```
// Function to track server operations
const trackServerOperation = (operation, status) => {
serverOperations.labels(operation, status).inc();
};

// Function to track health checks
const trackHealthCheck = (serverId, status, duration) => {
healthCheckDuration.labels(serverId, status).observe(duration / 1000);
};

// Function to track errors
const trackError = (type, severity) => {
errorCount.labels(type, severity).inc();
};

// Function to update active connections
const updateActiveConnections = (count) => {
activeConnections.set(count);
};

// Metrics endpoint
const getMetrics = async () => {
return await register.metrics();
};

module.exports = {
metricsMiddleware,
trackServerOperation,
trackHealthCheck,
trackError,
updateActiveConnections,
getMetrics
};
```

~~~
**Error Tracking Integration:**
```javascript
// error-tracking/sentry.js
const Sentry = require('@sentry/node');
const { ProfilingIntegration } = require('@sentry/profiling-node');

// Initialize Sentry
Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV,
  integrations: [
    new ProfilingIntegration(),
  ],
  // Performance Monitoring
  tracesSampleRate: process.env.NODE_ENV === 'production' ? 0.1 : 1.0,
  // Profiling
  profilesSampleRate: process.env.NODE_ENV === 'production' ? 0.1 : 1.0,
});

// Error handling middleware
const errorHandler = (error, req, res, next) => {
  // Log error to console
  console.error('Error:', error);

  // Track error in metrics
  const { trackError } = require('../metrics/metrics');
  trackError(error.name || 'UnknownError', 'error');

  // Send to Sentry
  Sentry.captureException(error, {
    tags: {
      component: 'api',
      endpoint: req.path,
      method: req.method
    },
    user: {
      ip_address: req.ip,
      id: req.session?.userId
    },
    extra: {
      body: req.body,
      query: req.query,
~~~

```
      params: req.params
    }
  });

  // Send error response
  if (res.headersSent) {
    return next(error);
  }

  const statusCode = error.statusCode || 500;
  const message = process.env.NODE_ENV === 'production'
    ? 'Internal Server Error'
    : error.message;

  res.status(statusCode).json({
    success: false,
    error: message,
    ...(process.env.NODE_ENV !== 'production' && { stack: error.stack })
  });
};

// Context enrichment
const enrichContext = (req, res, next) => {
  Sentry.configureScope(scope => {
    scope.setTag('route', req.route?.path);
    scope.setContext('request', {
      method: req.method,
      url: req.url,
      headers: req.headers,
      ip: req.ip
    });
  });
  next();
};

module.exports = {
  errorHandler,
  enrichContext,
  Sentry
};
```

**Usage Analytics:**

```
javascript
```

```javascript
// analytics/analytics.js
class AnalyticsService {
  constructor() {
    this.events = [];
    this.maxEvents = 10000;
    this.flushInterval = 60000; // 1 minute
    this.startFlushTimer();
  }

  track(event, properties = {}, userId = null) {
    const eventData = {
      event,
      properties: {
        ...properties,
        timestamp: new Date().toISOString(),
        sessionId: properties.sessionId,
        userAgent: properties.userAgent,
        ip: properties.ip
      },
      userId
    };

    this.events.push(eventData);

    // Flush if buffer is full
    if (this.events.length >= this.maxEvents) {
      this.flush();
    }
  }

  // Common events
  trackServerCreated(serverId, serverType, userId, properties = {}) {
    this.track('server_created', {
      serverId,
      serverType,
      templateUsed: properties.templateUsed,
      ...properties
    }, userId);
  }

  trackServerStarted(serverId, userId, properties = {}) {
    this.track('server_started', {
      serverId,
      startTime: properties.startTime,
      ...properties
    }, userId);
  }

  trackTemplateDeployed(templateId, serverId, userId, properties = {}) {
    this.track('template_deployed', {
      templateId,
      serverId,
      deploymentTime: properties.deploymentTime,
      configurationComplexity: properties.configurationComplexity,
      ...properties
    }, userId);
  }

  trackPageView(page, userId, properties = {}) {
    this.track('page_view', {
      page,
      referrer: properties.referrer,
      loadTime: properties.loadTime,
      ...properties
    }, userId);
  }

  trackError(error, userId, properties = {}) {
    this.track('error_occurred', {
      errorType: error.name,
      errorMessage: error.message,
      stackTrace: error.stack,
      component: properties.component,
      ...properties
    }, userId);
  }

  async flush() {
    if (this.events.length === 0) return;

    const eventsToFlush = [...this.events];
    this.events = [];

    try {
      // Send to analytics service (example: PostHog, Mixpanel, etc.)
      if (process.env.ANALYTICS_ENDPOINT) {
        await fetch(process.env.ANALYTICS_ENDPOINT, {
```

```
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${process.env.ANALYTICS_TOKEN}`
        },
        body: JSON.stringify({
          events: eventsToFlush
        })
      });
    }

    // Store in local database for backup
    await this.storeLocally(eventsToFlush);

  } catch (error) {
    console.error('Failed to flush analytics events:', error);
    // Put events back in queue
    this.events.unshift(...eventsToFlush);
  }
}

async storeLocally(events) {
  // Store in SQLite for local backup and reporting
  const db = require('../database/Database');
  const database = new db();

  for (const event of events) {
    await database.insertAnalyticsEvent(event);
  }
}

startFlushTimer() {
  setInterval(() => {
    this.flush();
  }, this.flushInterval);
}

// Generate usage reports
async generateUsageReport(timeRange = '30d') {
  const db = require('../database/Database');
  const database = new db();

  const report = {
    totalUsers: await database.getUniqueUsersCount(timeRange),
      totalServers: await database.getTotalServersCreated(timeRange),
      popularTemplates: await database.getPopularTemplates(timeRange),
      averageSessionDuration: await database.getAverageSessionDuration(timeRange),
      errorRate: await database.getErrorRate(timeRange),
      featureUsage: await database.getFeatureUsage(timeRange)
    };

    return report;
  }
}

// Analytics middleware for Express
const analyticsMiddleware = (req, res, next) => {
  // Track page views for frontend routes
  if (req.method === 'GET' && req.path.startsWith('/api')) {
    const analytics = req.app.get('analytics');
    analytics.trackPageView(req.path, req.session?.userId, {
      userAgent: req.get('User-Agent'),
      ip: req.ip,
      referrer: req.get('Referrer')
    });
  }

  next();
};

module.exports = {
  AnalyticsService,
  analyticsMiddleware
};
```

**Health Dashboard:**

javascript

```
// health/dashboard.js
class HealthDashboard {
  constructor(database, analytics) {
    this.db = database;
    this.analytics = analytics;
  }

  async getSystemHealth() {
    const now = new Date();
    const oneHourAgo = new Date(now.getTime() - 60 * 60 * 1000);

    return {
      application: await this.getApplicationHealth(),
      servers: await this.getServersHealth(),
      infrastructure: await this.getInfrastructureHealth(),
      performance: await this.getPerformanceMetrics(oneHourAgo, now),
      errors: await this.getErrorMetrics(oneHourAgo, now)
    };
  }

  async getApplicationHealth() {
    const uptime = process.uptime();
    const memoryUsage = process.memoryUsage();

    return {
      status: 'healthy',
      uptime: uptime,
      memory: {
        used: memoryUsage.heapUsed,
        total: memoryUsage.heapTotal,
        external: memoryUsage.external,
        usage: Math.round((memoryUsage.heapUsed / memoryUsage.heapTotal) * 100)
      },
      cpu: await this.getCpuUsage(),
      version: process.env.npm_package_version || '1.0.0',
      environment: process.env.NODE_ENV
    };
  }

  async getServersHealth() {
    const servers = await this.db.getAllServers();
    const healthySérvers = servers.filter(s => s.healthStatus === 'healthy');
    const unhealthyServers = servers.filter(s => s.healthStatus === 'unhealthy');
    const runningServers = servers.filter(s => s.status === 'running');

    return {
      total: servers.length,
      running: runningServers.length,
      healthy: healthySérvers.length,
      unhealthy: unhealthyServers.length,
      healthRate: servers.length > 0 ? (healthySérvers.length / servers.length) * 100 : 0
    };
  }

  async getInfrastructureHealth() {
    const checks = [];

    // Database check
    try {
      await this.db.healthCheck();
      checks.push({ component: 'database', status: 'healthy' });
    } catch (error) {
      checks.push({ component: 'database', status: 'unhealthy', error: error.message });
    }

    // Docker check
    try {
      const docker = require('../services/DockerService');
      const dockerService = new docker();
      await dockerService.ping();
      checks.push({ component: 'docker', status: 'healthy' });
    } catch (error) {
      checks.push({ component: 'docker', status: 'unhealthy', error: error.message });
    }

    // File system check
    try {
      const fs = require('fs').promises;
      await fs.access('/app/data', fs.constants.W_OK);
      checks.push({ component: 'filesystem', status: 'healthy' });
    } catch (error) {
      checks.push({ component: 'filesystem', status: 'unhealthy', error: error.message });
    }

    return {
      checks,
      overallStatus: checks.every(c => c.status === 'healthy') ? 'healthy' : 'degraded'
    };
  }

  async getPerformanceMetrics(startTime, endTime) {
    // These would come from Prometheus metrics in a real implementation
    return {
      averageResponseTime: 150, // ms
      requestsPerSecond: 25,
      throughput: 1500, // requests/hour
      p95ResponseTime: 300, // ms
      p99ResponseTime: 500 // ms
    };
  }

  async getErrorMetrics(startTime, endTime) {
    const totalRequests = await this.analytics.getTotalRequests(startTime, endTime);
    const totalErrors = await this.analytics.getTotalErrors(startTime, endTime);

    return {
      totalErrors,
      errorRate: totalRequests > 0 ? (totalErrors / totalRequests) * 100 : 0,
      criticalErrors: await this.analytics.getCriticalErrors(startTime, endTime),
      topErrors: await this.analytics.getTopErrors(startTime, endTime, 5)
    };
  }

  async getCpuUsage() {
    // Simplified CPU usage calculation
    const startUsage = process.cpuUsage();

    return new Promise((resolve) => {
      setTimeout(() => {
        const endUsage = process.cpuUsage(startUsage);
        const userCPU = endUsage.user / 1000; // Convert to milliseconds
        const systemCPU = endUsage.system / 1000;
        const totalCPU = userCPU + systemCPU;
        const usage = Math.round((totalCPU / 100) * 100) / 100; // Percentage
        resolve(usage);
      }, 100);
    });
  }
}
```

```
module.exports = HealthDashboard;
```

**User Feedback System:**

javascript

```
// feedback/feedback.js
class FeedbackService {
  constructor(database) {
    this.db = database;
  }

  async submitFeedback(feedback) {
    const feedbackData = {
      id: generateId(),
      type: feedback.type, // 'bug', 'feature', 'improvement', 'general'
      title: feedback.title,
      description: feedback.description,
      rating: feedback.rating, // 1-5 stars
      email: feedback.email,
      userId: feedback.userId,
      metadata: {
        userAgent: feedback.userAgent,
        url: feedback.url,
        timestamp: new Date().toISOString(),
        version: process.env.npm_package_version
      },
      status: 'open',
      createdAt: new Date().toISOString()
    };

    await this.db.insertFeedback(feedbackData);

    // Send notification to team
    if (feedback.type === 'bug' || feedback.rating <= 2) {
      await this.notifyTeam(feedbackData);
    }

    return feedbackData;
  }

  async getFeedback(filters = {}) {
    return await this.db.getFeedback(filters);
  }

  async updateFeedbackStatus(feedbackId, status, response = null) {
    await this.db.updateFeedback(feedbackId, {
      status,
      response,
      updatedAt: new Date().toISOString()
    });

    // Notify user if response provided
    if (response) {
      await this.notifyUser(feedbackId, response);
    }
  }

  async getFeedbackStats() {
    const stats = await this.db.getFeedbackStats();
    return {
      total: stats.total,
      byType: stats.byType,
      byRating: stats.byRating,
      averageRating: stats.averageRating,
      responseRate: stats.responseRate,
      resolutionTime: stats.averageResolutionTime
    };
  }

  async notifyTeam(feedback) {
    // Send to Slack, email, or other notification system
    if (process.env.SLACK_WEBHOOK_URL) {
      await fetch(process.env.SLACK_WEBHOOK_URL, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          text: `New ${feedback.type} feedback: ${feedback.title}`,
          attachments: [{
            color: feedback.type === 'bug' ? 'danger' : 'warning',
            fields: [{
              title: 'Description',
              value: feedback.description.substring(0, 200) + '...',
              short: false
            }, {
              title: 'Rating',
              value: `${feedback.rating}/5 stars`,
              short: true
            }, {
              title: 'User',
              value: feedback.email || 'Anonymous',
              short: true
            }]
          }]
        })
      });
    }
  }

  async notifyUser(feedbackId, response) {
    const feedback = await this.db.getFeedbackById(feedbackId);

    if (feedback.email) {
      // Send email notification
      // Implementation depends on email service
    }
  }
}

// React component for feedback widget
const FeedbackWidget = () => {
  const [isOpen, setIsOpen] = useState(false);
  const [feedback, setFeedback] = useState({
    type: 'general',
    title: '',
    description: '',
    rating: 5,
    email: ''
  });
  const [submitted, setSubmitted] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      await api.post('/feedback', {
        ...feedback,
        userAgent: navigator.userAgent,
        url: window.location.href
      });

      setSubmitted(true);
      setTimeout(() => {
        setIsOpen(false);
        setSubmitted(false);
        setFeedback({
          type: 'general',
          title: '',
          description: '',
          rating: 5,
          email: ''
        });
      }, 2000);
    } catch (error) {
      console.error('Failed to submit feedback:', error);
    }
  };

  return (
    <>
      <button
        onClick={() => setIsOpen(true)}
        className="fixed bottom-4 right-4 bg-blue-600 text-white px-4 py-2 rounded-full shadow-
      >
        📝 Feedback
      </button>

      {isOpen && (
        <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center
          <div className="bg-white rounded-lg max-w-md w-full p-6">
            {submitted ? (
              <div className="text-center">
                <h3 className="text-lg font-semibold text-green-600 mb-2">
                  Thank you for your feedback!
                </h3>
                <p className="text-gray-600">
                  We appreciate your input and will review it shortly.
                </p>
              </div>
            ) : (
              <form onSubmit={handleSubmit}>
                <h3 className="text-lg font-semibold mb-4">Send Feedback</h3>

                <div className="mb-4">
                  <label className="block text-sm font-medium mb-2">Type</label>
                  <select
                    value={feedback.type}
                    onChange={(e) => setFeedback({ ...feedback, type: e.target.value })}
                    className="w-full border border-gray-300 rounded-md px-3 py-2"
                  >
                    <option value="general">General</option>
                    <option value="bug">Bug Report</option>
                    <option value="feature">Feature Request</option>
                    <option value="improvement">Improvement</option>
                  </select>
                </div>

                <div className="mb-4">
                  <label className="block text-sm font-medium mb-2">Title</label>
                  <input
                    type="text"
                    value={feedback.title}
                    onChange={(e) => setFeedback({ ...feedback, title: e.target.value })}
                    className="w-full border border-gray-300 rounded-md px-3 py-2"
                    required
                  />
                </div>

                <div className="mb-4">
                  <label className="block text-sm font-medium mb-2">Description</label>
                  <textarea
                    value={feedback.description}
                    onChange={(e) => setFeedback({ ...feedback, description: e.target.value })}
                    className="w-full border border-gray-300 rounded-md px-3 py-2 h-24"
                    required
                  />
                </div>

                <div className="mb-4">
                  <label className="block text-sm font-medium mb-2">Rating</label>
                  <div className="flex space-x-1">
                    {[1, 2, 3, 4, 5].map((star) => (
                      <button
                        key={star}
                        type="button"
                        onClick={() => setFeedback({ ...feedback, rating: star })}
                        className={`text-2xl ${
                          star <= feedback.rating ? 'text-yellow-400' : 'text-gray-300'
                        }`}
                      >
                         ⭐
                      </button>
                    ))}
                  </div>
                </div>
```

```
              <div className="mb-6">
                <label className="block text-sm font-medium mb-2">
                  Email (optional)
                </label>
                <input
                  type="email"
                  value={feedback.email}
                  onChange={(e) => setFeedback({ ...feedback, email: e.target.value })}
                  className="w-full border border-gray-300 rounded-md px-3 py-2"
                  placeholder="For follow-up responses"
                />
              </div>

              <div className="flex justify-end space-x-3">
                <button
                  type="button"
                  onClick={() => setIsOpen(false)}
                  className="px-4 py-2 text-gray-600 hover:text-gray-800"
                >
                  Cancel
                </button>
                <button
                  type="submit"
                  className="px-4 py-2 bg-blue-600 text-white rounded-md hover:bg-blue-700"
                >
                  Submit
                </button>
              </div>
            </form>
          )}
        </div>
      </div>
    )}
  </>
  );
};

module.exports = {
  FeedbackService,
  FeedbackWidget
};
```

**Acceptance Criteria:**

- [ ] Prometheus metrics are collected and exposed
- [ ] Grafana dashboards display system health
- [ ] Error tracking captures and reports issues
- [ ] Usage analytics track user behavior
- [ ] Health dashboard shows real-time status
- [ ] User feedback system is functional
- [ ] Monitoring alerts fire correctly
- [ ] Performance metrics meet targets

### REQ-4.5: Community and Open Source Preparation

**Priority:** P1
**Estimated Effort:** 12 hours

**Functional Requirements:**

- Open source license and legal compliance
- Community guidelines and code of conduct
- Issue and pull request templates
- Release process and versioning
- Security policy and vulnerability reporting

**Open Source Configuration:**

```markdown
# LICENSE (MIT License)
MIT License

Copyright (c) 2024 MCP Manager Team

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

~~~markdown
# CONTRIBUTING.md
# Contributing to MCP Manager

Thank you for your interest in contributing to MCP Manager! This document provides guidelines f

## Code of Conduct

This project adheres to the [Contributor Covenant Code of Conduct](CODE_OF_CONDUCT.md). By part

## How to Contribute

### Reporting Bugs

Before creating bug reports, please check the existing issues to avoid duplicates. When creatir

- **Clear description** of the problem
- **Steps to reproduce** the issue
- **Expected behavior** vs actual behavior
- **Environment details** (OS, Docker version, etc.)
- **Screenshots** if applicable

### Suggesting Features

Feature requests are welcome! Please:

- Check existing feature requests first
- Clearly describe the proposed feature
- Explain the use case and benefits
- Consider implementation complexity

### Development Setup

1. **Fork and clone the repository**
   ```bash
   git clone https://github.com/your-username/mcp-manager.git
   cd mcp-manager
~~~

2. **Install dependencies**

```bash
npm install
```

3. **Set up development environment**

```bash
cp .env.example .env.local
# Edit .env.local with your configuration
```

4. **Start development server**

```bash
npm run dev
```

5. **Run tests**

```bash
npm test
```

## Pull Request Process

1. **Create a feature branch**

```bash
git checkout -b feature/your-feature-name
```

2. **Make your changes**
    - Follow the coding standards
    - Add tests for new functionality
    - Update documentation if needed

3. **Test your changes**

```bash
npm run test
npm run lint
npm run type-check
```

4. **Commit your changes**

```bash
git commit -m "feat: add new feature description"
```

Use conventional commits format:

- `feat:` for new features
- `fix:` for bug fixes
- `docs:` for documentation changes
- `test:` for test additions
- `refactor:` for code refactoring

5. **Push and create pull request**

   bash

   ```bash
   git push origin feature/your-feature-name
   ```

## Coding Standards

- **JavaScript/TypeScript**: Follow ESLint configuration
- **React**: Use functional components with hooks
- **CSS**: Use Tailwind CSS utility classes
- **Testing**: Write unit tests for new functionality
- **Documentation**: Update relevant documentation

## Project Structure

```
src/
├── components/          # React components
├── hooks/              # Custom React hooks
├── services/           # Business logic services
├── types/              # TypeScript type definitions
├── utils/              # Utility functions
└── __tests__/          # Test files
```

## Development Guidelines

### Adding New Features

1. **Plan the feature** - Create an issue first
2. **Design the API** - Consider backwards compatibility
3. **Implement with tests** - Maintain test coverage
4. **Update documentation** - Keep docs current

5. **Get feedback** - Submit PR for review

### Testing

- **Unit tests**: Test individual functions/components
- **Integration tests**: Test API endpoints
- **E2E tests**: Test complete user workflows
- **Coverage**: Maintain >90% test coverage

### Performance

- **Frontend**: Optimize bundle size and rendering
- **Backend**: Efficient database queries
- **Memory**: Monitor memory usage and leaks
- **Monitoring**: Add metrics for new features

## Release Process

Releases follow semantic versioning (SemVer):

- **Major** (x.0.0): Breaking changes
- **Minor** (x.y.0): New features (backwards compatible)
- **Patch** (x.y.z): Bug fixes

## Community

- **Discussions**: Use GitHub Discussions for questions
- **Discord**: Join our community Discord server
- **Blog**: Read updates on our development blog

## Recognition

Contributors are recognized in:

- CONTRIBUTORS.md file
- Release notes
- Annual contributor appreciation

Thank you for contributing to MCP Manager!

~~~
```markdown
# CODE_OF_CONDUCT.md
# Contributor Covenant Code of Conduct

## Our Pledge

We as members, contributors, and leaders pledge to make participation in our
community a harassment-free experience for everyone, regardless of age, body
size, visible or invisible disability, ethnicity, sex characteristics, gender
identity and expression, level of experience, education, socio-economic status,
nationality, personal appearance, race, religion, or sexual identity
and orientation.

## Our Standards

Examples of behavior that contributes to a positive environment:

* Using welcoming and inclusive language
* Being respectful of differing viewpoints and experiences
* Gracefully accepting constructive criticism
* Focusing on what is best for the community
* Showing empathy towards other community members

Examples of unacceptable behavior:

* The use of sexualized language or imagery
* Trolling, insulting/derogatory comments, and personal attacks
* Public or private harassment
* Publishing others' private information without explicit permission
* Other conduct which could reasonably be considered inappropriate

## Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards
and will take appropriate and fair corrective action in response to any
behavior that they deem inappropriate, threatening, offensive, or harmful.

## Scope

This Code of Conduct applies within all community spaces, and also applies when
an individual is officially representing the community in public spaces.
~~~

```
## Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be
reported to the community leaders responsible for enforcement at
[conduct@mcpmanager.io](mailto:conduct@mcpmanager.io).

All complaints will be reviewed and investigated promptly and fairly.

## Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://www.contributor-
covenant.org),
version 2.0, available at https://www.contributor-
covenant.org/version/2/0/code_of_conduct.html.
```

**GitHub Templates:**

markdown

```markdown
<!-- .github/ISSUE_TEMPLATE/bug_report.md -->
---
name: Bug report
about: Create a report to help us improve
title: '[BUG] '
labels: 'bug'
assignees: ''
---

**Describe the bug**
A clear and concise description of what the bug is.

**To Reproduce**
Steps to reproduce the behavior:
1. Go to '...'
2. Click on '....'
3. Scroll down to '....'
4. See error

**Expected behavior**
A clear and concise description of what you expected to happen.

**Screenshots**
If applicable, add screenshots to help explain your problem.

**Environment (please complete the following information):**
- OS: [e.g. Ubuntu 20.04]
- Docker Version: [e.g. 20.10.7]
- MCP Manager Version: [e.g. 1.0.0]
- Browser: [e.g. Chrome 91.0]

**Server Information**
- Total servers managed: [e.g. 5]
- Server types: [e.g. 3 containers, 2 Node.js processes]
- Error logs: [Paste relevant logs here]

**Additional context**
Add any other context about the problem here.
```

markdown

```
<!-- .github/ISSUE_TEMPLATE/feature_request.md -->
---
name: Feature request
about: Suggest an idea for this project
title: '[FEATURE] '
labels: 'enhancement'
assignees: ''
---

**Is your feature request related to a problem? Please describe.**
A clear and concise description of what the problem is. Ex. I'm always frustrated when [...]

**Describe the solution you'd like**
A clear and concise description of what you want to happen.

**Describe alternatives you've considered**
A    await page.fill('[data-testid=allowed-paths]', '/tmp');
    await page.click('[data-testid=deploy-server]');

    // Verify deployment success
    await expect(page.locator('[data-testid=toast-success]')).toContainText('deployed successfu

    // Navigate to servers and verify
    await page.goto('http://localhost:3000/servers');
    await expect(page.locator('[data-testid=server-card]')).toContainText('filesystem-from-temp
  });

  test('Real-time updates', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Create server via API (simulating external change)
    await page.evaluate(async () => {
      await fetch('/api/servers', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          name: 'realtime-test-server',
          type: 'container',
          image: 'mcp/test:latest'
        })
      });
    });
    // Verify server appears in real-time without refresh
    await expect(page.locator('[data-testid=server-card]')).toContainText('realtime-test-server
  });

  test('Error handling and recovery', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Test invalid server creation
    await page.click('text=Add Server');
    await page.fill('[data-testid=server-name]', '');
    await page.click('[data-testid=create-server]');

    // Verify error message
    await expect(page.locator('[data-testid=error-message]')).toContainText('Server name is rec

    // Test network error handling
    await page.route('/api/servers', route => route.abort());
    await page.reload();

    // Verify error state
    await expect(page.locator('[data-testid=error-boundary]')).toBeVisible();
  });
});

// Performance Tests
describe('Performance Tests', () => {
  test('Dashboard loads with 100 servers', async ({ page }) => {
    // Seed database with 100 servers
    await seedDatabase(100);

    const startTime = Date.now();
    await page.goto('http://localhost:3000');

    // Wait for content to load
    await page.waitForSelector('[data-testid=server-grid]');
    const loadTime = Date.now() - startTime;

    // Assert load time is acceptable
    expect(loadTime).toBeLessThan(3000); // 3 seconds

    // Check for performance issues
    const performanceMetrics = await page.evaluate(() => {
      return JSON.stringify(performance.getEntriesByType('navigation')[0]);
    });
    const metrics = JSON.parse(performanceMetrics);
    expect(metrics.domContentLoadedEventEnd - metrics.domContentLoadedEventStart).toBeLessThan(
  });

  test('Real-time updates with high message volume', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Simulate high-frequency updates
    await page.evaluate(() => {
      const ws = new WebSocket('ws://localhost:3001/ws');
      ws.onopen = () => {
        // Send 100 updates rapidly
        for (let i = 0; i < 100; i++) {
          setTimeout(() => {
            ws.send(JSON.stringify({
              type: 'server:status',
              data: { serverId: 'test-server', status: i % 2 ? 'running' : 'stopped' }
            }));
          }, i * 10);
        }
      };
    });

    // Verify UI remains responsive
    await page.click('[data-testid=refresh-button]');
    await expect(page.locator('[data-testid=loading-spinner]')).toBeVisible();
  });
});

// Security Tests
describe('Security Tests', () => {
  test('XSS prevention', async ({ page }) => {
    // Test server name XSS
    await page.goto('http://localhost:3000');
    await page.click('text=Add Server');
    await page.fill('[data-testid=server-name]', '<script>alert("xss")</script>');
    await page.fill('[data-testid=docker-image]', 'mcp/test:latest');
    await page.click('[data-testid=create-server]');

    // Verify script is not executed
    const alerts = [];
    page.on('dialog', dialog => {
      alerts.push(dialog.message());
      dialog.dismiss();
    });

    await page.waitForTimeout(1000);
    expect(alerts).toHaveLength(0);

    // Verify content is properly escaped
    await expect(page.locator('[data-testid=server-card]')).toContainText('<script>alert("xss")
  });

  test('CSRF protection', async ({ page }) => {
    // Attempt to make requests without proper headers
    const response = await page.evaluate(async () => {
      return await fetch('/api/servers', {
        method: 'POST',
        body: JSON.stringify({ name: 'csrf-test' }),
        // Missing Content-Type header
      });
    });

    expect(response.status).toBe(400);
  });

  test('Input validation', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Test SQL injection attempts
    await page.click('text=Add Server');
    await page.fill('[data-testid=server-name]', "'; DROP TABLE servers; --");
    await page.fill('[data-testid=docker-image]', 'mcp/test:latest');
    await page.click('[data-testid=create-server]');

    // Verify request is rejected
    await expect(page.locator('[data-testid=error-message]')).toContainText('Invalid character'
  });
});

// Accessibility Tests
describe('Accessibility Tests', () => {
  test('Keyboard navigation', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Test tab navigation
    await page.keyboard.press('Tab');
    await expect(page.locator(':focus')).toHaveAttribute('data-testid', 'add-server-button');

    await page.keyboard.press('Tab');
    await expect(page.locator(':focus')).toHaveAttribute('data-testid', 'refresh-button');

    // Test Enter key activation
    await page.keyboard.press('Enter');
    // Should trigger refresh action
  });

  test('Screen reader compatibility', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Check ARIA labels
    await expect(page.locator('[data-testid=add-server-button]')).toHaveAttribute('aria-label')
    await expect(page.locator('[data-testid=server-status]')).toHaveAttribute('aria-live');

    // Check heading structure
    const headings = await page.locator('h1, h2, h3, h4, h5, h6').allTextContents();
    expect(headings[0]).toContain('MCP Manager'); // Main heading
  });

  test('Color contrast', async ({ page }) => {
    await page.goto('http://localhost:3000');

    // Check contrast ratios for key elements
    const contrastRatios = await page.evaluate(() => {
      const elements = document.querySelectorAll('[data-testid*="button"], [data-testid*="statu
      const ratios = [];

      elements.forEach(el => {
        const styles = getComputedStyle(el);
        const bgColor = styles.backgroundColor;
        const textColor = styles.color;

        // Calculate contrast ratio (simplified)
        // In real implementation, use proper contrast calculation
        ratios.push({
          element: el.getAttribute('data-testid'),
          bgColor,
          textColor,
          // ratio: calculateContrast(bgColor, textColor)
        });
      });
      return ratios;
    });

    // Verify all elements meet WCAG AA standards (4.5:1)
    // This would need proper contrast calculation implementation
  });
});

// Load Testing
describe('Load Tests', () => {
  test('Concurrent user simulation', async () => {
    const users = 50;
    const promises = [];

    for (let i = 0; i < users; i++) {
      promises.push(simulateUser(i));
    }

    const results = await Promise.allSettled(promises);
    const failures = results.filter(r => r.status === 'rejected');

    // Allow up to 5% failure rate
    expect(failures.length / users).toBeLessThan(0.05);
  });

  async function simulateUser(userId) {
    const browser = await playwright.chromium.launch();
    const page = await browser.newPage();

    try {
      await page.goto('http://localhost:3000');
      await page.click('text=Add Server');
      await page.fill('[data-testid=server-name]', `load-test-${userId}`);
      await page.fill('[data-testid=docker-image]', 'mcp/test:latest');
      await page.click('[data-testid=create-server]');

      await page.waitForSelector(`[data-testid="server-card"]:has-text("load-test-${userId}")`)

      await page.click('[data-testid=start-server]');
      await page.waitForSelector('[data-testid="server-status"]:has-text("running")');

    } finally {
      await browser.close();
      }
    }
  });
```

**Test Data and Utilities:**

javascript

```
// Test Utilities
export class TestUtils {
  static async seedDatabase(serverCount = 10) {
    const servers = [];

    for (let i = 0; i < serverCount; i++) {
      servers.push({
        name: `test-server-${i}`,
        type: i % 3 === 0 ? 'container' : 'nodejs',
        image: i % 3 === 0 ? 'mcp/test:latest' : undefined,
        command: i % 3 !== 0 ? 'node' : undefined,
        args: i % 3 !== 0 ? ['server.js'] : undefined,
        status: ['running', 'stopped', 'error'][i % 3],
        healthStatus: ['healthy', 'unhealthy', 'unknown'][i % 3]
      });
    }

    // Insert into test database
    await Promise.all(servers.map(server =>
      fetch('/api/servers', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(server)
      })
    ));
  }

  static async cleanupDatabase() {
    const response = await fetch('/api/servers');
    const { data: servers } = await response.json();

    await Promise.all(servers.map(server =>
      fetch(`/api/servers/${server.id}`, { method: 'DELETE' })
    ));
  }

  static generateMockServer(overrides = {}) {
    return {
      id: `srv_${Math.random().toString(36).substr(2, 9)}`,
      name: `mock-server-${Date.now()}`,
      type: 'container',
      image: 'mcp/test:latest',
      status: 'stopped',
```

```javascript
        healthStatus: 'unknown',
        createdAt: new Date().toISOString(),
        updatedAt: new Date().toISOString(),
        ...overrides
      };
    }

    static async waitForCondition(condition, timeout = 5000) {
      const start = Date.now();

      while (Date.now() - start < timeout) {
        if (await condition()) {
          return true;
        }
        await new Promise(resolve => setTimeout(resolve, 100));
      }

      throw new Error('Condition not met within timeout');
    }
  }

  // Mock Services for Testing
  export class MockWebSocketService {
    constructor() {
      this.clients = new Set();
      this.messages = [];
    }

    addClient(client) {
      this.clients.add(client);
    }

    removeClient(client) {
      this.clients.delete(client);
    }

    broadcast(type, data) {
      const message = { type, data, timestamp: new Date().toISOString() };
      this.messages.push(message);

      this.clients.forEach(client => {
        if (client.readyState === WebSocket.OPEN) {
          client.send(JSON.stringify(message));
        }
      });
    }

    getMessages() {
      return this.messages;
    }

    clearMessages() {
      this.messages = [];
    }
  }
```

**Acceptance Criteria:**

- [ ] >90% test coverage across all modules
- [ ] All E2E scenarios pass consistently
- [ ] Performance tests meet defined thresholds
- [ ] Security tests pass without vulnerabilities
- [ ] Accessibility tests achieve WCAG AA compliance
- [ ] Load tests handle expected concurrent users
- [ ] CI/CD pipeline runs all tests automatically

### REQ-4.3: Production Deployment Pipeline

**Priority:** P0
**Estimated Effort:** 16 hours

**Functional Requirements:**

- Automated CI/CD pipeline
- Multi-environment deployments
- Docker image building and publishing
- Kubernetes deployment manifests
- Monitoring and alerting setup

**CI/CD Pipeline Configuration:**

yaml

```yaml
# .github/workflows/ci-cd.yml
name: CI/CD Pipeline

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

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: mcpmanager/mcp-manager

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      docker:
        image: docker:dind
        options: --privileged

    steps:
    - uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '18'
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run linting
      run: npm run lint

    - name: Run type checking
      run: npm run type-check

    - name: Run unit tests
      run: npm run test:unit -- --coverage
    - name: Upload coverage reports
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage/lcov.info

    - name: Build application
      run: npm run build

    - name: Run E2E tests
      run: |
        docker-compose -f docker-compose.test.yml up -d
        npm run test:e2e
        docker-compose -f docker-compose.test.yml down

    - name: Run security scan
      uses: securecodewarrior/github-action-add-sarif@v1
      with:
        sarif-file: 'security-scan-results.sarif'

  build:
    needs: test
    runs-on: ubuntu-latest
    if: github.event_name == 'push'

    outputs:
      image-digest: ${{ steps.build.outputs.digest }}
      image-tag: ${{ steps.meta.outputs.tags }}

    steps:
    - uses: actions/checkout@v4

    - name: Setup Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Extract metadata
      id: meta
      uses: docker/metadata-action@v5
      with:
        images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
        tags: |
          type=ref,event=branch
          type=ref,event=pr
          type=sha,prefix={{branch}}-
          type=raw,value=latest,enable={{is_default_branch}}

    - name: Build and push Docker image
      id: build
      uses: docker/build-push-action@v5
      with:
        context: .
        push: true
        tags: ${{ steps.meta.outputs.tags }}
        labels: ${{ steps.meta.outputs.labels }}
        cache-from: type=gha
        cache-to: type=gha,mode=max
        platforms: linux/amd64,linux/arm64

  deploy-staging:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/develop'
    environment: staging

    steps:
    - uses: actions/checkout@v4

    - name: Deploy to staging
      run: |
        echo "Deploying to staging environment"
        # Add staging deployment logic

    - name: Run smoke tests
      run: |
        echo "Running smoke tests against staging"
        # Add smoke test logic

  deploy-production:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    environment: production
```

```yaml
      steps:
      - uses: actions/checkout@v4

      - name: Deploy to production
        run: |
          echo "Deploying to production environment"
          # Add production deployment logic

      - name: Update release notes
        uses: release-drafter/release-drafter@v5
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

    security-scan:
      runs-on: ubuntu-latest
      steps:
      - uses: actions/checkout@v4

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:latest
          format: 'sarif'
          output: 'trivy-results.sarif'

      - name: Upload Trivy scan results
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: 'trivy-results.sarif'
```

**Kubernetes Deployment Manifests:**

yaml

```yaml
# k8s/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: mcp-manager
  labels:
    app.kubernetes.io/name: mcp-manager

---
# k8s/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mcp-manager-config
  namespace: mcp-manager
data:
  NODE_ENV: "production"
  LOG_LEVEL: "info"
  DATABASE_URL: "sqlite:///app/data/mcp-manager.db"
  WS_HEARTBEAT_INTERVAL: "30000"

---
# k8s/secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mcp-manager-secrets
  namespace: mcp-manager
type: Opaque
data:
  JWT_SECRET: <base64-encoded-secret>
  SESSION_SECRET: <base64-encoded-secret>

---
# k8s/pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mcp-manager-data
  namespace: mcp-manager
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: standard

---
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mcp-manager
  namespace: mcp-manager
  labels:
    app.kubernetes.io/name: mcp-manager
    app.kubernetes.io/version: "1.0.0"
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: mcp-manager
  template:
    metadata:
      labels:
        app.kubernetes.io/name: mcp-manager
    spec:
      containers:
      - name: mcp-manager
        image: ghcr.io/mcpmanager/mcp-manager:latest
        ports:
        - containerPort: 3000
          name: http
        - containerPort: 3001
          name: api
        envFrom:
        - configMapRef:
            name: mcp-manager-config
        - secretRef:
            name: mcp-manager-secrets
        volumeMounts:
        - name: data
          mountPath: /app/data
        - name: docker-sock
          mountPath: /var/run/docker.sock
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 5
        securityContext:
          runAsNonRoot: true
          runAsUser: 1001
          allowPrivilegeEscalation: false
          readOnlyRootFilesystem: true
          capabilities:
            drop:
            - ALL
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: mcp-manager-data
      - name: docker-sock
        hostPath:
          path: /var/run/docker.sock
          type: Socket
      securityContext:
        fsGroup: 1001

---
# k8s/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: mcp-manager-service
  namespace: mcp-manager
  labels:
    app.kubernetes.io/name: mcp-manager
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 3000
    protocol: TCP
    name: http
  - port: 3001
    targetPort: 3001
    protocol: TCP
    name: api
  selector:
    app.kubernetes.io/name: mcp-manager

---
# k8s/ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mcp-manager-ingress
  namespace: mcp-manager
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/proxy-read-timeout: "86400"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "86400"
    nginx.ingress.kubernetes.io/server-snippets: |
      location /ws {
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
      }
spec:
  tls:
  - hosts:
    - mcp-manager.example.com
    secretName: mcp-manager-tls
  rules:
```

```yaml
  - host: mcp-manager.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: mcp-manager-service
            port:
              number: 80

---
# k8s/hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: mcp-manager-hpa
  namespace: mcp-manager
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mcp-manager
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

**Helm Chart Structure:**

```
yaml
```

```yaml
# helm/Chart.yaml
apiVersion: v2
name: mcp-manager
description: A Helm chart for MCP Manager
version: 1.0.0
appVersion: "1.0.0"
home: https://github.com/mcpmanager/mcp-manager
sources:
  - https://github.com/mcpmanager/mcp-manager
maintainers:
  - name: MCP Manager Team
    email: team@mcpmanager.io

# helm/values.yaml
replicaCount: 1

image:
  repository: ghcr.io/mcpmanager/mcp-manager
  tag: latest
  pullPolicy: IfNotPresent

service:
  type: ClusterIP
  port: 80
  targetPort: 3000

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
  hosts:
    - host: mcp-manager.local
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: mcp-manager-tls
      hosts:
        - mcp-manager.local

persistence:
  enabled: true
```

```yaml
    storageClass: standard
    size: 10Gi

  resources:
    requests:
      memory: "512Mi"
      cpu: "250m"
    limits:
      memory: "1Gi"
      cpu: "500m"

  autoscaling:
    enabled: true
    minReplicas: 1
    maxReplicas: 5
    targetCPUUtilizationPercentage: 70
    targetMemoryUtilizationPercentage: 80

  config:
    nodeEnv: production
    logLevel: info
    wsHeartbeatInterval: 30000

  secrets:
    jwtSecret: ""
    sessionSecret: ""
```

**Acceptance Criteria:**

- [ ] CI/CD pipeline runs automatically on code changes
- [ ] Docker images are built and published successfully
- [ ] Kubernetes manifests deploy without errors
- [ ] Helm chart installs and upgrades correctly
- [ ] Security scans pass with no critical vulnerabilities
- [ ] Deployment rollbacks work correctly
- [ ] Environment promotion process is automated

### REQ-4.4: Monitoring and Analytics

**Priority:** P1
**Estimated Effort:** 14 hours

**Functional Requirements:**

- Application performance monitoring
- Error tracking and alerting
- Usage analytics
- Health dashboard
- User feedback collection

**Monitoring Stack:**

```
yaml
```

```yaml
# monitoring/prometheus.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s

    scrape_configs:
    - job_name: 'mcp-manager'
      static_configs:
      - targets: ['mcp-manager-service:3001']
      metrics_path: /metrics
      scrape_interval: 5s

    - job_name: 'node-exporter'
      static_configs:
      - targets: ['node-exporter:9100']

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
      - name: prometheus
        image: prom/prometheus:latest
        ports:
        - containerPort: 9090
        volumeMounts:
        - name: config
```

```yaml
          mountPath: /etc/prometheus
        - name: storage
          mountPath: /prometheus
        args:
          - '--config.file=/etc/prometheus/prometheus.yml'
          - '--storage.tsdb.path=/prometheus'
          - '--web.console.libraries=/etc/prometheus/console_libraries'
          - '--web.console.templates=/etc/prometheus/consoles'
          - '--storage.tsdb.retention.time=168h'
      volumes:
      - name: config
        configMap:
          name: prometheus-config
      - name: storage
        emptyDir: {}

---
# monitoring/grafana.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
      - name: grafana
        image: grafana/grafana:latest
        ports:
        - containerPort: 3000
        env:
        - name: GF_SECURITY_ADMIN_PASSWORD
          value: admin
        volumeMounts:
        - name: grafana-storage
          mountPath: /var/lib/grafana
        - name: grafana-config
          mountPath: /etc/grafana/provisioning
      volumes:
      - name: grafana-storage
        emptyDir: {}
      - name: grafana-config
        configMap:
          name: grafana-config
```

### Application Metrics:

```
javascript
```

```javascript
// metrics/metrics.js
const prometheus = require('prom-client');

// Create a Registry to register the metrics
const register = new prometheus.Registry();

// Add default Node.js metrics
prometheus.collectDefaultMetrics({ register });

// Custom metrics
const httpRequestDuration = new prometheus.Histogram({
  name: 'http_request_duration_seconds',
  help: 'Duration of HTTP requests in seconds',
  labelNames: ['method', 'route', 'status_code'],
  buckets: [0.1, 0.5, 1, 2, 5]
});

const activeConnections = new prometheus.Gauge({
  name: 'websocket_connections_active',
  help: 'Number of active WebSocket connections'
});

const serverOperations = new prometheus.Counter({
  name: 'server_operations_total',
  help: 'Total number of server operations',
  labelNames: ['operation', 'status']
});

const healthCheckDuration = new prometheus.Histogram({
  name: 'health_check_duration_seconds',
  help: 'Duration of health checks in seconds',
  labelNames: ['server_id', 'status']
});

const errorCount = new prometheus.Counter({
  name: 'application_errors_total',
  help: 'Total number of application errors',
  labelNames: ['type', 'severity']
});

// Register metrics
register.registerMetric(httpRequestDuration);
register.registerMetric(activeConnections);
register.registerMetric(serverOperations);
register.registerMetric(healthCheckDuration);
register.registerMetric(errorCount);

// Middleware to track HTTP requests
const metricsMiddleware = (req, res, next) => {
  const start = Date.now();

  res.on('finish', () => {
    const duration = (Date.now() - start) / 1000;
    httpRequestDuration
      .labels(req.method, req.route?.path || req.path, res.statusCode)
      .observe(duration);
  });

  next();
};

// Function to track# MCP Manager - Phase 4 PRD: Polish and Release

**Timeline:** Weeks 13-16
**Goal:** Final polish, documentation, testing, and preparation for open source release
**Success Criteria:** Production-ready application with comprehensive documentation and communi

## Phase 4 Architecture Overview
```

```
┌─────────────────────────────────────────────┐
│      Production Release        │
├─────────────────────────────────────────────┤
│ Documentation │ Testing │ Deploy │
│ - User Guide  │ - E2E   │ - CI/CD │
│ - API Docs    │ - Load  │ - Docker │
│ - Dev Setup   │ - Sec   │ - Helm  │
│ - Contributing │ - A11y  │ - GitHub │
└─────────────────────────────────────────────┘
│          │          │
│      Community      Release
│      Features      Infrastructure
│          │          │
└─────────────────────────────────────────────┘
│       Monitoring & Analytics       │
│   - Error Tracking                    │
│   - Usage Analytics                   │
│   - Performance Monitoring            │
│   - User Feedback System              │
└───────────────────────────────────────┘
```

~~~
## Core Requirements

### REQ-4.1: Documentation System
**Priority:** P0
**Estimated Effort:** 20 hours

**Functional Requirements:**
- Comprehensive user documentation
- API documentation with examples
- Developer setup guide
- Contributing guidelines
- Architecture documentation

**Technical Implementation:**
```markdown
# Documentation Structure
docs/
├── README.md                    # Main project overview
├── user-guide/
│   ├── getting-started.md
│   ├── server-management.md
│   ├── marketplace.md
│   ├── monitoring.md
│   ├── troubleshooting.md
│   └── faq.md
├── api/
│   ├── README.md
│   ├── servers.md
│   ├── templates.md
│   ├── health.md
│   ├── logs.md
│   └── websockets.md
├── development/
│   ├── setup.md
│   ├── architecture.md
│   ├── contributing.md
│   ├── testing.md
│   └── deployment.md
├── examples/
│   ├── docker-compose.yml
│   ├── kubernetes.yaml
│   ├── custom-templates/
~~~

```
│   └── api-examples/
└── assets/
    ├── screenshots/
    ├── diagrams/
    └── videos/
```

**User Guide Content:**

~~~markdown
# Getting Started with MCP Manager

## Quick Start

MCP Manager provides a web-based interface for managing Model Context Protocol servers. Get up

### Prerequisites
- Docker Engine 20.0+
- 4GB RAM minimum
- 10GB disk space

### Installation

1. **Pull the Docker image:**
   ```bash
   docker pull mcpmanager/mcp-manager:latest
~~~

2. **Run the container:**

   ```bash
   docker run -d \
     --name mcp-manager \
     -p 3000:3000 \
     -v /var/run/docker.sock:/var/run/docker.sock \
     -v mcp-data:/app/data \
     mcpmanager/mcp-manager:latest
   ```

3. **Access the web interface:** Open http://localhost:3000 in your browser

## First Steps

1. **Add your first server** from the marketplace
2. **Configure** any required settings

3. **Deploy** and start monitoring
4. **View logs** and health metrics

## Server Management

### Creating Servers

MCP Manager supports two types of servers:

**Container-based servers:**

- Pre-built Docker images
- Isolated and secure
- Easy deployment and updates
- Resource management

**Process-based servers:**

- Node.js and Python servers
- Custom implementations
- Local development
- Direct system access

### Configuration

Each server type has specific configuration options:

**Environment Variables:** Set environment variables for API keys, database connections, and other configuration.

**Volume Mounts:** For filesystem servers, configure which directories are accessible.

**Port Mapping:** Expose server ports for HTTP health checks and client connections.

**Health Checks:** Configure HTTP endpoints for health monitoring.

## Marketplace

### Using Templates

The marketplace provides pre-configured templates for popular MCP servers:

1. **Browse** available templates by category
2. **Search** for specific functionality

3. **Review** configuration requirements
4. **Deploy** with custom settings

### Popular Templates

**Filesystem Access:**

- Secure file operations
- Configurable permissions
- Path restrictions

**GitHub Integration:**

- Repository management
- Issue tracking
- Pull request automation

**Database Connectors:**

- PostgreSQL, MySQL support
- Query execution
- Schema introspection

## Monitoring and Troubleshooting

### Health Monitoring

MCP Manager automatically monitors server health:

- **HTTP checks** for web-based servers
- **Process monitoring** for local servers
- **Response time** tracking
- **Uptime** statistics

### Log Management

- **Real-time streaming** of server output
- **Log level** filtering and search
- **Export** capabilities
- **Historical** log retention

### Common Issues

**Server won't start:**

- Check configuration syntax
- Verify required environment variables
- Review server logs for errors

**Health checks failing:**

- Confirm health endpoint is correct
- Check server is listening on expected port
- Verify network connectivity

**Performance issues:**

- Monitor resource usage
- Check for memory leaks
- Review log volume

~~~
**API Documentation:**
```markdown
# MCP Manager API Reference

## Authentication

Currently, the MCP Manager API does not require authentication. This will be added in future
versions for production deployments.

## Base URL
~~~

http://localhost:3001/api

~~~
## Server Management

### List Servers

Get all registered MCP servers.

**Endpoint:** `GET /servers`

**Response:**
```json
{
  "success": true,
  "data": [
    {
      "id": "srv_123",
      "name": "filesystem-server",
      "type": "container",
      "status": "running",
      "healthStatus": "healthy",
      "createdAt": "2024-01-01T00:00:00Z",
      "updatedAt": "2024-01-01T12:00:00Z"
    }
  ]
}
~~~

### Create Server

Create a new MCP server.

**Endpoint:** `POST /servers`

**Request Body:**

json

```json
{
  "name": "my-filesystem-server",
  "type": "container",
  "image": "mcp/filesystem:latest",
  "config": {
    "environment": {
      "ALLOWED_PATHS": "/tmp,/home/user/documents"
    },
    "port": 3000
  }
}
```

**Response:**

json

```json
{
  "success": true,
  "data": {
    "id": "srv_124",
    "name": "my-filesystem-server",
    "status": "stopped"
  }
}
```

## Server Actions

Control server lifecycle.

**Start Server:** `POST /servers/{id}/start` **Stop Server:** `POST /servers/{id}/stop` **Restart Server:** `POST /servers/{id}/restart`

**Response:**

json

```json
{
  "success": true,
  "message": "Server started successfully"
}
```

## Health Monitoring

### Get Health Status

**Endpoint:** `GET /servers/{id}/health`

**Response:**

```json
{
  "success": true,
  "data": {
    "status": "healthy",
    "lastCheck": "2024-01-01T12:00:00Z",
    "responseTime": 150,
    "uptime": 99.5
  }
}
```

### Health History

**Endpoint:** `GET /servers/{id}/health/history?range=24h`

**Query Parameters:**

- `range`: Time range (1h, 24h, 7d, 30d)

## Error Handling

All endpoints return errors in a consistent format:

```json
{
  "success": false,
  "error": "Server not found",
  "code": "SERVER_NOT_FOUND"
}
```

**Common Error Codes:**

- `SERVER_NOT_FOUND` (404)
- `VALIDATION_ERROR` (400)
- `INTERNAL_ERROR` (500)

- `RATE_LIMITED` (429)

~~~~
**Acceptance Criteria:**
- [ ] Complete user guide with screenshots
- [ ] API documentation with examples
- [ ] Developer setup guide tested by external contributor
- [ ] Contributing guidelines published
- [ ] All documentation reviewed and edited
- [ ] Documentation website deployed

### REQ-4.2: Testing and Quality Assurance
**Priority:** P0
**Estimated Effort:** 24 hours

**Functional Requirements:**
- Comprehensive test coverage (>90%)
- End-to-end testing scenarios
- Performance and load testing
- Security testing
- Accessibility testing

**Testing Strategy:**
```javascript
// Test Configuration
module.exports = {
  // Unit Tests - Jest
  testMatch: [
    '<rootDir>/src/**/__tests__/**/*.{js,jsx,ts,tsx}',
    '<rootDir>/src/**/*.{test,spec}.{js,jsx,ts,tsx}'
  ],
  collectCoverageFrom: [
    'src/**/*.{js,jsx,ts,tsx}',
    '!src/**/*.d.ts',
    '!src/index.tsx',
    '!src/serviceWorker.ts'
  ],
  coverageThreshold: {
    global: {
      branches: 90,
      functions: 90,
      lines: 90,
      statements: 90
    }
  },
~~~~

```
    // E2E Tests - Playwright
    e2eTestDir: './e2e',
    webServer: {
      command: 'npm run start',
      port: 3000,
      reuseExistingServer: !process.env.CI
    }
  };
```

**Comprehensive Test Suite:**

javascript

```
// E2E Test Scenarios
describe('MCP Manager E2E Tests', () => {

  test('Complete user workflow', async ({ page }) => {
    // Navigate to application
    await page.goto('http://localhost:3000');

    // Verify dashboard loads
    await expect(page.locator('h1')).toContainText('MCP Manager');

    // Create new server
    await page.click('text=Add Server');
    await page.fill('[data-testid=server-name]', 'test-server');
    await page.fill('[data-testid=docker-image]', 'mcp/filesystem:latest');
    await page.click('[data-testid=create-server]');

    // Verify server appears in list
    await expect(page.locator('[data-testid=server-card]')).toContainText('test-server');

    // Start server
    await page.click('[data-testid=start-server]');
    await expect(page.locator('[data-testid=server-status]')).toContainText('running');

    // View server details
    await page.click('[data-testid=view-server]');
    await expect(page.locator('h1')).toContainText('test-server');

    // Check logs tab
    await page.click('text=Logs');
    await expect(page.locator('[data-testid=log-viewer]')).toBeVisible();

    // Check health tab
    await page.click('text=Health');
    await expect(page.locator('[data-testid=health-chart]')).toBeVisible();

    // Stop and delete server
    await page.click('text=Overview');
    await page.click('[data-testid=stop-server]');
    await page.click('[data-testid=delete-server]');
    await page.click('[data-testid=confirm-delete]');
  });

  test('Marketplace template deployment', async ({ page }) => {
```

```
await page.goto('http://localhost:3000/marketplace');

// Search for filesystem template
await page.fill('[data-testid=search-input]', 'filesystem');
await expect(page.locator('[data-testid=template-card]')).toContainText('Filesystem Access'

// Deploy template
await page.click('[data-testid=deploy-template]');
await page.fill('[data-testid=server-name]', 'filesystem-from-template');
await page.fill('[data-testid=allowed-paths
```