Azure Performance Issue Analysis

Problem Summary

The Azure-deployed site is experiencing 504 Gateway Timeout errors on static file requests (CSS, JS files), indicating the application is taking too long to respond.

Root Causes Identified

1. **Resource-Intensive Health Check** (PRIMARY ISSUE)

The `/health` endpoint in `app/main_saas.py` performs extensive operations:

- Creates database connections
- Instantiates agent factories
- Creates full agent graphs (LangGraph)
- Processes test messages through the entire agent pipeline
- Tests conversation management

Impact: Each health check can take 5-30+ seconds, consuming significant CPU/memory. Azure App Service health probes hit this endpoint repeatedly, keeping the app under constant load.

2. **Static Files Served Through Python Application**

All static files are routed through FastAPI's StaticFiles middleware:

```
app.mount("/saas", StaticFiles(directory="app/web/static/saas"),
name="saas_static")
```

Impact: Every CSS/JS request goes through:

- Python application
- All middleware (CORS, Tenant, RequestTiming with telemetry)
- File system access
- Logging/telemetry processing

This adds 100-500ms+ latency per static file vs. direct web server serving.

3. **Multiple Heavy Middlewares**

Request pipeline includes:

- CORS middleware
- Request timing with Application Insights telemetry
- Tenant identification middleware
- Each request triggers database queries and logging

Impact: Even simple static file requests process through all middleware layers.

4. **Azure App Service Configuration**

- `web.config` sets `requestTimeout="00:10:00"` but Azure's load balancer times out at ~60 seconds
- `startupTimeLimit="120"` may not be sufficient for slow-starting Python application
- Application runs on limited resources (basic/free tier likely)

Performance Metrics (Estimated)

- Health check: 5-30+ seconds
- Static file through Python: 100-500ms
- Static file direct serving: 5-20ms
- Health probe frequency: Every 30 seconds (default)

Solutions

IMMEDIATE FIXES (High Priority)

1. Simplify Health Check Endpoint

Replace the complex agent test with a simple status check:

```
@app.get("/health")
async def health_check():
    """Lightweight health check for load balancers."""
    return {
        "status": "healthy",
        "timestamp": time.time()
    }

@app.get("/health/detailed")
async def detailed_health_check():
    """Detailed health check with agent testing (manual use only)."""
    # Move existing complex health check code here
    ...
```

2. Configure Azure to Serve Static Files Directly

Add `.deployment` configuration to route static files through IIS/nginx instead of Python:

```
[config]
SCM_DO_BUILD_DURING_DEPLOYMENT = true
WEBSITE_RUN_FROM_PACKAGE = 1
```

Add `staticwebapp.config.json`:

```
{
  "routes": [
    {
      "route": "/saas/*",
      "rewrite": "/app/web/static/saas/"
    }
  ]
}
```

3. Optimize Middleware Chain

Add conditional middleware processing:

```
# Skip heavy middleware for static files
class ConditionalMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        # Skip processing for static files
        if request.url.path.startswith(('/static/', '/saas/')):
            return await call_next(request)

        # Normal processing for API routes
        return await heavy_processing(request, call_next)
```

4. Update Azure App Service Settings

Via Azure Portal or CLI:

- Increase App Service plan tier (at least B1 or higher)
- Set `WEBSITES_PORT=8000`
- Add `SCM_DO_BUILD_DURING_DEPLOYMENT=true`
- Configure health check to use `/health` endpoint
- Increase timeout: `requestTimeout="00:02:00"` (2 minutes)

MEDIUM-TERM IMPROVEMENTS

5. Implement Caching

- Add response caching for static files
- Use Azure CDN for static assets
- Implement Redis cache for frequently accessed data

6. Optimize Application Startup

- Lazy load heavy dependencies

- Move database migrations to separate deployment step
- Use connection pooling
- Implement async database operations

7. Add Application Monitoring

- Configure Application Insights properly
- Set up performance alerts
- Monitor memory/CPU usage patterns
- Track slow requests

LONG-TERM OPTIMIZATIONS

8. Separate Static Assets

- Move static files to Azure Blob Storage + CDN
- Use Azure Front Door for global distribution
- Implement asset versioning/cache busting

9. Microservices Architecture

- Separate agent processing into dedicated service
- Use Azure Functions for background tasks
- Implement message queue for async operations

10. Database Optimization

- Implement connection pooling with proper configuration
- Add database indexes for frequent queries
- Use read replicas for heavy read operations
- Optimize ORM queries (avoid N+1 problems)

Immediate Action Plan

1. **Deploy simplified health check** (5 minutes)
2. **Scale up App Service plan** to B1 or higher (2 minutes)
3. **Configure Azure static file serving** (10 minutes)
4. **Add conditional middleware** (15 minutes)
5. **Monitor and verify** (30 minutes)

Expected Improvements

- Health check: 5-30s → <100ms (99% improvement)
- Static files: 100-500ms → 5-20ms (90-95% improvement)
- Overall application responsiveness: 2-5x improvement
- Reduced 504 errors: 95%+ reduction
- Lower resource consumption: 50-70% reduction

Monitoring Checklist

- [ ] Health check response times (<100ms target)
- [ ] Static file load times (<50ms target)
- [ ] Application Insights error rates
- [ ] Memory consumption trends
- [ ] CPU utilization patterns
- [ ] Request queue lengths
- [ ] Failed request counts

Next Steps

1. Implement immediate fixes
2. Test in staging environment
3. Deploy to production with monitoring
4. Evaluate performance improvements
5. Plan medium-term optimizations based on metrics