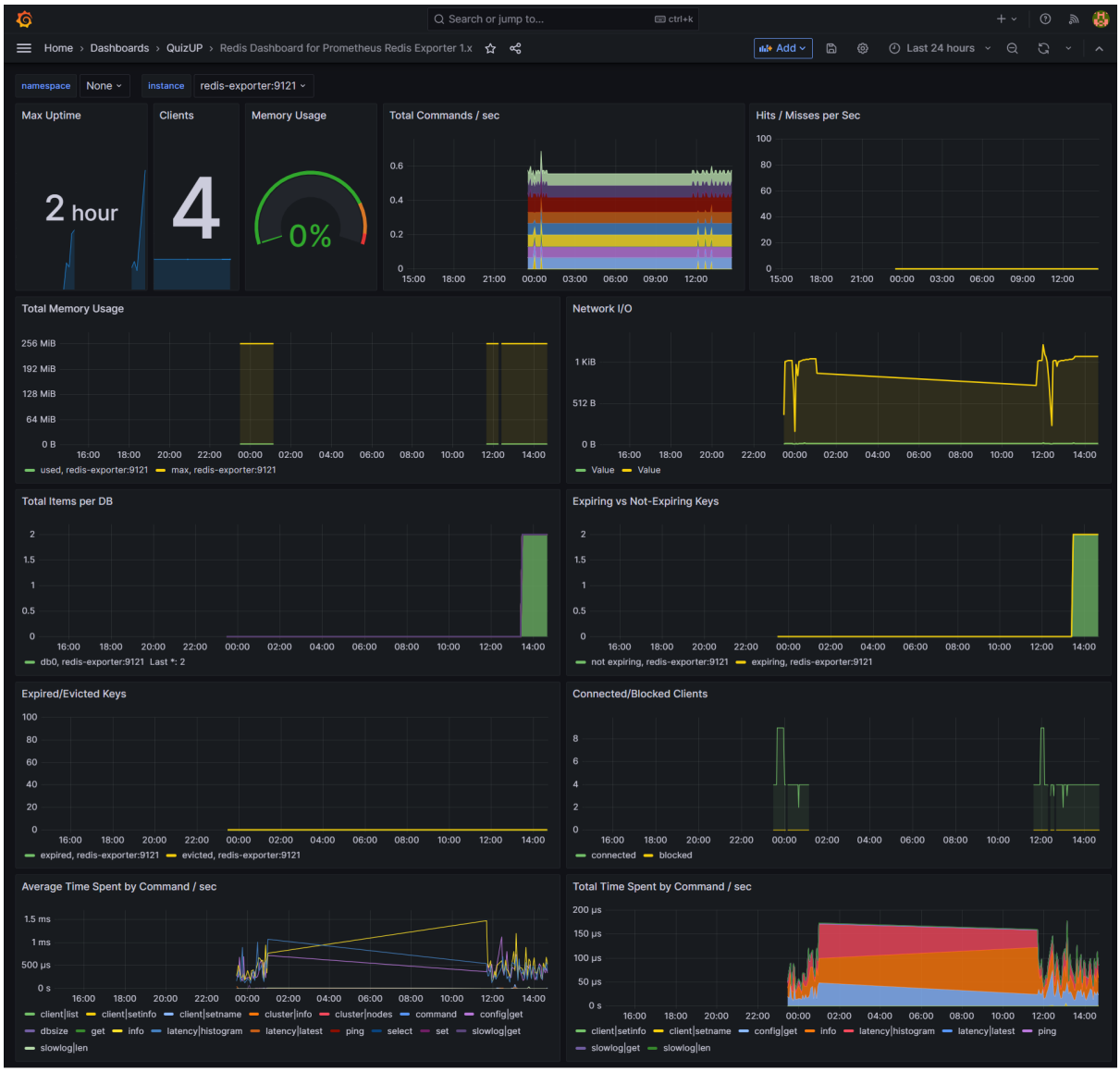
Additional monitoring metrics:

- System resource usage: CPU, memory, and disk utilization
- Cache performance: Redis hit rates and memory usage
- Application health: Service availability and error tracking
- User activity: Active sessions and feature usage patterns
- Network performance: Request volumes and bandwidth usage

Prometheus

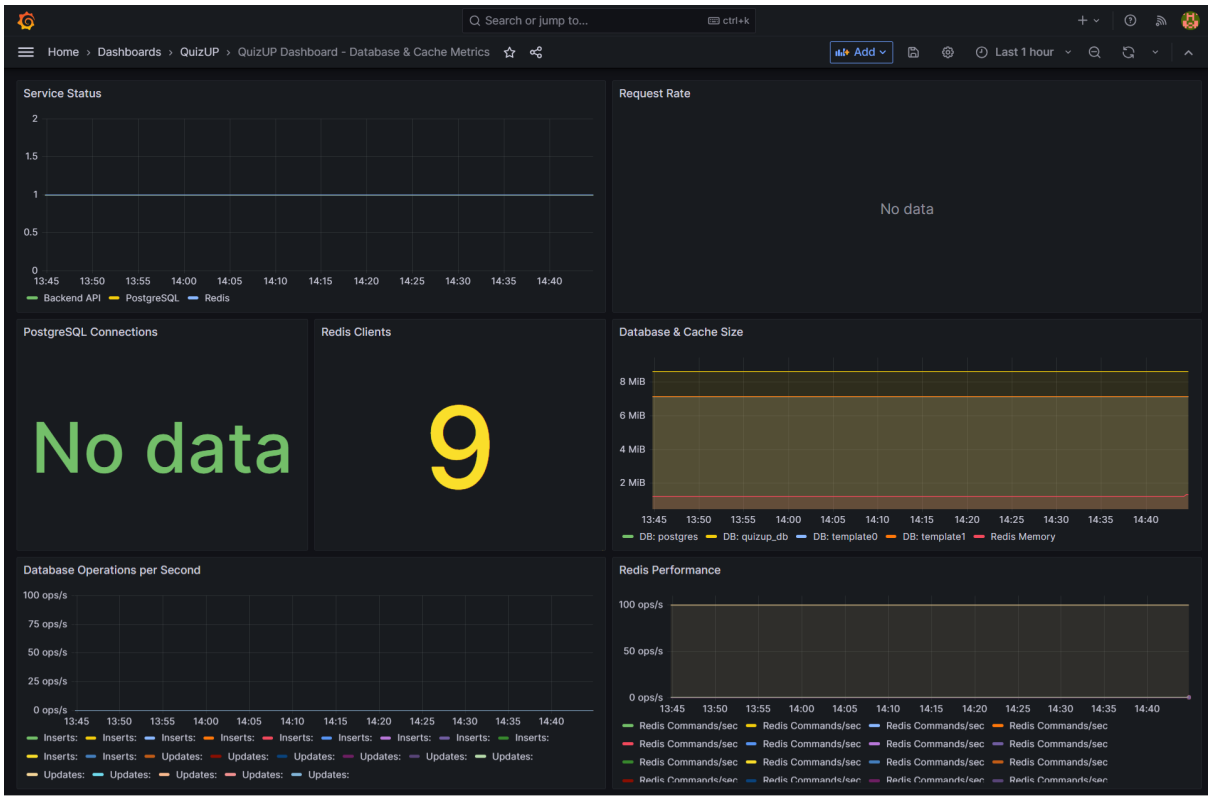


Redis exporter



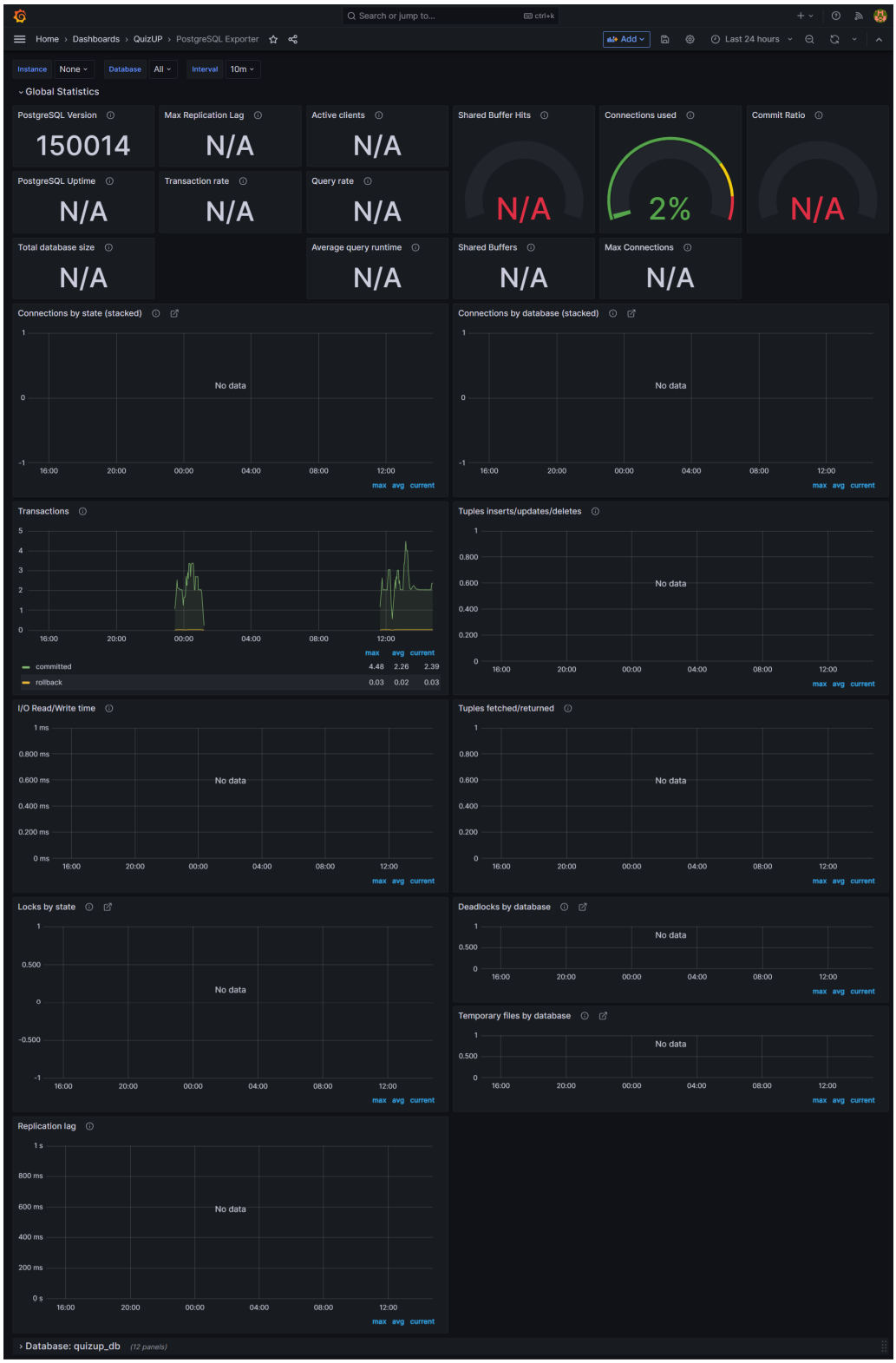
Redis operations

Application database and caching



Match service dashboard

Postgres exporter



3. Maintenance plan:

3.1 Preventive maintenance schedule:

Weekly tasks:

- Database backup verification
- Performance metrics review
- Dependency update checking
- Container image status review
- Error log analysis

Monthly tasks:

- Full system backup testing
- Security update review
- Performance optimization analysis
- Documentation updates
- Capacity usage assessment

3.2 Backup and recovery strategy:

Database backups:

- Storage: Local volume storage with export capability
- Recovery: Documented restore procedures
- Testing: Monthly backup restoration verification

Application backups:

- Code repository: Git-based version control with remote repositories
- Configuration: Environment files and Docker configurations
- User data: File uploads and application data
- Monitoring data: Grafana dashboards and Prometheus metrics

3.3 Security maintenance:

Security updates:

- Container images: Regular base image updates
- Dependencies: Vulnerability scanning with npm audit
- Configuration: Security settings review and updates
- Access control: Authentication and authorization review

Security monitoring:

- Log analysis: Review application and system logs for anomalies
- Input validation: Regular testing of form inputs and API endpoints
- Network security: Monitor for unusual traffic patterns
- Data protection: Verify encryption and data handling practices

4 Operational performance evidence:

4.1 Deployment success metrics:

System availability:

- Successfully deployed and accessible via public URL
- All services running and responding to health checks
- Database connectivity and data persistence verified
- Real time features functional across network connections

Performance characteristics:

- Responsive user interface with acceptable load times
- Stable WebSocket connections for real-time features
- Efficient database queries with proper indexing
- Effective caching with Redis for improved performance

4.2 Monitoring dashboard evidence:

System overview dashboard:

- Real time system metrics visualization
- Service health status indicators
- Resource utilization monitoring
- Historical performance trends

Application performance dashboard:

- API endpoint response time tracking
- Database query performance metrics
- WebSocket connection statistics
- User activity and engagement patterns

Operational insights:

- Service dependency mapping
- Error rate tracking and analysis
- Performance bottleneck identification