AgentForge: Technical Implementation Guide & Quick Start

Version: 1.0 | Implementation Ready
Last Updated: December 6, 2025

TABLE OF CONTENTS

PAGE 1: Project Setup & Local Development

1.1 Prerequisites

```
# System requirements:
- Docker Desktop (latest)
- Node.js 18+ (frontend)
- Python 3.10+ (backend)
- Terraform 1.5+ (infrastructure)
- kubectl (Kubernetes CLI)
- Git

# Optional (but recommended):
- VS Code with extensions:
  - Python extension
  - Thunder Client (API testing)
  - Kubernetes extension
  - Git Graph
```

1.2 Clone & Initialize

```
# Clone repository
git clone https://github.com/AdityaShenvi/AgentForge.git
cd AgentForge
```

```
# Create .env files
cp .env.example .env.local       # Frontend config
cp backend/.env.example backend/.env

# Update environment variables
# Set:
# - NEXTAUTH_URL=http://localhost:3000
# - DATABASE_URL=postgresql://user:pass@localhost:5432/agentforge
# - REDIS_URL=redis://localhost:6379
# - OLLAMA_API_BASE=http://localhost:11434
# - CLERK_API_KEY=<your_clerk_key>
# - OPENROUTER_API_KEY=<your_openrouter_key>
```

1.3 Start Local Development Stack

```
# Method 1: Docker Compose (Recommended)
docker-compose up --build

# What spins up:
# - PostgreSQL (port 5432)
# - Redis (port 6379)
# - Ollama (port 11434)
# - MLflow tracking server (port 5000)
# - FastAPI backend (port 8000)
# - Next.js frontend (port 3000)
# - Prometheus (port 9090)
# - Grafana (port 3001)

# Method 2: Manual (for development)
# Terminal 1: Backend
cd backend
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8000

# Terminal 2: Frontend
cd frontend
npm install
npm run dev

# Terminal 3: Celery Worker
cd backend
celery -A app.workers.celery_app worker --loglevel=info
```

1.4 Verify Setup

```
# Check all services are running
curl http://localhost:3000        # Frontend
curl http://localhost:8000/docs   # FastAPI Swagger docs
curl http://localhost:5432        # PostgreSQL (should timeout, that's OK)
curl http://localhost:6379        # Redis (should timeout, that's OK)
curl http://localhost:11434/api/tags  # Ollama
curl http://localhost:5000        # MLflow
curl http://localhost:9090        # Prometheus
curl http://localhost:3001        # Grafana

# Test first agent creation
curl -X POST http://localhost:8000/api/v1/agents \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Test Agent",
    "description": "My first agent",
    "graph": {
      "nodes": [...],
      "edges": [...]
    }
  }'
```

PAGE 2: Frontend Implementation Guide

2.1 Project Structure

```
frontend/
  app/
      layout.tsx           # Root layout + providers
      page.tsx             # Home/landing page
      dashboard/
          page.tsx         # Main dashboard
      agents/
          page.tsx         # List agents
          [id]/
              page.tsx     # Agent detail + canvas editor
          new/
              page.tsx     # Create new agent
      runs/
          [runId]/
              page.tsx     # Run detail + trace viewer
      templates/
          page.tsx         # Agent templates gallery
```

```
    settings/
        page.tsx          # User settings, billing, API keys
  components/
    AgentCanvas.tsx       # React Flow canvas editor
    DashboardMetrics.tsx  # Real-time metrics charts
    TraceViewer.tsx       # OpenTelemetry trace timeline
    LogViewer.tsx         # Structured log viewer
    ModelSelector.tsx     # LLM model selection dropdown
    NavBar.tsx            # Top navigation
    Sidebar.tsx           # Left sidebar navigation
  lib/
    api.ts                # API client (React Query hooks)
    hooks.ts              # Custom React hooks
    auth.ts               # Clerk auth helpers
    types.ts              # TypeScript interfaces
    utils.ts              # Utility functions
  styles/
    globals.css           # Tailwind CSS imports
  package.json
  next.config.js
  tsconfig.json
```

2.2 Key Components Implementation

AgentCanvas Component (React Flow)

```tsx
// components/AgentCanvas.tsx
import React, { useCallback } from 'react';
import ReactFlow, {
  Node,
  Edge,
  addEdge,
  MiniMap,
  Controls,
  Background,
  useNodesState,
  useEdgesState,
  Connection,
} from 'reactflow';
import 'reactflow/dist/style.css';

// Custom node types
const nodeTypes = {
  input: InputNode,
  llm: LLMNode,
  tool: ToolNode,
```

```
  decision: DecisionNode,
  output: OutputNode,
};

export function AgentCanvas({ agentId }: { agentId: string }) {
  const [nodes, setNodes, onNodesChange] = useNodesState([]);
  const [edges, setEdges, onEdgesChange] = useEdgesState([]);

  // Fetch agent definition
  useEffect(() => {
    const fetchAgent = async () => {
      const res = await fetch(`/api/v1/agents/${agentId}`);
      const agent = await res.json();
      setNodes(agent.graph.nodes);
      setEdges(agent.graph.edges);
    };
    fetchAgent();
  }, [agentId]);

  const onConnect = useCallback(
    (connection: Connection) => {
      setEdges((eds) => addEdge(connection, eds));
    },
    [setEdges]
  );

  const handleSave = async () => {
    // POST updated graph to backend
    const response = await fetch(`/api/v1/agents/${agentId}`, {
      method: 'PATCH',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ graph: { nodes, edges } }),
    });
    // Show success toast
  };

  return (
    <div style={{ width: '100vw', height: '100vh' }}>
      <ReactFlow
        nodes={nodes}
        edges={edges}
        onNodesChange={onNodesChange}
        onEdgesChange={onEdgesChange}
        onConnect={onConnect}
        nodeTypes={nodeTypes}
      >
```

```
        <Background />
        <Controls />
        <MiniMap />
      </ReactFlow>
      <button onClick={handleSave}>Save & Deploy</button>
    </div>
  );
}
```

DashboardMetrics Component (Recharts)

```
// components/DashboardMetrics.tsx
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip } from 'recharts';
import { useQuery } from '@tanstack/react-query';

export function DashboardMetrics() {
  const { data: metrics } = useQuery({
    queryKey: ['metrics'],
    queryFn: () => fetch('/api/v1/metrics/agents').then(r => r.json()),
    refetchInterval: 5000, // Update every 5 seconds
  });

  if (!metrics) return <div>Loading...</div>;

  return (
    <div className="grid grid-cols-2 gap-4">
      <div className="p-4 bg-white rounded-lg shadow">
        <h3>Agent Latency (p95)</h3>
        <LineChart width={400} height={300} data={metrics.latency}>
          <CartesianGrid strokeDasharray="3 3" />
          <XAxis dataKey="timestamp" />
          <YAxis />
          <Tooltip />
          <Line type="monotone" dataKey="value" stroke="#8884d8" />
        </LineChart>
      </div>
      {/* More charts... */}
    </div>
  );
}
```

2.3 API Client (React Query)

```
// lib/api.ts
import { useQuery, useMutation } from '@tanstack/react-query';
```

```typescript
const API_BASE = process.env.NEXT_PUBLIC_API_URL || 'http://localhost:8000';

// Get agents list
export const useAgents = () => {
  return useQuery({
    queryKey: ['agents'],
    queryFn: async () => {
      const res = await fetch(`${API_BASE}/api/v1/agents`, {
        headers: { Authorization: `Bearer ${localStorage.getItem('token')}` },
      });
      return res.json();
    },
  });
};

// Create agent
export const useCreateAgent = () => {
  return useMutation({
    mutationFn: async (agent) => {
      const res = await fetch(`${API_BASE}/api/v1/agents`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${localStorage.getItem('token')}`,
        },
        body: JSON.stringify(agent),
      });
      return res.json();
    },
  });
};

// Execute agent
export const useExecuteAgent = () => {
  return useMutation({
    mutationFn: async ({ agentId, input }) => {
      const res = await fetch(`${API_BASE}/api/v1/agents/${agentId}/run`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${localStorage.getItem('token')}`,
        },
        body: JSON.stringify({ input }),
      });
      return res.json();
    },
```

```
  });
};
```

PAGE 3: Backend Implementation Guide

3.1 FastAPI Application Structure

```python
# backend/app/main.py
from fastapi import FastAPI, Depends
from fastapi.middleware.cors import CORSMiddleware
from app.api import agents, runs, metrics
from app.middleware.auth import auth_middleware
from app.middleware.otel import otel_middleware

app = FastAPI(title="AgentForge", version="1.0.0")

# Middleware
app.add_middleware(CORSMiddleware, allow_origins=["*"])
app.add_middleware(auth_middleware)
app.add_middleware(otel_middleware)

# Routes
app.include_router(agents.router, prefix="/api/v1/agents", tags=["agents"])
app.include_router(runs.router, prefix="/api/v1/runs", tags=["runs"])
app.include_router(metrics.router, prefix="/api/v1/metrics", tags=["metrics"])

@app.get("/health")
def health():
    return {"status": "ok"}
```

3.2 Agent CRUD Endpoints

```python
# backend/app/api/agents.py
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from app.models.agent import Agent
from app.models.user import User
from app.services.agent_service import AgentService

router = APIRouter()

@router.post("/")
async def create_agent(
    agent_data: dict,
    current_user: User = Depends(get_current_user),
```

```
    db: Session = Depends(get_db)
):
    """Create new agent"""
    service = AgentService(db)
    agent = service.create_agent(
        name=agent_data['name'],
        description=agent_data.get('description', ''),
        graph=agent_data['graph'],
        owner_id=current_user.id,
        org_id=current_user.org_id,
    )
    return agent.to_dict()

@router.get("/{agent_id}")
async def get_agent(
    agent_id: str,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Get agent by ID"""
    agent = db.query(Agent).filter(
        Agent.id == agent_id,
        Agent.org_id == current_user.org_id
    ).first()
    if not agent:
        raise HTTPException(status_code=404, detail="Agent not found")
    return agent.to_dict()

@router.patch("/{agent_id}")
async def update_agent(
    agent_id: str,
    agent_data: dict,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Update agent"""
    service = AgentService(db)
    agent = service.update_agent(
        agent_id=agent_id,
        org_id=current_user.org_id,
        updates=agent_data
    )
    return agent.to_dict()

@router.post("/{agent_id}/run")
async def execute_agent(
```

```
    agent_id: str,
    execution_data: dict,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    """Execute agent (enqueue task)"""
    # Check agent exists and belongs to user's org
    agent = db.query(Agent).filter(
        Agent.id == agent_id,
        Agent.org_id == current_user.org_id
    ).first()
    if not agent:
        raise HTTPException(status_code=404)

    # Enqueue execution task
    from app.workers.execute_agent import execute_agent_task
    task = execute_agent_task.delay(
        agent_id=agent_id,
        input_data=execution_data.get('input'),
        user_id=current_user.id,
        org_id=current_user.org_id,
    )

    return {
        "run_id": task.id,
        "status": "enqueued",
        "message": "Agent execution started"
    }
```

3.3 LangGraph Orchestration

```
# backend/app/services/langgraph_engine.py
from langgraph.graph import StateGraph, END
from langchain_core.language_model import BaseLanguageModel
from typing import Dict, Any
import json

class LangGraphEngine:
    def __init__(self, llm: BaseLanguageModel):
        self.llm = llm
        self.graph = None

    def build_graph_from_definition(self, agent_def: Dict[str, Any]):
        """Convert UI definition to LangGraph"""
        workflow = StateGraph(dict)
```

```
        # Parse nodes and edges from UI definition
        for node in agent_def['nodes']:
            if node['type'] == 'llm':
                # Add LLM node
                workflow.add_node(
                    node['id'],
                    self.create_llm_node(node)
                )
            elif node['type'] == 'tool':
                workflow.add_node(
                    node['id'],
                    self.create_tool_node(node)
                )
            elif node['type'] == 'decision':
                workflow.add_node(
                    node['id'],
                    self.create_decision_node(node)
                )

        # Add edges
        for edge in agent_def['edges']:
            workflow.add_edge(edge['source'], edge['target'])

        # Set entry point
        workflow.set_entry_point(agent_def['entryPoint'])
        workflow.set_finish_point(agent_def['exitPoint'])

        self.graph = workflow.compile()
        return self.graph

    def create_llm_node(self, node_def):
        """Create LLM processing node"""
        async def llm_node(state):
            response = await self.llm.agenerate_text(
                prompt=state.get('input', ''),
                temperature=node_def.get('temperature', 0.7),
                max_tokens=node_def.get('max_tokens', 1000),
            )
            state['output'] = response
            return state
        return llm_node

    def create_tool_node(self, node_def):
        """Create tool execution node"""
        async def tool_node(state):
            tool_name = node_def.get('tool')
```

```
            if tool_name == 'code_executor':
                # Execute code safely in E2B sandbox
                result = await self.execute_code(state.get('code', ''))
            elif tool_name == 'api_caller':
                result = await self.call_api(
                    url=node_def.get('url'),
                    method=node_def.get('method', 'GET')
                )
            state['tool_result'] = result
            return state
        return tool_node

    def create_decision_node(self, node_def):
        """Create decision/branching node"""
        async def decision_node(state):
            condition = node_def.get('condition')
            if eval(condition, {'state': state}):
                return {'next': node_def.get('true_branch')}
            else:
                return {'next': node_def.get('false_branch')}
        return decision_node

    async def run(self, input_data: Dict[str, Any]) -> Dict[str, Any]:
        """Execute the agent graph"""
        result = await self.graph.ainvoke({'input': input_data})
        return result
```

3.4 Celery Worker Task

```
# backend/app/workers/execute_agent.py
from celery import shared_task
from app.services.langgraph_engine import LangGraphEngine
from app.services.mlflow_logger import MLflowLogger
from app.models.run import Run
import time

@shared_task(bind=True, name='execute_agent')
def execute_agent_task(
    self,
    agent_id: str,
    input_data: dict,
    user_id: str,
    org_id: str
):
    """Execute agent and track metrics"""
    db = get_db()
```

```
# Create run record
run = Run(
    id=self.request.id,
    agent_id=agent_id,
    user_id=user_id,
    org_id=org_id,
    status='running',
    input=input_data,
)
db.add(run)
db.commit()

start_time = time.time()

try:
    # Get agent definition
    agent = db.query(Agent).get(agent_id)

    # Build LangGraph
    engine = LangGraphEngine(llm=get_llm(agent.model))
    graph = engine.build_graph_from_definition(agent.graph)

    # Execute
    result = asyncio.run(engine.run(input_data))

    # Log to MLflow
    mlflow_logger = MLflowLogger()
    mlflow_logger.log_run(
        agent_id=agent_id,
        run_id=run.id,
        params=agent.graph.get('params', {}),
        metrics={
            'latency_ms': (time.time() - start_time) * 1000,
            'success': True,
        },
        output=result,
    )

    # Update run record
    run.status = 'completed'
    run.output = result
    run.duration = time.time() - start_time
    db.commit()

    return {'status': 'success', 'result': result}
```

```
    except Exception as e:
        # Log error
        run.status = 'failed'
        run.error = str(e)
        db.commit()

        raise
```

PAGE 4: Agent Runtime (LangGraph + Ollama)

4.1 Ollama Setup & Model Management

```
# Install Ollama (macOS/Linux/Windows)
# https://ollama.ai

# Run Ollama service
ollama serve

# In another terminal, pull models
ollama pull llama3.2        # Small, CPU-optimized
ollama pull mistral         # Larger model for complex tasks

# Test locally
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt": "Why is the sky blue?"
}'
```

4.2 Ollama Integration with FastAPI

```
# backend/app/services/ollama_client.py
import httpx
import os

class OllamaClient:
    def __init__(self):
        self.base_url = os.getenv('OLLAMA_API_BASE', 'http://localhost:11434')
        self.client = httpx.AsyncClient()

    async def generate(
        self,
        model: str,
        prompt: str,
        temperature: float = 0.7,
```

```
        top_p: float = 0.9,
        max_tokens: int = 1000,
    ):
        """Generate text using Ollama"""
        response = await self.client.post(
            f"{self.base_url}/api/generate",
            json={
                "model": model,
                "prompt": prompt,
                "temperature": temperature,
                "top_p": top_p,
                "num_predict": max_tokens,
                "stream": False,
            }
        )
        response.raise_for_status()
        return response.json()["response"]

    async def embeddings(
        self,
        model: str,
        text: str,
    ):
        """Generate embeddings"""
        response = await self.client.post(
            f"{self.base_url}/api/embeddings",
            json={"model": model, "prompt": text}
        )
        response.raise_for_status()
        return response.json()["embedding"]

# Usage in LangGraph
ollama_client = OllamaClient()

async def llm_node(state):
    response = await ollama_client.generate(
        model="llama3.2",
        prompt=state["input"],
        temperature=0.7,
    )
    return {"output": response}
```

4.3 Tool Binding with E2B

```
# backend/app/services/tool_executor.py
import e2b
```

```python
class ToolExecutor:
    @staticmethod
    async def execute_code(code: str, language: str = "python"):
        """Execute code safely in E2B sandbox"""
        sandbox = e2b.Sandbox(
            template="Python 3.11",
        )

        result = sandbox.commands.run(
            code,
            background=False,
            timeout=30,
        )

        sandbox.close()

        return {
            "stdout": result.stdout,
            "stderr": result.stderr,
            "exit_code": result.exit_code,
        }

    @staticmethod
    async def call_api(url: str, method: str = "GET", headers: dict = None, data: dict = Nor
        """Call external API safely"""
        import httpx
        async with httpx.AsyncClient() as client:
            response = await client.request(
                method,
                url,
                headers=headers or {},
                json=data,
                timeout=30.0,
            )
            return {
                "status_code": response.status_code,
                "body": response.json() if response.headers.get('content-type') == 'applicat
            }

    @staticmethod
    async def query_database(connection_string: str, query: str):
        """Query database safely"""
        import sqlalchemy
        engine = sqlalchemy.create_engine(connection_string)
        with engine.connect() as conn:
```

```
    result = conn.execute(sqlalchemy.text(query))
    return result.fetchall()
```

PAGE 5: Infrastructure as Code (Terraform + Kubernetes)

5.1 Terraform AWS EKS Module

```
# terraform/aws_eks.tf

variable "environment" {
  type = string
  description = "dev or prod"
}

variable "region" {
  type = string
  default = "us-east-1"
}

resource "aws_eks_cluster" "agentforge" {
  name            = "agentforge-${var.environment}"
  version         = "1.27"
  role_arn        = aws_iam_role.eks_cluster_role.arn
  vpc_config {
    subnet_ids = aws_subnet.private[*].id
  }
}

resource "aws_eks_node_group" "agentforge" {
  cluster_name    = aws_eks_cluster.agentforge.name
  node_group_name = "agentforge-nodes-${var.environment}"
  node_role_arn   = aws_iam_role.eks_node_role.arn
  subnet_ids      = aws_subnet.private[*].id

  scaling_config {
    desired_size = var.environment == "prod" ? 5 : 2
    max_size     = var.environment == "prod" ? 10 : 5
    min_size     = var.environment == "prod" ? 3 : 1
  }

  instance_types = ["t3.large"]

  depends_on = [aws_iam_role_policy_attachment.eks_node_policy]
```

```
}

output "cluster_endpoint" {
  value = aws_eks_cluster.agentforge.endpoint
}

output "cluster_name" {
  value = aws_eks_cluster.agentforge.name
}
```

5.2 Kubernetes Helm Chart

```yaml
# helm/agentforge/Chart.yaml
apiVersion: v2
name: agentforge
version: 1.0.0
description: AgentForge - Cloud-Native AI Agent Platform
type: application
```

```yaml
# helm/agentforge/values.yaml
replicaCount:
  frontend: 2
  backend: 3
  worker: 5

image:
  repository: agentforge
  tag: latest

frontend:
  port: 3000
  resources:
    requests:
      memory: "256Mi"
      cpu: "200m"
    limits:
      memory: "512Mi"
      cpu: "500m"

backend:
  port: 8000
  resources:
    requests:
      memory: "512Mi"
      cpu: "500m"
    limits:
```

```
      memory: "1Gi"
      cpu: "1000m"

worker:
  resources:
    requests:
      memory: "1Gi"
      cpu: "1000m"
    limits:
      memory: "2Gi"
      cpu: "2000m"

postgres:
  enabled: true
  persistence:
    size: 20Gi

redis:
  enabled: true
  persistence:
    size: 5Gi

autoscaling:
  enabled: true
  minReplicas: 2
  maxReplicas: 10
  targetCPU: 70
```

5.3 ArgoCD GitOps Setup

```
# argocd/agentforge-app.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: agentforge
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/AdityaShenvi/AgentForge
    targetRevision: main
    path: k8s/

  destination:
    server: https://kubernetes.default.svc
```

```
    namespace: agentforge

  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
    - CreateNamespace=true
```

PAGE 6: Observability Stack Setup

6.1 Prometheus Configuration

```yaml
# observability/prometheus/prometheus.yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'fastapi'
    static_configs:
      - targets: ['backend:8000']
    metrics_path: '/metrics'

  - job_name: 'kubernetes'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true

rule_files:
  - 'rules.yaml'
```

6.2 Grafana Dashboards

```
{
  "dashboard": {
    "title": "AgentForge - Agent Health",
    "panels": [
      {
        "title": "Agent Success Rate",
        "targets": [
          {
```

```
            "expr": "rate(agent_executions_total{status='success'}[5m])"
          }
        ]
      },
      {
        "title": "Agent Latency (p95)",
        "targets": [
          {
            "expr": "histogram_quantile(0.95, agent_execution_duration_seconds)"
          }
        ]
      },
      {
        "title": "Cost per Agent Run",
        "targets": [
          {
            "expr": "avg(agent_run_cost_usd)"
          }
        ]
      }
    ]
  }
}
```

6.3 OpenTelemetry Instrumentation

```python
# backend/app/middleware/otel.py
from opentelemetry import trace, metrics
from opentelemetry.exporter.otlp.proto.grpc.trace_exporter import OTLPSpanExporter
from opentelemetry.exporter.otlp.proto.grpc.metric_exporter import OTLPMetricExporter
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import BatchSpanProcessor

# Configure trace exporter
otel_exporter_otlp_endpoint = "otel-collector:4317"

trace_provider = TracerProvider()
trace_provider.add_span_processor(
    BatchSpanProcessor(OTLPSpanExporter(endpoint=otel_exporter_otlp_endpoint))
)
trace.set_tracer_provider(trace_provider)

# Instrument FastAPI
from opentelemetry.instrumentation.fastapi import FastAPIInstrumentor
FastAPIInstrumentor().instrument()
```

```python
# Get tracer
tracer = trace.get_tracer(__name__)

async def otel_middleware(request, call_next):
    with tracer.start_as_current_span(f"{request.method} {request.url.path}"):
        response = await call_next(request)
    return response
```

PAGE 7: Deployment Checklist

7.1 Pre-Deployment

- ☐ All tests passing locally (`pytest`, frontend tests)
- ☐ Environment variables configured (.env files)
- ☐ Docker images built and tagged
- ☐ Terraform plan reviewed and approved
- ☐ Secrets encrypted (using Sealed Secrets or HashiCorp Vault)
- ☐ Database migrations tested
- ☐ Load testing passed (1000 agents/sec)
- ☐ Security scan passed (OWASP Top 10)

7.2 Deployment Steps

```bash
# 1. Deploy infrastructure
terraform init
terraform plan -out=tfplan
terraform apply tfplan

# 2. Save kubeconfig
aws eks update-kubeconfig --name agentforge-prod --region us-east-1

# 3. Create namespaces
kubectl create namespace agentforge
kubectl create namespace argocd

# 4. Deploy ArgoCD
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifes

# 5. Deploy AgentForge via ArgoCD
kubectl apply -f argocd/agentforge-app.yaml

# 6. Wait for rollout
kubectl rollout status deployment/agentforge-frontend -n agentforge

# 7. Verify
```

```
kubectl get pods -n agentforge
```

7.3 Post-Deployment Verification

- ☐ Frontend accessible at HTTPS endpoint
- ☐ Backend API responding (health check: GET /health)
- ☐ Database connected and initialized
- ☐ Redis connected
- ☐ Ollama running and models loaded
- ☐ Prometheus scraping metrics
- ☐ Grafana dashboards populated
- ☐ First agent creation successful
- ☐ Agent execution successful
- ☐ Logs in Loki accessible
- ☐ Traces in Tempo visible
- ☐ Alerts configured and tested

PAGE 8: Testing & Quality Assurance

8.1 Backend Testing

```python
# backend/tests/test_agents.py
import pytest
from httpx import AsyncClient
from app.main import app

@pytest.mark.asyncio
async def test_create_agent():
    async with AsyncClient(app=app, base_url="http://test") as client:
        response = await client.post(
            "/api/v1/agents",
            json={
                "name": "Test Agent",
                "description": "A test agent",
                "graph": {
                    "nodes": [...],
                    "edges": [...]
                }
            }
        )
        assert response.status_code == 201
        assert response.json()["id"] is not None

@pytest.mark.asyncio
async def test_execute_agent():
```

```python
    # Create agent
    agent = create_test_agent()

    # Execute
    response = await client.post(
        f"/api/v1/agents/{agent.id}/run",
        json={"input": "test input"}
    )

    assert response.status_code == 200
    assert response.json()["status"] == "enqueued"
```

8.2 Frontend Testing (Jest + React Testing Library)

```typescript
// frontend/__tests__/AgentCanvas.test.tsx
import { render, screen, fireEvent } from '@testing-library/react';
import { AgentCanvas } from '@/components/AgentCanvas';

describe('AgentCanvas', () => {
  it('renders canvas', () => {
    render(<AgentCanvas agentId="test-123" />);
    expect(screen.getByText(/Agent Canvas/i)).toBeInTheDocument();
  });

  it('saves agent on button click', async () => {
    render(<AgentCanvas agentId="test-123" />);
    fireEvent.click(screen.getByText('Save & Deploy'));

    // Verify API call
    await expect(fetch).toHaveBeenCalledWith(
      '/api/v1/agents/test-123',
      expect.objectContaining({ method: 'PATCH' })
    );
  });
});
```

8.3 Load Testing (K6)

```javascript
// tests/load.js
import http from 'k6/http';
import { check } from 'k6';

export let options = {
  vus: 100,
  duration: '5m',
};
```

```javascript
export default function() {
  let res = http.post('http://localhost:8000/api/v1/agents/test/run', {
    input: 'test input'
  });

  check(res, {
    'status 200': (r) => r.status === 200,
    'response time < 1s': (r) => r.timings.duration < 1000,
  });
}
```

QUICK REFERENCE COMMANDS

```bash
# Local development
docker-compose up

# Deploy to AWS
terraform apply -var-file=prod.tfvars

# Watch deployment
kubectl rollout status deployment/agentforge -n agentforge

# View logs
kubectl logs -f deployment/agentforge-backend -n agentforge

# Access database
kubectl port-forward svc/postgres 5432:5432 -n agentforge
psql postgresql://user:pass@localhost:5432/agentforge

# Monitor metrics
kubectl port-forward svc/prometheus 9090:9090 -n agentforge

# View traces
kubectl port-forward svc/grafana 3000:3000 -n agentforge
# Visit http://localhost:3000

# Scale agents
kubectl scale deployment/agentforge-worker --replicas=10 -n agentforge
```

Document Version: 1.0 | **Status:** Ready to Implement | **Last Updated:** December 6, 2025