

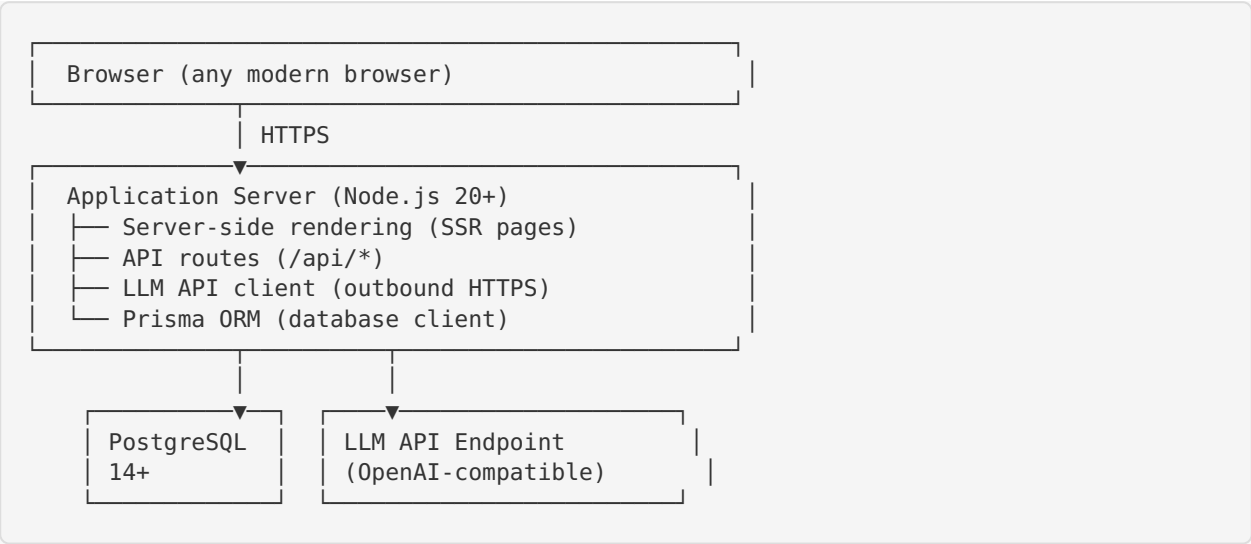
Deployment & Environment Guide

AI Operations Center — How to Run This in Your Environment

Document Owner: AiSO Platform Team
Last Updated: May 2026
Audience: DevOps, Platform Engineering, IT Architecture

Architecture Overview

The AI Operations Center is a full-stack web application with the following components:



Infrastructure Requirements

Minimum Specifications

Component	Requirement	Notes
Runtime	Node.js 20 LTS or later	Required for ES2022 features
Database	PostgreSQL 14+	15+ recommended for performance
Memory	2 GB RAM minimum	4 GB recommended for build step
Storage	1 GB for application	Database storage varies with data volume
Network	Outbound HTTPS to LLM API	No inbound ports required beyond web server

Cloud Provider Compatibility

Provider	Compute	Database	Tested
AWS	ECS Fargate, EC2, Lambda	RDS PostgreSQL, Aurora	✓
Azure	App Service, AKS, Container Apps	Azure Database for PostgreSQL	Compatible
GCP	Cloud Run, GKE	Cloud SQL for PostgreSQL	Compatible
On-Premise	Any Linux server with Node.js	Any PostgreSQL 14+ instance	Compatible

Environment Variables

Create a `.env` file in the application root with the following variables:

Variable	Required	Description	Example
<code>DATABASE_URL</code>	✓	PostgreSQL connection string	<code>postgresql://user:pass@host:5432/aiops?schema=public</code>
<code>ABACUSAI_API_KEY</code>	✓	LLM API key (see “LLM Provider” below)	<code>sk-abc123...</code>
<code>NEXTAUTH_URL</code>	✓	Public URL of the deployed application	<code>https://aiops.yourcorp.com</code>
<code>NEXTAUTH_SECRET</code>	✓	Random 32+ character secret for session signing	<code>openssl rand -base64 32</code>

Optional Variables

Variable	Description	Default
<code>LLM_MODEL</code>	Model identifier for the LLM provider	<code>gpt-5.4-mini</code>
<code>NODE_ENV</code>	Environment mode	<code>production</code>
<code>PORT</code>	Application server port	<code>3000</code>

LLM Provider Configuration

The application uses the OpenAI-compatible chat completions API (`/v1/chat/completions`). Any provider that implements this interface is compatible.

Supported Providers

Provider	Base URL	Model Examples
Abacus.AI (current)	<code>https://api.abacus.ai/api</code>	<code>gpt-5.4-mini</code>
Azure OpenAI	<code>https://{re-source}.openai.azure.com/openai</code>	<code>gpt-4o</code> , <code>gpt-4o-mini</code>
OpenAI Direct	<code>https://api.openai.com</code>	<code>gpt-4o</code> , <code>gpt-4o-mini</code>
AWS Bedrock	Via AWS SDK (requires adapter)	Claude 3.5, Llama 3
Self-hosted	Your endpoint	Llama 3, Mixtral

How to Swap LLM Providers

1. Update `ABACUSAI_API_KEY` with the new provider's API key
2. Search for `baseURL` in the codebase — update the API endpoint URL
3. Update the `LLM_MODEL` constant (found in `lib/agent-swarm*.ts` files) to the new model identifier
4. Test with: `curl -X POST /api/agent-swarm/run -d '{"mvp":"MVP1","caseId":"DED-00001"}'`

Note: The application uses structured JSON output from the LLM. Ensure your chosen model reliably produces JSON when instructed. GPT-4o class models or better are recommended.

Database Setup

Initial Setup

```
# 1. Create the database
createdb aiops_center

# 2. Set the connection string
export DATABASE_URL="postgresql://user:password@localhost:5432/aiops_center?schema=public"

# 3. Apply the schema (creates all 68 tables)
npx prisma db push

# 4. Generate the Prisma client
npx prisma generate

# 5. Seed reference data
curl -X POST http://localhost:3000/api/seed-data
curl -X POST http://localhost:3000/api/seed-extra
```

Schema Overview

The database contains 68 tables organized into 7 domains:

Domain	Tables	Key Models
Core Master Data	6	Customer, SKU, Supplier, TradeDeal
MVP1: Trade Deductions	12	Deduction, ClassificationResult, ValidityCheck, RemittanceArtifact, TradeDealValidity
MVP2: Collections	5	CollectionAccount, CollectionStrategy, CollectionInteraction, CustomerPaymentBehavior
MVP3: Invoice Triage	8	Invoice, InvoiceLineItem, InvoiceException, InvoiceValidityCheck, DuplicateInvoiceLink, ArbitrageOpportunity
MVP4: Buyer Copilot	8	PurchaseRequisition, BuyerProposal, BuyerPolicy, BuyerPolicyEvaluation, PurchaseOrder
MVP5: Demand Sensing	3	DemandSensor, DemandData
Governance & Platform	26	AgentRun, ActionProposal, HITLQueueItem, ValueLedgerEntry, MCPToolCall, AuditLogEntry, EvalResult, DriftMonitor, BoundaryCondition, PromptVersion, etc.

Migration Strategy

For production environments, use Prisma Migrate instead of `db push` :

```
# Generate migration files (review before applying)
npx prisma migrate dev --name initial_schema

# Apply to production
npx prisma migrate deploy
```

Build & Deploy

Local Development

```
# Install dependencies
yarn install

# Generate Prisma client
yarn prisma generate

# Start development server
yarn dev
```

Production Build

```
# Build optimized production bundle
yarn build

# Start production server
yarn start
```

The production build creates a standalone server in `.build/standalone/` that includes all dependencies and can be deployed as a single artifact.

Docker Deployment

```
FROM node:20-alpine AS builder
WORKDIR /app
COPY package.json yarn.lock ./
RUN yarn install --frozen-lockfile
COPY . .
RUN yarn prisma generate && yarn build

FROM node:20-alpine AS runner
WORKDIR /app
COPY --from=builder /app/.build/standalone ./
COPY --from=builder /app/public ./public
COPY --from=builder /app/.build/static ./build/static
EXPOSE 3000
CMD ["node", "server.js"]
```

How to Replace Stub Integrations

All external system calls flow through the MCP Gateway (`lib/mcp-gateway.ts`). Each system has a tool implementations map.

Step-by-Step for Each System

1. **Locate the stub** — In `lib/mcp-gateway.ts` , find the `toolImplementations` map entry for the target system
2. **Create an adapter file** — e.g., `lib/adapters/sap-s4.ts`
3. **Implement the adapter** — Call the real API, map response to the expected return type

4. **Replace the stub entry** — Point the `toolImplementations` entry to your adapter
5. **Test in shadow mode** — Run both stub and real adapter, compare outputs
6. **Cut over** — Remove the stub entry

Example: Replacing a Stub

```
// lib/adapters/sap-s4.ts
import { MCPToolCallRequest, MCPToolCallResult } from '../mcp-gateway';

export async function getDebitMemo(req: MCPToolCallRequest):
Promise<MCPToolCallResult> {
  const response = await fetch(`${SAP_BASE_URL}/odata/v4/API_DEBIT_MEMO_SRV/Debit-
MemoSet('${req.params.memoId}')`, {
    headers: { Authorization: `Bearer ${SAP_TOKEN}` },
  });
  const data = await response.json();
  return { success: true, data, latencyMs: /* measured */ };
}
```

Monitoring & Observability

Built-in Observability

Feature	Endpoint / Table	Description
Agent run traces	AgentRun table, /api/agent-swarm/list	Full reasoning chain for every agent execution
MCP tool call audit	MCPToolCall table, /api/mcp-health	Every external system call with latency and status
Drift detection	DriftMonitor table	Input/output distribution changes over time
Cost tracking	CostMeterEntry table	LLM token usage and cost per MVP
Eval results	EvalResult table, /api/test-runner	Golden case agreement scores

Recommended External Monitoring

Tool	Purpose
Datadog / New Relic	APM, error tracking, infrastructure metrics
PagerDuty / OpsGenie	Alerting for circuit breaker trips, SLA breaches
Grafana + Prometheus	Dashboard for custom metrics (latency, throughput, cost)
ELK / Splunk	Centralized log aggregation

Security Checklist for Production

- ☐ Database connection uses SSL (`?sslmode=require` in connection string)
- ☐ LLM API key stored in secrets manager (AWS Secrets Manager, Azure Key Vault, HashiCorp Vault)
- ☐ Application runs as non-root user
- ☐ HTTPS enforced at load balancer / reverse proxy
- ☐ CORS configured for your domain only
- ☐ Rate limiting enabled at API gateway level
- ☐ Database credentials rotated on schedule
- ☐ Audit log retention policy configured