Complete Performance Optimization Plan

Life Psychology Australia - Maximum Performance Configuration

---

🎯 Goal: Extract Every Millisecond of Performance

Target: Sub-1-second load times globally, 95+ Lighthouse scores, perceptibly instant user experience

Current Status:

- ✅ Azure Front Door (CDN) enabled - 40-70% faster
- ✅ Azure DNS - fast DNS resolution
- 🔄 Further optimizations available

🏗️ Layer 1: Infrastructure Optimization (Already Done ✅)

✅ Azure Front Door (Enterprise Grade Edge)

Impact: 40-70% faster load times

- Global CDN with 118+ edge locations
- Content cached near users worldwide
- Smart routing to fastest origin

✅ Azure DNS

Impact: 10-30ms faster DNS resolution

- Anycast network for fast lookups
- Global distribution

✅ HTTPS/TLS

Impact: Required for HTTP/2, security, trust

- Free SSL certificates from Azure
- Modern TLS 1.3 support

🔧 Layer 2: Front Door Configuration Optimization

Action 1: Enable Advanced Caching Rules

Navigate to: Azure Front Door → each endpoint → Caching

Configure:

1. **Query String Caching Behavior**:
   - Set to "Use Query String" for dynamic content
   - Set to "Ignore Query String" for static assets
   - **Impact**: Faster cache hits, reduced origin load
2. **Compression**:
   - Enable Gzip compression ✓
   - Enable Brotli compression ✓ (10-15% better than Gzip)
   - **Impact**: 50-70% smaller file sizes, faster transfers
3. **Caching Duration**:
   - Static assets (CSS, JS, images): 1 year (31536000 seconds)
   - HTML: 1 hour (3600 seconds) or versioned
   - API responses: Configure based on update frequency
   - **Impact**: More cache hits = faster loads

Steps:

```
Front Door → Endpoint → Caching
├── Enable compression: Gzip + Brotli
├── Set cache duration by file type
└── Configure query string behavior
```

Expected Improvement: 15-30% additional speed boost

Action 2: Enable HTTP/3 (QUIC Protocol)

Navigate to: Front Door → Configuration → Protocol Settings

Enable:

- ✅ HTTP/3 (latest, fastest protocol)
- ✅ HTTP/2 (fallback)
- ✅ HTTP/1.1 (legacy fallback)

Impact:

- 20-30% faster on mobile networks
- Better performance on lossy connections
- Reduced latency for subsequent requests

Steps:

```
Front Door → Configuration
└── Protocol Settings
    ├── HTTP/3: Enabled
    ├── HTTP/2: Enabled (fallback)
    └── TLS 1.3: Enabled
```

Expected Improvement: 20-30% on mobile, 10-15% on desktop

Action 3: Configure Rules Engine for Smart Routing

Navigate to: Front Door → Rules engine

Create Rules:

Rule 1: Force HTTPS

```
IF: Request protocol = HTTP
THEN: Redirect to HTTPS (301)
```

Impact: Security + HTTP/2/3 benefits

Rule 2: Add Security Headers

```
Add Response Headers:
- Strict-Transport-Security: max-age=31536000
- X-Content-Type-Options: nosniff
- X-Frame-Options: SAMEORIGIN
- Referrer-Policy: strict-origin-when-cross-origin
```

Impact: Security + trust signals

Rule 3: Cache Control Headers

```
IF: Path matches *.css, *.js, *.jpg, *.png, *.woff2
THEN: Add header Cache-Control: public, max-age=31536000, immutable
```

Impact: Browser caching, instant repeat visits

Rule 4: Preconnect Headers

```
Add Response Header:
- Link: <https://fonts.googleapis.com>; rel=preconnect
```

Impact: Faster third-party resource loading

Expected Improvement: 10-20% for returning users, better security

---

Action 4: Configure Origin Timeouts and Health Probes

Navigate to: Front Door → Origin Groups → Health Probes

Configure:

- **Probe interval**: 30 seconds
- **Probe path**: `/` or health endpoint
- **Timeout**: 5 seconds
- **Healthy threshold**: 2 successful probes

Origin Settings:

- **Connection timeout**: 30 seconds (reduce if your app is fast)
- **Send/receive timeout**: 30 seconds

Impact: Faster failover, better reliability

🎨 Layer 3: Static Web App Build Optimization

Action 5: Optimize Build Configuration

For React/Next.js/Vue Apps:

Add to build configuration:

```
json
```

```
{
  "optimization": {
    "minimize": true,
    "splitChunks": {
      "chunks": "all",
      "cacheGroups": {
        "vendor": {
          "test": /[\\/]node_modules[\\/]/,
          "priority": -10
        }
      }
    }
  },
  "performance": {
    "maxAssetSize": 250000,
    "maxEntrypointSize": 250000,
    "hints": "warning"
  }
}
```

Impact:

- Smaller bundle sizes (30-50% reduction)
- Code splitting = faster initial load
- Vendor chunking = better caching

Action 6: Image Optimization

Implement:

1. **Use Next-Gen Formats**:
   - Convert to WebP (30% smaller than JPEG)
   - Use AVIF where supported (50% smaller than JPEG)
   - Fallback to JPEG/PNG for legacy browsers
2. **Lazy Loading**:

```
html

  <img src="image.jpg" loading="lazy" />
```

Impact: Only load images when visible

3. **Responsive Images**:

```
html

  <img
    srcset="small.jpg 400w, medium.jpg 800w, large.jpg 1200w"
    sizes="(max-width: 600px) 400px, (max-width: 900px) 800px, 1200px"
  />
```

Impact: Serve appropriately sized images

4. **Image Compression**:
   - Use tools like ImageOptim, Squoosh, or Sharp
   - Target: 80-85% quality (imperceptible loss)
   - **Impact**: 40-60% file size reduction

Expected Improvement: 30-50% faster image loading

Action 7: Font Optimization

Implement:

1. **Font Display Strategy**:

```
css

  @font-face {
    font-family: 'YourFont';
    font-display: swap; /* Show fallback immediately */
    src: url('/fonts/font.woff2') format('woff2');
  }
```

2. **Preload Critical Fonts**:

```
html

  <link rel="preload" href="/fonts/main.woff2" as="font" type="font/woff2" crossorigin>
```

3. **Use WOFF2 Format Only** (95%+ browser support):
   - 30% smaller than WOFF
   - 50% smaller than TTF

4. **Subset Fonts** (if custom fonts):
   - Only include used characters

- Can reduce size by 70-90%

Expected Improvement: Eliminate font-loading flicker, 200-500ms faster text rendering

---

Action 8: Critical CSS Inlining

Implement:

1. **Inline Above-the-Fold CSS**:

```
html

<style>
  /* Critical CSS for hero section, header */
</style>
```

2. **Defer Non-Critical CSS**:

```
html

<link rel="preload" href="styles.css" as="style" onload="this.onload=null;this.rel='stylesheet'">
```

Tools:

- Critical (npm package)
- PurgeCSS (remove unused CSS)

Expected Improvement: 500-1000ms faster first paint

Action 9: JavaScript Optimization

Implement:

1. **Code Splitting**:

```
javascript

// Route-based splitting
const Component = lazy(() => import('./Component'));
```

2. **Tree Shaking**:
   - Remove unused code
   - Use ES6 modules

- Configure webpack/vite properly

3. **Defer Non-Critical JS**:

```html
<script defer src="non-critical.js"></script>
```

4. **Minimize Third-Party Scripts**:
   - Audit: Google Analytics, chat widgets, etc.
   - Load asynchronously
   - Consider alternatives (e.g., Plausible vs Google Analytics)

Expected Improvement: 30-50% faster JavaScript execution

📊 Layer 4: Monitoring & Continuous Optimization

Action 10: Set Up Application Insights

Navigate to: Azure Portal → Create Application Insights

Configure:

1. Create Application Insights resource
2. Connect to both Static Web Apps
3. Enable:
   - Real User Monitoring (RUM)
   - Performance tracking
   - Dependency tracking
   - Custom events

Benefits:

- See actual user load times
- Identify slow pages/components
- Track performance over time
- Geographic performance breakdown

Cost: ~$2-10/month depending on traffic

Action 11: Implement Lighthouse CI

Add to your CI/CD pipeline:

```yaml
# GitHub Actions example
- name: Lighthouse CI
  run: |
    npm install -g @lhci/cli
    lhci autorun
```

Configure target scores:

```json
{
  "ci": {
    "assert": {
      "assertions": {
        "categories:performance": ["error", {"minScore": 0.95}],
        "categories:accessibility": ["error", {"minScore": 0.95}],
        "first-contentful-paint": ["error", {"maxNumericValue": 1500}],
        "speed-index": ["error", {"maxNumericValue": 2000}],
        "largest-contentful-paint": ["error", {"maxNumericValue": 2500}]
      }
    }
  }
}
```

Impact: Prevent performance regressions, enforce standards

Action 12: Set Up Performance Budgets

Configure in build pipeline:

```json
```

```
{
 "budgets": [
  {
   "type": "bundle",
   "name": "main",
   "baseline": "250kb",
   "maximumWarning": "300kb",
   "maximumError": "350kb"
  },
  {
   "type": "initial",
   "maximumWarning": "500kb",
   "maximumError": "600kb"
  }
 ]
}
```

Impact: Prevent bundle bloat over time

🌐 Layer 5: DNS & Network Optimization

Action 13: Optimize DNS Records

Current TTLs:

- Most records: 4 hours (14400 seconds)
- Nameservers: 48 hours (172800 seconds)

Recommended Changes (After propagation complete):

For frequently accessed records:

```
bloom → 1 hour (3600 seconds)
www → 1 hour (3600 seconds)
@ (root) → 1 hour (3600 seconds)
```

Impact: Faster DNS updates, minimal performance trade-off

For stable records:

```
MX → Keep at 1 hour (3600 seconds) ✓
TXT → Keep at 4 hours (14400 seconds) ✓
NS → Keep at 48 hours (172800 seconds) ✓
```

Lower TTL Benefits:

- Faster DNS propagation for changes
- More flexibility **Costs**:
- Slightly more DNS queries (negligible with caching)

Action 14: Implement DNS Prefetching & Preconnect

Add to HTML `<head>`:

```
html

<!-- DNS Prefetch for third-party domains -->
<link rel="dns-prefetch" href="//fonts.googleapis.com">
<link rel="dns-prefetch" href="//www.google-analytics.com">

<!-- Preconnect for critical third-parties -->
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<!-- Preload critical assets -->
<link rel="preload" href="/fonts/main.woff2" as="font" type="font/woff2" crossorigin>
<link rel="preload" href="/critical.css" as="style">
```

Impact: 100-300ms faster for third-party resources

🔒 Layer 6: Advanced Security & Performance

Action 15: Implement Content Security Policy (CSP)

Add via Front Door Rules Engine or meta tag:

```
html

<meta http-equiv="Content-Security-Policy"
    content="default-src 'self';
            script-src 'self' 'unsafe-inline' https://cdn.example.com;
            style-src 'self' 'unsafe-inline' https://fonts.googleapis.com;
            font-src 'self' https://fonts.gstatic.com;
            img-src 'self' data: https:;">
```

Impact:

- Security (prevent XSS)

- Forces HTTPS for all resources
- Better trust signals

Action 16: Enable Resource Hints

Add to HTML:

```html
<!-- Prefetch next page user is likely to visit -->
<link rel="prefetch" href="/clinician-signup">

<!-- Prerender critical pages (use sparingly) -->
<link rel="prerender" href="/booking">
```

Impact: Instant navigation to prefetched pages

🎯 Layer 7: Progressive Web App (PWA) Features

Action 17: Implement Service Worker

Create `service-worker.js`:

```javascript
// Cache-first strategy for static assets
self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((response) => {
      return response || fetch(event.request);
    })
  );
});
```

Benefits:

- Offline capability
- Instant repeat visits (cached)
- App-like experience

Impact: 500-1000ms faster for returning users

Action 18: Add Web App Manifest

Create `manifest.json`:

```
json

{
  "name": "Life Psychology Australia",
  "short_name": "LPA",
  "icons": [
    {
      "src": "/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "/icon-512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ],
  "theme_color": "#your-brand-color",
  "background_color": "#ffffff",
  "display": "standalone",
  "start_url": "/"
}
```

Link in HTML:

```
html

<link rel="manifest" href="/manifest.json">
```

Benefits:

- Installable to home screen
- App-like experience
- Better engagement

📈 Expected Performance Improvements

Current Status (After Front Door)

- **Load Time**: ~2-3 seconds (estimate)

- **Lighthouse Score**: ~70-80 (estimate)
- **Time to Interactive**: ~3-4 seconds

After Full Optimization

- **Load Time**: <1 second (first visit), <500ms (repeat)
- **Lighthouse Score**: 95+ across all categories
- **Time to Interactive**: <1.5 seconds
- **First Contentful Paint**: <800ms
- **Largest Contentful Paint**: <1.5 seconds

Cumulative Impact by Layer

Layer	Improvement	Cumulative Total
Layer 1: Infrastructure (Done ✅)	40-70%	40-70%
Layer 2: Front Door Config	15-30%	55-85%
Layer 3: Build Optimization	30-50%	70-95%
Layer 4: Monitoring	0-5% (prevents regression)	70-95%
Layer 5: DNS & Network	5-10%	75-97%
Layer 6: Security Headers	0-5%	75-97%
Layer 7: PWA	20-40% (repeat visits)	80-98%

Overall Expected Improvement from baseline: **80-98% faster**

🛠️ Implementation Priority

Priority 1: High Impact, Low Effort (Do First)

1. ✅ Front Door caching + compression (Action 1)
2. ✅ HTTP/3 enablement (Action 2)
3. ✅ Image optimization (Action 6)
4. ✅ Font optimization (Action 7)

Expected Improvement: 40-60% additional boost

Time to Implement: 2-4 hours

Priority 2: High Impact, Medium Effort (Do Next)

5. ✅ Front Door Rules Engine (Action 3)
6. ✅ Critical CSS inlining (Action 8)
7. ✅ JavaScript optimization (Action 9)
8. ✅ Resource hints (Action 14)

Expected Improvement: 20-30% additional boost
Time to Implement: 4-8 hours

Priority 3: Medium Impact, Low Effort (Quick Wins)

9. ✅ Security headers (Action 3)
10. ✅ DNS prefetching (Action 14)
11. ✅ Application Insights (Action 10)

Expected Improvement: 5-15% additional boost
Time to Implement: 1-2 hours

Priority 4: Long-term Optimization (Ongoing)

12. ✅ Lighthouse CI (Action 11)
13. ✅ Performance budgets (Action 12)
14. ✅ PWA implementation (Actions 17-18)

Expected Improvement: Prevents regression, improves over time
Time to Implement: 4-8 hours initial, ongoing monitoring

🎯 Quick Start: Do These 5 Things First

1. Enable Compression in Front Door (5 minutes)

```
Front Door → Caching → Enable Gzip + Brotli
```

Impact: 50-70% smaller files immediately

2. Enable HTTP/3 (2 minutes)

```
Front Door → Protocol Settings → Enable HTTP/3
```

Impact: 20-30% faster on mobile

3. Add Security + Cache Headers via Rules Engine (15 minutes)

```
Front Door → Rules Engine → Add caching and security headers
```

Impact: Better caching, security, performance

4. Optimize Images (30-60 minutes)

- Convert to WebP
- Add lazy loading
- Compress **Impact**: 30-50% faster image loading

5. Add Resource Hints to HTML (10 minutes)

```html
<link rel="dns-prefetch" href="//fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.googleapis.com">
```

Impact: 100-300ms faster third-party loading

Total Time: ~1-2 hours

Total Impact: 60-80% improvement over current

📊 How to Measure Success

Tools to Use:

1. **Google PageSpeed Insights**: https://pagespeed.web.dev
   - Target: 95+ score on both mobile and desktop
2. **WebPageTest**: https://www.webpagetest.org
   - Target: <1s load time from Sydney
3. **Lighthouse** (Chrome DevTools):
   - Target: All categories >95
4. **Azure Application Insights**:

- Monitor real user performance
- Track improvements over time

Key Metrics to Track:

- **First Contentful Paint (FCP)**: <800ms
- **Largest Contentful Paint (LCP)**: <1.5s
- **Time to Interactive (TTI)**: <1.5s
- **Cumulative Layout Shift (CLS)**: <0.1
- **Total Blocking Time (TBT)**: <200ms

💰 Cost Impact

All optimizations listed are included in existing services or free:

Optimization	Additional Cost
Front Door config	$0 (included)
Build optimization	$0 (one-time dev effort)
Application Insights	~$2-10/month
Image optimization	$0 (one-time effort)
PWA features	$0 (one-time dev effort)

Total Additional Monthly Cost: ~$2-10 (Application Insights only)

✅ Action Plan Summary

Week 1 (After DNS Propagates):

1. Enable Front Door compression + HTTP/3
2. Configure caching rules
3. Add security headers
4. Set up Application Insights

Week 2:

1. Optimize images (WebP, compression, lazy loading)
2. Optimize fonts (WOFF2, preload, font-display)

3. Add resource hints
4. Critical CSS inlining

Week 3:

1. Implement service worker
2. Add PWA manifest
3. Set up Lighthouse CI
4. Configure performance budgets

Week 4:

1. Monitor and measure
2. Fine-tune based on real user data
3. Document optimizations
4. Train team on maintaining performance

🎓 Additional Resources

Performance Best Practices:

- web.dev/fast
- https://developer.mozilla.org/en-US/docs/Web/Performance

Azure Front Door Optimization:

- https://docs.microsoft.com/en-us/azure/frontdoor/front-door-caching

Image Optimization Tools:

- Squoosh: https://squoosh.app
- Sharp: https://sharp.pixelplumbing.com

Performance Testing:

- Lighthouse: https://developers.google.com/web/tools/lighthouse
- WebPageTest: https://www.webpagetest.org

Remember: Performance is a feature, not a checkbox. These optimizations will make your sites perceptibly faster, improve user experience, increase conversions, and reflect your practice's commitment to excellence.

Every millisecond counts. Let's make it blazing fast. ⚡