Issue #7: Add health check endpoint

Labels: `infrastructure`, `monitoring`, `production`
Milestone: Production Ready
Estimated Effort: 2-4 hours

Description

Implement health check and readiness probe endpoints to enable proper monitoring, load balancing, and deployment management on Railway.com and other cloud platforms.

Why Health Checks?

Problems Without Health Checks

- ❌ No way to verify if application is running
- ❌ Load balancers can't detect unhealthy instances
- ❌ Deployment platforms can't verify successful startup
- ❌ No automated restart on failure
- ❌ Difficult to diagnose production issues

Benefits of Health Checks

- ✅ Automated monitoring and alerting
- ✅ Graceful handling of unhealthy instances
- ✅ Zero-downtime deployments
- ✅ Better incident response
- ✅ Compliance with cloud platform best practices

Types of Health Checks

1. Liveness Probe

Purpose: Is the application running?
Endpoint: `GET /health`
Use Case: Detect if application needs to be restarted

2. Readiness Probe

Purpose: Is the application ready to serve traffic?
Endpoint: `GET /ready`
Use Case: Detect if application can handle requests

3. Startup Probe

Purpose: Has the application finished starting up?
Endpoint: `GET /startup`
Use Case: Give slow-starting apps time to initialize

Tasks

Phase 1: Basic Health Check (1 hour)

- [ ] Create Express server for health endpoints
- [ ] Implement `/health` endpoint (liveness probe)
- [ ] Return 200 OK with basic system info
- [ ] Add uptime and timestamp information

Phase 2: Readiness Check (1-2 hours)

- [ ] Implement `/ready` endpoint
- [ ] Check database connectivity
- [ ] Check Slack API connectivity
- [ ] Return 503 if not ready to serve
- [ ] Add detailed status information

Phase 3: Integration (1 hour)

- [ ] Integrate health server with main application
- [ ] Add graceful shutdown support
- [ ] Update state on connection changes
- [ ] Test with Railway.com

Phase 4: Monitoring (1 hour)

- [ ] Add metrics endpoint `/metrics` (optional)
- [ ] Add debug endpoint `/debug/status` (development only)
- [ ] Document monitoring setup
- [ ] Add example monitoring queries

Implementation

Health Check Server

```
// health.js
const express = require('express');
const pool = require('./db');

class HealthServer {
  constructor(port = 3001) {
    this.app = express();
    this.port = port;
    this.server = null;
    this.startTime = Date.now();

    // State tracking
    this.state = {
      isReady: false,
      database: { connected: false, lastCheck: null, error: null },
      slack: { connected: false, lastCheck: null, error: null },
    };

    this.setupRoutes();
  }

  setupRoutes() {
    // Liveness probe - Always returns 200 if process is running
    this.app.get('/health', (req, res) => {
      const uptime = Math.floor((Date.now() - this.startTime) / 1000);
      res.status(200).json({
        status: 'ok',
        service: 'pp-bot',
        timestamp: new Date().toISOString(),
        uptime: `${uptime}s`,
        version: process.env.npm_package_version || '1.0.0',
      });
    });

    // Readiness probe - Returns 200 only if ready to serve
    this.app.get('/ready', (req, res) => {
      const { database, slack, isReady } = this.state;

      const ready = isReady && database.connected && slack.connected;
      const status = ready ? 200 : 503;

      res.status(status).json({
        status: ready ? 'ready' : 'not ready',
        timestamp: new Date().toISOString(),
        checks: {
          database: {
            status: database.connected ? 'connected' : 'disconnected',
            lastCheck: database.lastCheck,
            error: database.error,
          },
          slack: {
            status: slack.connected ? 'connected' : 'disconnected',
            lastCheck: slack.lastCheck,
            error: slack.error,
          },
        },
      });
    });

    // Startup probe - Returns 200 when startup is complete
    this.app.get('/startup', (req, res) => {
      const startupComplete = this.state.isReady;
```

```
      const status = startupComplete ? 200 : 503;

      res.status(status).json({
        status: startupComplete ? 'started' : 'starting',
        timestamp: new Date().toISOString(),
        uptimeMs: Date.now() - this.startTime,
      });
    });

    // Metrics endpoint (optional, for monitoring)
    this.app.get('/metrics', (req, res) => {
      const metrics = {
        uptime_seconds: Math.floor((Date.now() - this.startTime) / 1000),
        memory_usage: process.memoryUsage(),
        cpu_usage: process.cpuUsage(),
        nodejs_version: process.version,
        platform: process.platform,
        arch: process.arch,
      };

      res.status(200).json(metrics);
    });

    // Debug endpoint (development only)
    if (process.env.NODE_ENV !== 'production') {
      this.app.get('/debug/status', (req, res) => {
        res.status(200).json({
          state: this.state,
          env: {
            NODE_ENV: process.env.NODE_ENV,
            DATABASE_URL: process.env.DATABASE_URL ? '[REDACTED]' : 'not set',
            SLACK_BOT_TOKEN: process.env.SLACK_BOT_TOKEN ? '[REDACTED]' : 'not set',
          },
          startTime: new Date(this.startTime).toISOString(),
        });
      });
    }
  }

  // Start health check server
  start() {
    return new Promise((resolve, reject) => {
      this.server = this.app.listen(this.port, (err) => {
        if (err) {
          reject(err);
        } else {
          console.log(`✓ Health check server running on port ${this.port}`);
          resolve();
        }
      });
    });
  }

  // Stop health check server
  stop() {
    return new Promise((resolve) => {
      if (this.server) {
        this.server.close(() => {
          console.log('✓ Health check server stopped');
          resolve();
        });
      } else {
        resolve();
```

```
      }
    });
  }

  // Update readiness state
  setReady(ready) {
    this.state.isReady = ready;
  }

  // Update database connection state
  setDatabaseConnected(connected, error = null) {
    this.state.database.connected = connected;
    this.state.database.lastCheck = new Date().toISOString();
    this.state.database.error = error ? error.message : null;
  }

  // Update Slack connection state
  setSlackConnected(connected, error = null) {
    this.state.slack.connected = connected;
    this.state.slack.lastCheck = new Date().toISOString();
    this.state.slack.error = error ? error.message : null;
  }

  // Periodic health checks
  startPeriodicChecks(interval = 30000) { // 30 seconds
    this.checkInterval = setInterval(async () => {
      await this.checkDatabaseHealth();
      // Slack health is checked via connection events
    }, interval);
  }

  stopPeriodicChecks() {
    if (this.checkInterval) {
      clearInterval(this.checkInterval);
    }
  }

  // Check database health
  async checkDatabaseHealth() {
    try {
      await pool.query('SELECT 1');
      this.setDatabaseConnected(true);
    } catch (error) {
      console.error('Database health check failed:', error);
      this.setDatabaseConnected(false, error);
    }
  }
}

module.exports = HealthServer;
```

Integration with Main Application

```
// index.js
require('dotenv').config();
const { App } = require('@slack/bolt');
const pool = require('./db');
const HealthServer = require('./health');

// Initialize health check server
const healthServer = new HealthServer(process.env.HEALTH_PORT || 3001);

// Initialize Slack app
const app = new App({
  token: process.env.SLACK_BOT_TOKEN,
  appToken: process.env.SLACK_APP_TOKEN,
  socketMode: true,
});

// ... (existing bot logic) ...

// Startup sequence
async function startBot() {
  try {
    // 1. Start health check server
    await healthServer.start();
    console.log('✓ Health check server started');

    // 2. Check database connection
    try {
      await pool.query('SELECT 1');
      console.log('✓ Database connected');
      healthServer.setDatabaseConnected(true);
    } catch (error) {
      console.error('✗ Database connection failed:', error);
      healthServer.setDatabaseConnected(false, error);
      throw error;
    }

    // 3. Start Slack bot
    await app.start(process.env.PORT || 3000);
    console.log('⚡ Bolt app is running!');
    healthServer.setSlackConnected(true);

    // 4. Mark as ready
    healthServer.setReady(true);
    console.log('✓ Application ready to serve requests');

    // 5. Start periodic health checks
    healthServer.startPeriodicChecks(30000); // Every 30 seconds

  } catch (error) {
    console.error('✗ Failed to start bot:', error);
    healthServer.setReady(false);
    process.exit(1);
  }
}

// Graceful shutdown
async function shutdown() {
  console.log('Shutting down gracefully...');

  // Mark as not ready (stop accepting new requests)
  healthServer.setReady(false);
```

```
  // Stop periodic checks
  healthServer.stopPeriodicChecks();

  // Give time for in-flight requests to complete
  await new Promise(resolve => setTimeout(resolve, 5000));

  // Stop Slack bot
  try {
    await app.stop();
    console.log('✓ Slack bot stopped');
  } catch (error) {
    console.error('✗ Error stopping Slack bot:', error);
  }

  // Close database connections
  try {
    await pool.end();
    console.log('✓ Database connections closed');
  } catch (error) {
    console.error('✗ Error closing database:', error);
  }

  // Stop health server
  await healthServer.stop();

  console.log('✓ Shutdown complete');
  process.exit(0);
}

// Handle shutdown signals
process.on('SIGTERM', shutdown);
process.on('SIGINT', shutdown);

// Handle uncaught errors
process.on('uncaughtException', (error) => {
  console.error('Uncaught exception:', error);
  healthServer.setReady(false);
  shutdown();
});

process.on('unhandledRejection', (reason, promise) => {
  console.error('Unhandled rejection at:', promise, 'reason:', reason);
  healthServer.setReady(false);
  shutdown();
});

// Start the bot
if (require.main === module) {
  startBot();
}

module.exports = { app, healthServer };
```

Railway.com Configuration

Update `railway.toml` or `railway.json`:

```
[deploy]
startCommand = "npm start"
healthcheckPath = "/health"
healthcheckTimeout = 100
healthcheckInterval = 30
restartPolicyType = "ON_FAILURE"
restartPolicyMaxRetries = 10
```

Testing

Manual Testing

```
# Test liveness probe
curl http://localhost:3001/health

# Test readiness probe
curl http://localhost:3001/ready

# Test startup probe
curl http://localhost:3001/startup

# Test metrics endpoint
curl http://localhost:3001/metrics
```

Expected Responses

Healthy Response (`/health`):

```
{
  "status": "ok",
  "service": "pp-bot",
  "timestamp": "2025-10-23T12:00:00.000Z",
  "uptime": "3600s",
  "version": "1.0.0"
}
```

Ready Response (`/ready`):

```
{
  "status": "ready",
  "timestamp": "2025-10-23T12:00:00.000Z",
  "checks": {
    "database": {
      "status": "connected",
      "lastCheck": "2025-10-23T12:00:00.000Z",
      "error": null
    },
    "slack": {
      "status": "connected",
      "lastCheck": "2025-10-23T12:00:00.000Z",
      "error": null
    }
  }
}
```

Not Ready Response (`/ready`):

```
{
  "status": "not ready",
  "timestamp": "2025-10-23T12:00:00.000Z",
  "checks": {
    "database": {
      "status": "disconnected",
      "lastCheck": "2025-10-23T12:00:00.000Z",
      "error": "connection timeout"
    },
    "slack": {
      "status": "connected",
      "lastCheck": "2025-10-23T12:00:00.000Z",
      "error": null
    }
  }
}
```

Automated Tests

```
// health.test.js
const request = require('supertest');
const HealthServer = require('./health');

describe('Health Check Endpoints', () => {
  let healthServer;

  beforeAll(async () => {
    healthServer = new HealthServer(3002);
    await healthServer.start();
  });

  afterAll(async () => {
    await healthServer.stop();
  });

  test('GET /health returns 200', async () => {
    const response = await request(healthServer.app).get('/health');
    expect(response.status).toBe(200);
    expect(response.body.status).toBe('ok');
    expect(response.body.service).toBe('pp-bot');
  });

  test('GET /ready returns 503 when not ready', async () => {
    healthServer.setReady(false);
    const response = await request(healthServer.app).get('/ready');
    expect(response.status).toBe(503);
    expect(response.body.status).toBe('not ready');
  });

  test('GET /ready returns 200 when ready', async () => {
    healthServer.setReady(true);
    healthServer.setDatabaseConnected(true);
    healthServer.setSlackConnected(true);

    const response = await request(healthServer.app).get('/ready');
    expect(response.status).toBe(200);
    expect(response.body.status).toBe('ready');
  });

  test('GET /metrics returns system metrics', async () => {
    const response = await request(healthServer.app).get('/metrics');
    expect(response.status).toBe(200);
    expect(response.body).toHaveProperty('uptime_seconds');
    expect(response.body).toHaveProperty('memory_usage');
  });
});
```

Monitoring Setup

Example Monitoring Script

```
#!/bin/bash
# monitor-health.sh

HEALTH_URL="https://pp-bot-production.up.railway.app/health"
SLACK_WEBHOOK_URL="your-slack-webhook-url"

response=$(curl -s -o /dev/null -w "%{http_code}" $HEALTH_URL)

if [ $response -ne 200 ]; then
  curl -X POST $SLACK_WEBHOOK_URL \
    -H 'Content-Type: application/json' \
    -d "{\"text\":\"⚠️ pp-bot health check failed! HTTP $response\"}"
fi
```

Uptime Monitoring Services

- UptimeRobot (https://uptimerobot.com/) - Free tier available
- Pingdom (https://www.pingdom.com/)
- Better Uptime (https://betteruptime.com/)
- Healthchecks.io (https://healthchecks.io/)

Acceptance Criteria

- [ ] `/health` endpoint returns 200 OK when application is running
- [ ] `/ready` endpoint returns 200 when ready, 503 when not ready
- [ ] `/startup` endpoint tracks startup completion
- [ ] Health server runs on separate port from main app
- [ ] Database connectivity check works correctly
- [ ] Slack connectivity check works correctly
- [ ] Graceful shutdown marks service as not ready
- [ ] Railway.com uses health check for deployments
- [ ] Automated tests cover all endpoints
- [ ] Documentation includes monitoring setup

Documentation Requirements

- [ ] Update README.md with health check information
- [ ] Add monitoring guide to DEPLOYMENT.md
- [ ] Document endpoint specifications
- [ ] Include example monitoring setup
- [ ] Add troubleshooting guide for health check failures

Resources

- Health Check Best Practices (https://microservices.io/patterns/observability/health-check-api.html)
- Railway Health Checks (https://docs.railway.app/deploy/healthchecks)

- Express.js Documentation (https://expressjs.com/)
- Kubernetes Probes (for reference) (https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/)

Dependencies

Recommended before:
- Issue #2: PostgreSQL integration (for database health check)
- Issue #4: Railway.com deployment (to use health checks)