KERNEX

Primary Startup Documentation

Control Plane + Runtime for Edge AI Operations

Author: Gourav Mukherjee

Solo Technical Founder, SJSU AI/ML Engineering

December 27, 2025

Version 1.0

Contents

Chapter 1

Executive Summary

1.1 What is Kernex?

Kernex is an infrastructure platform consisting of a **control plane** (cloud-hosted SaaS) and a **lightweight runtime** (deployed on edge and on-premise devices) that enables reliable deployment, versioning, observability, and rollback of AI agents running outside the cloud.

The core value proposition is operational: teams can train AI models locally, package them as versioned bundles, deploy to 100 edge devices in 5 minutes, and roll back to a known-good state in under 30 seconds—without SSH, without manual scripts, without field visits.

1.2 Who It's For

- **Primary Wedge**: System integrators and edge AI agencies building solutions for retail, logistics, manufacturing, and industrial customers.
- **Secondary**: Mid-market operators (20–500 devices) managing their own edge AI fleets.
- **Long-term**: Enterprise IT/OT teams running distributed intelligence at scale.

1.3 Why Now

1. Edge AI hardware market is USD 58B+ and growing 40% annually. Every vendor (NVIDIA, Qualcomm, Google) ships accelerators.
2. Software deployment success rate for edge AI pilots is <30%; projects stall in "pilot purgatory."

3. Companies lose millions in failed rollouts, manual device management, and field visits.

4. Cloud-native approaches (containers, K8s) fail on edge due to bandwidth, compute, and connectivity constraints.

5. Data sovereignty and regulatory pressures (GDPR, HIPAA, AI localization laws) mandate on-device processing.

1.4 Core Product Definition

Kernex is **not** about making better AI agents. Kernex is about making AI agent **operations** reliable, auditable, and scalable. It is the DevOps layer that enterprise and integrator teams have been manually scripting away for years.

Core loop: device enrollment → bundle upload → versioned deploy → instant rollback → audit trail. All from a single web dashboard. No SSH. No spreadsheets. No field visits.

Chapter 2

The Problem: Pilot Purgatory

2.1 Real-World Failure Modes

2.1.1 Update Hell

A retailer trains a Qwen 1.5B shoplifting detection model. It works in the lab. They deploy Docker to 10 stores via Ansible. Three weeks later, the model hallucinates on dark lighting. Retraining takes 1 week. Rolling out v2 requires:

- Manual VPN/SSH connections to 50 devices
- 2GB container upload (30 mins per device on 4G)
- Zero ability to rollback if v2 breaks something
- No visibility: which stores are running v1 vs v2 vs v1.5-hotfix?

2.1.2 Observability Blindness

A device is down. Is the model crashed? Network disconnected? Agent hung? No dashboard exists. IT staff must physically visit the location or SSH in to investigate. At $300/visit, this eats all the cost savings from local AI.

2.1.3 Bandwidth Tyranny

A factory has 500 devices on 4G. Broadcasting 2GB updates simultaneously causes network saturation. There is no scheduling mechanism. Devices fail mid-update.

2.1.4 Device Fragmentation

Some stores have Raspberry Pi 4s; others have Jetson Nanos; a few have x86 edge servers. One codebase does not fit all. Manual per-device configurations become chaos.

2.1.5 Version Chaos

Which devices run which version? No one knows. The spreadsheet tracking it is out of date. When a bug emerges, there is no way to know impact.

2.1.6 Audit & Compliance

Regulated industries need proof that exact code ran on exact devices at exact times, auditable to regulatory bodies. DIY deployments provide no chain of custody.

2.2 Why This Problem is Worsening

- **Hardware Proliferation**: Every vendor releases new accelerators. "Write once, run everywhere" is dead.
- **Model Velocity**: Open-source LLMs enable weekly retraining and deployment. Quarterly pipelines no longer exist.
- **Scale Explosion**: Companies pilot with 10 devices, then try to scale to 1000. Manual ops cannot scale.
- **Data Sovereignty**: GDPR, HIPAA, and emerging AI localization laws mandate data stays on-device. Cloud solutions no longer viable.
- **Talent Shortage**: Engineers who understand edge deployments are rare.

2.3 Consequence of Not Solving This

- **Pilot Purgatory**: Projects stall 3–6 months in. ROI never proven. Budgets shift.
- **Wasted Investment**: $200k spent training a model; cannot deploy profitably across 100+ devices.
- **Competitive Disadvantage**: Competitors who solve this own entire verticals.
- **Security Risk**: Manual deployments lead to inconsistent configs, unpatched devices, easy attack vectors.

Chapter 3

Product Thesis

3.1 "Agent Ops" vs. Other Paradigms

3.1.1 Not IoT Device Management

AWS IoT Core, Azure IoT Hub manage device firmware and OS. They are opaque to the application layer. Kernex is **agent-aware**: it understands model bundles, not just bits.

3.1.2 Not Kubernetes-at-Edge

K3s/ArgoCD are excellent for DevOps teams. They assume persistent connectivity, support arbitrary workloads, and add operational complexity. Kernex is laser-focused on AI agent deployment: smaller, bandwidth-aware, offline-tolerant.

3.1.3 Not DIY Scripts

SSH + Ansible + Python scripts are free but brittle, unscalable, and error-prone. Kernex trades a small fee for reliability, auditability, and peace of mind.

3.2 Core Principles

3.2.1 Versioning as First-Class

Every agent bundle has a semantic version (v1.0, v1.1, v2.0). Versions are immutable once deployed. Rollback means "revert to known version."

3.2.2 Immutability

Bundles are signed and checksummed at upload. Once in the system, they cannot be modified. Provenance is trackable.

3.2.3 Last-Known-Good Design

Every device keeps the previous bundle on disk. If new bundle crashes, device automatically reverts. No manual intervention.

3.2.4 Auditability

Every deployment, rollback, and configuration change is logged with timestamp, user, and device. Compliance audits are trivial.

3.2.5 Minimal Runtime

The device runtime is intentionally small and simple. It can run on Raspberry Pi, Jetson, or x86. It does not assume a package manager, systemd, or modern kernel.

3.2.6 Offline Tolerance

Devices are queued commands. When offline, they queue locally. On reconnect, they execute atomically. No lost messages; no stale state.

3.3 What Kernex Will NOT Be

- **NOT a model trainer**: Training happens upstream. Kernex ships pre-trained models.
- **NOT an inference optimizer**: Optimization is the agent's job. Kernex just runs it.
- **NOT a device provisioner**: Assume ethernet/WiFi already working. Kernex does not set up networks.
- **NOT an AI observability platform**: We log agent events; we do not analyze model quality or hallucination. (Later: integrate with Datadog, etc.)
- **NOT a mobile app**: Web UI only.
- **NOT a hardware management layer**: BIOS, firmware, thermal management—not Kernex's job.

Chapter 4

Users, Customers, and Wedge Strategy

4.1 Three Candidate ICPs

4.1.1 ICP A: System Integrators & Agencies

Definition Companies building edge AI solutions for customers. They handle architecture, training, deployment, support.

Size 10–100 engineers; $5M–$50M revenue; distributed teams.

Pain Using SSH, Ansible, hand-rolled scripts. Zero versioning. Zero rollback. 20+ hours/month on manual ops.

Who Feels Pain Deployment engineers, DevOps leads.

Who Pays VP Engineering or Project Manager.

TAM 5,000 integrators globally; 2% use edge AI; TAM = $1.5B.

4.1.2 ICP B: Mid-Market Operators

Definition Retail chains, logistics, manufacturers, hospitals deploying edge AI themselves (20–500 devices).

Size $50M–$2B revenue; IT teams: 5–50 people.

Pain Own staff managing deployments; ops teams lack technical expertise.

Who Feels Pain IT Director, Ops Manager.

Who Pays CIO or VP Operations.

TAM 30,000 candidates; 0.5% deployed edge AI; TAM = $2B.

4.1.3 ICP C: Enterprise IT/OT

Definition Large enterprises (GE, Siemens, banks, healthcare) with 500+ edge devices.

Size $2B+ revenue; IT/OT teams: 50–500 people.

Pain Legacy IoT platforms, custom deployments, no unified ops layer.

Who Feels Pain OT Directors, Security Officers.

Who Pays CIO or Chief Digital Transformation.

TAM 2,000 candidates; 1% deploying AI; TAM = $5B.

4.2 Recommended First Wedge: System Integrators

4.2.1 Why Integrators First

1. **Speed**: They want to ship to customers. Kernex becomes a competitive advantage they sell.
2. **Sales Cycles**: Technical founder can sell directly to VP Engineering. No procurement committees.
3. **Viral Distribution**: "Powered by Kernex Runtime" branding spreads as they deploy.
4. **Feedback**: They test at scale; you get real data and bugs.
5. **Monetization Readiness**: They understand SaaS and per-device pricing from day one.
6. **Lower Support Burden**: They are technical; they can self-serve.

4.2.2 Why NOT Mid-Market or Enterprise First

- Slow procurement (6–12 months).
- Demand 24/7 support, SOC 2, legal contracts (expensive).
- Multiple stakeholders; decision paralysis.

4.3 Initial ICP Persona: Dave (VP Engineering at Integrator)

Background Founded 2017, 40 engineers, $20M revenue, edge AI portfolio.

Pain "We shipped a model that hallucinates in bright light. Rolling back v1 took 6 hours because we SSH'd into 50 devices manually."

Motivation "I need a tool to deploy to 50 devices in 5 minutes and rollback in 30 seconds."

Authority Recommends + approves purchase.

Budget $5k–$50k/month for ops tools.

4.4 First 100 Users Acquisition

4.4.1 Distribution Loop 1: Cold Outreach

Email Template:

```
Subject: Deploying edge AI across 50+ devices?

Hi [First Name],

I noticed [Company] has deployed several edge AI
projects. I'm building Kernex---a control plane
for managing AI models on edge devices without
SSH scripts and manual rollbacks.

Current problem: Updating models to 50 devices
takes 6 hours. Rollback takes 2 hours. Kernex
does both in minutes.

Do you manage 30+ devices? Free 15-min call?

---
[Your name]
kernex.io
```

Outreach Volume: 20 emails/week; expect 2–5% response.

4.4.2 Distribution Loop 2: Community Presence

- **Jetson Community Forum**: Answer 2–3 questions/week on model deployment; mention "building a tool for this."
- **Edge AI Slack communities**: Introduce self; help others; share demo.
- **Reddit** (r/MachineLearning, r/embedded): "Built a deployment tool for edge AI. Show HN: Kernex"
- **Twitter/LinkedIn**: 2–3 posts/week (tips, case studies, product updates).

4.4.3 Distribution Loop 3: Powered by Kernex Runtime

When an integrator deploys a Kernex-managed solution to their customer, the dashboard shows: "Powered by Kernex Runtime (with link)." Customer interest flows back to Kernex directly, converting to operator accounts.

4.4.4 Distribution Loop 4: Content & SEO

- **Blog Post 1**: "Why 70% of Edge AI Pilots Fail—And How to Fix It"
- **Blog Post 2**: "Deploying GGUF Models to 100 Devices: A Step-by-Step Guide"
- **Blog Post 3**: "The Complete Guide to Model Rollback on Edge Devices"
- **Video Demo**: "From model to 100 devices in 5 minutes" (2 mins, YouTube)

Chapter 5

Competitive Landscape

5.1 Alternative 1: Generic IoT Fleet Management

Examples: AWS IoT Core, Azure IoT Hub, Cisco Kinetic

Strengths:

- Device enrollment at scale
- OTA firmware updates
- Basic telemetry

Weaknesses for Agents:

- Opaque to application layer; manage OS, not logic
- No rollback of agent bundles; only firmware
- Designed for sensors, not ML inference
- Vendor lock-in; expensive at scale
- No offline-safe queueing
- No "last-known-good" recovery

5.2 Alternative 2: Container-Based Edge

Examples: Docker + K3s + ArgoCD

Strengths:

- Declarative deployments
- Versioning and rollback

- Industry standard

Weaknesses for Agents:

- Container overhead: Qwen 1.5B + runtime + OS = 2–4GB; 30+ mins on 4G
- Assumes persistent connectivity (fails offline)
- Kubernetes overkill for 2–50 devices; adds complexity
- No agent-specific observability (CPU/memory only; no inference latency)
- Poor bandwidth efficiency (no delta updates)

5.3 Alternative 3: DIY Scripts

Examples: Ansible, Bash, custom Python

Weaknesses:

- Not scalable beyond 20 devices
- Error-prone; breaks every two weeks
- No observability
- Single point of failure (one person knows script)
- No audit trail
- Impossible to rollback safely

5.4 Differentiation Table

Dimension	**Kernex**	**IoT Hubs**	**K3s**	**DIY**	**In-House**
Agent-aware ops	✓	×	×	×	✓
Versioned bundles	✓	×	Containers	Manual	✓
1-click rollback	✓	×	Complex	Manual	✓
Offline-safe	✓	Limited	×	Manual	✓
Bandwidth optimized	✓	Limited	×	No	Maybe
Agent telemetry	✓	×	×	Manual	✓
Setup time	1h	2–4h	4–8h	Variable	Months
Integrator friendly	✓	×	Limited	×	N/A
Starting cost	Free	$500+/mo	$0	$0	$500k+

5.5 Why Kernex Wins

- **Agent-first, not generic**: We understand agents, not just devices.
- **Bandwidth-aware**: Delta updates, scheduling, offline-first.
- **Operator-friendly**: One dashboard; one click to deploy/rollback.
- **No bloat**: We do one thing reliably.

Chapter 6

Kernex v0/v1/v2 Product Specification

6.1 Kernex v0: Minimum Viable Operations

6.1.1 Goal

Prove core loop (deploy → monitor → rollback) with 2–5 devices in 8–10 weeks.

6.1.2 Core Features

Device Enrollment

- Device generates UUID.
- Device POSTs registration request to control plane.
- Control plane stores device metadata and issues device_id.
- Device performs heartbeat every 60 seconds.

Bundle Upload & Versioning

- Web UI: Upload tarball (model + config + manifest).
- System: Assign version number (v0.1, v0.2, ...).
- Compute SHA256; store in object storage.

Deploy to Device(s)

- Select bundle version.
- Select target device(s).
- Click "Deploy".

- Device polls, downloads bundle, verifies checksum.
- Device atomically swaps bundles (move, not overwrite).
- Device reports success/failure.

Rollback

- UI shows deployment history.
- Click "Rollback to v0.1".
- Device reverts to previous bundle.

Heartbeat & Basic Logs

- Device sends: `{device_id, agent_version, memory_mb, cpu_pct, status}`
- Control plane stores in DB.
- UI shows: "Device X running v0.2, healthy, last seen 30s ago".
- Tail logs from UI (stderr/stdout).

Web UI

- Dashboard: list devices (status, version, last heartbeat).
- Device detail: version, logs, rollback button.
- Bundle upload page.
- Release history.

6.1.3 Success Metrics

- Deploy v1 $\rightarrow$ v2 $\rightarrow$ v1 all succeed.
- Heartbeat latency $<$ 100ms.
- Bundle download $<$ 5 mins (even on slow network).
- All logs visible in UI.
- No crashes or resource leaks.

6.2 Kernex v1: Pilot-Ready

6.2.1 Goal

Support 5–20 devices per integrator; production-grade pilots.

6.2.2 Additional Features

Device Groups Tag devices; deploy to group.

Release Channels Dev → Staging → Prod channels.

Canary Deployments Deploy to 5 devices; auto-promote if healthy.

Offline-Safe Queueing Commands queue; execute atomically on reconnect.

Enhanced Observability Metrics dashboard; agent-specific latency; device health trends.

RBAC Admin, ops, read-only roles.

Audit Log Every action logged with user/timestamp.

6.2.3 Non-Goals (v1)

- No self-hosted control plane yet.
- No SSO/SAML (basic auth only).
- No delta updates.
- No external integrations.
- No tamper-evident logs.

6.3 Kernex v2: Scale & Enterprise

6.3.1 Timeline

Months 6–12 after achieving product-market fit.

6.3.2 Features

Delta Updates Only send changed weights (50MB vs 2GB).

Policy Engine "Auto-rollback if error rate $> 5\%$"; "Deploy at midnight UTC".

Tamper-Evident Logs Append-only with Merkle tree verification.

Self-Hosted Control Plane Docker Compose + air-gapped option.

SSO/SAML Okta, Azure AD, Google Workspace.

Advanced Analytics Model performance trends; predictive alerts.

Integrations Marketplace Datadog, PagerDuty, Slack, Splunk.

Multi-Tenancy Support managed service providers.

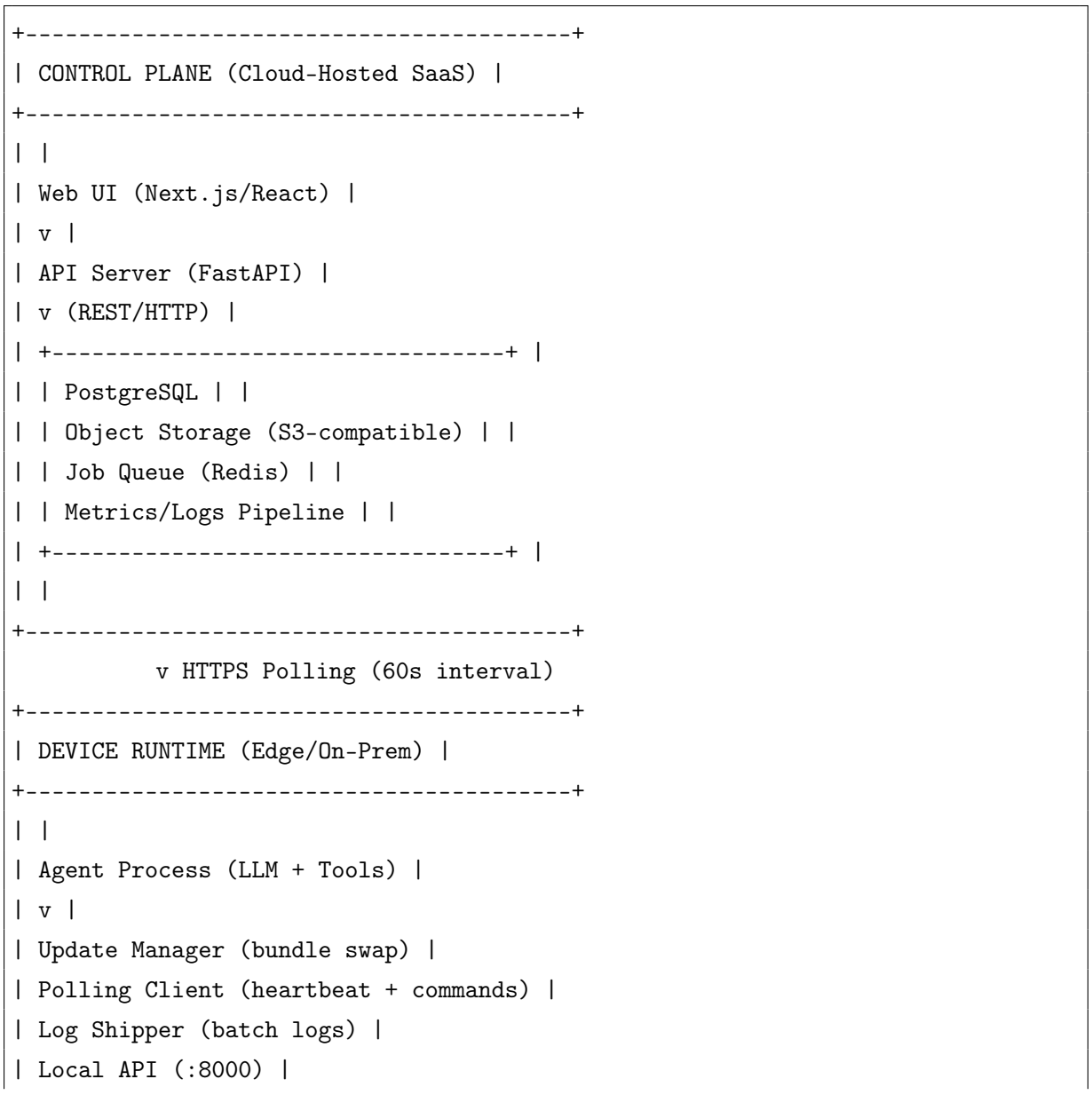
6.4 NOT in Scope (v0–v2)

- Hardware management (BIOS, firmware).
- Model training or fine-tuning.
- LLM inference optimization.
- Device networking setup.
- Mobile app (web UI only).

Chapter 7

System Architecture

7.1 High-Level Diagram



```
| |
| /var/kernex/ |
| +-- agent_bundle_current/ |
| +-- agent_bundle_previous/ |
| +-- staging/ |
| +-- logs/ |
| +-- private_key |
| |
+----------------------------------------+
```

7.2 Control Plane Components

7.2.1 Web UI (Next.js/React)

- Dashboard: devices, deployments, releases.
- Device detail page: status, logs, metrics, rollback.
- Bundle upload page.
- Release/deployment history.
- Audit log viewer.

7.2.2 API Server (FastAPI)

- REST endpoints for device enrollment, bundle upload, deploy, rollback, heartbeat, logs.
- Authentication: device token (device auth) and user JWT (web auth).
- Rate limiting, CORS, request logging middleware.

7.2.3 PostgreSQL Database

- Devices, Users, Organizations, Bundles, Deployments, Heartbeats, Logs, Audit trail.
- JSONB fields for flexible metadata.
- Indexes on frequently queried columns (device_id, deployment_status, timestamp).

7.2.4 Object Storage (S3-compatible)

- Store bundle tarballs (models + configs).
- Lifecycle policies: archive old bundles after 90 days.

7.2.5 Job Queue (Redis)

- Queue deployment jobs for background workers.
- Worker polls queue, orchestrates device deployments.

7.2.6 Background Workers

- Process deployment jobs.
- Aggregate heartbeat metrics.
- Cleanup old logs.
- Generate alerts.

7.3 Device Runtime Components

7.3.1 Agent Process

- Runs LLM inference with tool access.
- Emits metrics: inference time, tokens, memory.
- Supervised by systemd or custom process manager.

7.3.2 Update Manager

- Monitors control plane for new deployment commands.
- Downloads bundle from control plane (resumable HTTP).
- Verifies checksum.
- Atomically swaps: staging $\rightarrow$ current.
- Signals agent process to reload.
- On failure, reverts to previous bundle.

7.3.3 Polling Client

- Every 60 seconds: POST heartbeat.
- On heartbeat response: check for pending commands.
- If commands: fetch and queue for update manager.
- Post command result back to control plane.

7.3.4 Log Shipper

- Reads agent process logs (/var/kernex/agent.log).
- Batches and ships to control plane every 60s or on buffer fill.
- Handles offline: queues to local buffer; drains on reconnect.

7.3.5 Local API Server (Optional, v1)

- Exposes /v1/complete (OpenAI-compatible).
- Exposes /health, /metrics.
- Accessible to on-device applications only (localhost).

7.4 Data Flow Walkthroughs

7.4.1 Device Enrollment

```
Device (at boot):
  1. Generate RSA keypair (private stays local)
  2. POST /api/devices/register
     { public_key, device_type, hardware_metadata }
  3. Receive device_id + registration_token
  4. Store in /var/kernex/config.json
  5. Begin polling loop
```

7.4.2 Deploy Bundle

```
User:
  1. Web UI: Upload bundle (tarball)
  2. Control plane computes SHA256 checksum
```

```
  3. Stores bundle in S3 with version tag
  4. User clicks "Deploy v0.2 to Group A"

Control Plane:
  1. Creates Deployment record (status = pending)
  2. Queues deployment job

Device (60s polling loop):
  1. POST /api/heartbeat { device_id, status, ... }
  2. Receive response: { commands: [ deployment_cmd ] }
  3. Signal update manager

Update Manager:
  1. GET /api/bundles/v0.2 (streaming, resumable)
  2. Verify SHA256 checksum
  3. Extract to /var/kernex/staging/
  4. Atomic move: staging -> current
  5. Signal agent process via SIGHUP
  6. Agent reloads config, reloads model
  7. POST /api/deployments/{id}/result { status: success }

Control Plane:
  1. Marks Deployment as success
  2. Updates Device.current_bundle_version
```

7.4.3 Rollback

```
User:
  1. Web UI: Device detail -> "Rollback to v0.1"
  2. Control plane queues rollback command

Device (60s polling):
  1. Receives rollback command
  2. Update manager: atomic move: previous -> current
  3. Signal agent process
  4. POST result back
```

Chapter 8

Data Models and Schemas

8.1 Entity Relationship Overview

```
Organization (1) --< (M) User
Organization (1) --< (M) Device
Organization (1) --< (M) Bundle
Organization (1) --< (M) Deployment
Device (1) --< (M) Heartbeat
Device (1) --< (M) Log
Deployment references Bundle + Device list
```

8.2 SQL Schema (PostgreSQL)

8.2.1 Organizations

```
CREATE TABLE organizations (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name VARCHAR(255) NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
  metadata JSONB DEFAULT '{}'::JSONB
);
```

8.2.2 Users

```
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id UUID NOT NULL REFERENCES organizations(id),
```

```
  email VARCHAR(255) NOT NULL,
  hashed_password VARCHAR(255) NOT NULL,
  role VARCHAR(50) CHECK (role IN ('admin','ops','read_only')),
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  UNIQUE(org_id, email)
);
```

8.2.3 Devices

```
CREATE TABLE devices (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id UUID NOT NULL REFERENCES organizations(id),
  device_id VARCHAR(255) NOT NULL UNIQUE,
  device_type VARCHAR(100),
  hardware_metadata JSONB,
  current_bundle_version VARCHAR(50),
  public_key TEXT NOT NULL,
  status VARCHAR(50) CHECK (status IN
    ('online','offline','error')),
  last_heartbeat TIMESTAMP,
  registered_at TIMESTAMP NOT NULL DEFAULT NOW(),
  tags JSONB DEFAULT '{}'::JSONB,
  UNIQUE(org_id, device_id)
);
```

8.2.4 Bundles

```
CREATE TABLE bundles (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id UUID NOT NULL REFERENCES organizations(id),
  version VARCHAR(50) NOT NULL,
  model_name VARCHAR(255),
  model_size_mb INTEGER,
  checksum_sha256 VARCHAR(64) NOT NULL,
  signature VARCHAR(512),
  manifest JSONB,
  storage_path VARCHAR(512),
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  UNIQUE(org_id, version)
);
```

8.2.5 Deployments

```
CREATE TABLE deployments (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  org_id UUID NOT NULL REFERENCES organizations(id),
  bundle_id UUID NOT NULL REFERENCES bundles(id),
  status VARCHAR(50) CHECK (status IN
    ('pending','in_progress','success','failed','rolled_back')),
  target_device_ids UUID[] DEFAULT '{}',
  created_by UUID REFERENCES users(id),
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  completed_at TIMESTAMP,
  error_message TEXT,
  INDEX (org_id, status)
);
```

8.2.6 Heartbeats

```
CREATE TABLE heartbeats (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  device_id UUID NOT NULL REFERENCES devices(id),
  agent_version VARCHAR(50),
  memory_mb INTEGER,
  cpu_pct FLOAT,
  status VARCHAR(50) CHECK (status IN
    ('healthy','degraded','error')),
  timestamp TIMESTAMP NOT NULL DEFAULT NOW(),
  INDEX (device_id, timestamp)
);
```

8.2.7 Logs

```
CREATE TABLE logs (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  device_id UUID NOT NULL REFERENCES devices(id),
  level VARCHAR(20) CHECK (level IN
    ('debug','info','warn','error')),
  message TEXT NOT NULL,
  timestamp TIMESTAMP NOT NULL DEFAULT NOW(),
  INDEX (device_id, timestamp)
```

```
);
```

8.3 JSON Examples

8.3.1 Device Registration Payload

```
{
  "public_key": "-----BEGIN RSA PUBLIC KEY-----\n...",
  "device_type": "raspberry_pi",
  "hardware_metadata": {
    "ram_gb": 8,
    "cpu_cores": 4,
    "storage_gb": 32,
    "os": "Raspberry Pi OS"
  }
}
```

8.3.2 Bundle Manifest

```
{
  "version": "1.0",
  "model": {
    "name": "qwen-1.5b-instruct",
    "format": "gguf",
    "quantization": "Q4_K_M"
  },
  "system_prompt": "You are a helpful assistant...",
  "tools": [
    {
      "name": "search",
      "description": "Search the web",
      "endpoint": "/api/search"
    }
  ],
  "runtime_version_min": "0.1.0",
  "metadata": {
    "trained_date": "2025-12-01",
    "fine_tuned_on": ["retail", "surveillance"]
  }
}
```

8.3.3 Deployment Payload

```
{
  "bundle_version": "1.2",
  "target_devices": ["device-001", "device-002"],
  "scheduled_time": "2025-12-27T06:00:00Z",
  "description": "Shoplifting model v1.2 with better night lighting"
}
```

8.4 Data Retention Policies

- **Heartbeats**: Keep 7 days; aggregate to daily after 24 hours.
- **Logs**: Keep 30 days in hot storage; archive to cold storage after.
- **Deployments**: Keep forever (audit trail).
- **Bundles**: Keep forever if referenced by active devices; delete if unused for 1 year.

Chapter 9

API Design

9.1 MVP REST Endpoints

9.1.1 Device Endpoints

```
POST /api/v1/devices/register
  Request: { public_key, device_type, hardware_metadata }
  Response: { device_id, registration_token }

GET /api/v1/devices
  Query: ?org_id=...&limit=100&offset=0
  Response: { devices: [...], total: 150 }

GET /api/v1/devices/{device_id}
  Response: { device, last_heartbeat, current_bundle, status }

POST /api/v1/devices/{device_id}/heartbeat
  Request: { agent_version, memory_mb, cpu_pct, status }
  Response: { commands: [...] }
```

9.1.2 Bundle Endpoints

```
POST /api/v1/bundles
  Multipart: file (tarball), manifest
  Response: { bundle_id, version, checksum_sha256 }

GET /api/v1/bundles
  Response: { bundles: [...] }

```

```
GET /api/v1/bundles/{bundle_id}
  Streaming download (supports Range header)

POST /api/v1/bundles/{bundle_id}/verify
  Request: { checksum_sha256 }
  Response: { valid: true/false }
```

9.1.3 Deployment Endpoints

```
POST /api/v1/deployments
  Request: { bundle_version, target_devices, description }
  Response: { deployment_id, status }

GET /api/v1/deployments
  Response: { deployments: [...] }

GET /api/v1/deployments/{deployment_id}
  Response: { deployment, target_devices, progress }

POST /api/v1/deployments/{deployment_id}/rollback
  Request: { target_version }
  Response: { deployment_id, status: 'pending' }

POST /api/v1/deployments/{deployment_id}/result
  Request: { device_id, status, error_message }
  Response: { success: true }
```

9.1.4 Logs Endpoints

```
POST /api/v1/logs/batch
  Request: [
    { device_id, level, message, timestamp },
    ...
  ]
  Response: { ingested: 10 }

GET /api/v1/devices/{device_id}/logs
  Query: ?limit=100&offset=0&level=error
  Response: { logs: [...], total: 500 }
```

9.2 HTTP Status Codes and Error Handling

- **200 OK**: Success.
- **201 Created**: Resource created.
- **400 Bad Request**: Invalid input.
- **401 Unauthorized**: Missing/invalid token.
- **403 Forbidden**: Permission denied.
- **404 Not Found**: Resource not found.
- **409 Conflict**: State conflict (e.g., bundle already exists).
- **500 Internal Server Error**: Server error.

9.3 Idempotency and Retries

- All POST requests must be idempotent; use Idempotency-Key header.
- Device polls every 60 seconds; retries on network error (exponential backoff: 1s, 2s, 4s, max 60s).
- Bundle download supports HTTP Range header for resumable transfers.

9.4 Authentication

9.4.1 Device Authentication

Device includes signature in HTTP header: `X-Device-Signature: RSA_sign(payload, private_key)`. Control plane verifies signature using stored public key.

9.4.2 User Authentication

User login via email + password; receives JWT token (expires in 24 hours). Requests include: `Authorization: Bearer <JWT>`.

9.5 Rate Limiting

- Per device: 100 requests/minute.
- Per user: 1000 requests/minute.

- Per bundle upload: 10 per minute.

Chapter 10

Runtime Agent Bundle Specification

10.1 Bundle File Structure

```
kernex-bundle-v0.2.tar.gz
+-- manifest.json
+-- model.gguf
+-- config.json
+-- tools/
    +-- search.json
    +-- api_gateway.json
```

10.2 Manifest Format

```
{
  "version": "1.2.0",
  "runtime_version_min": "0.1.0",
  "runtime_version_max": "1.0.0",
  "model": {
    "name": "qwen-1.5b-instruct",
    "format": "gguf",
    "size_mb": 1024,
    "quantization": "Q4_K_M",
    "sha256": "abc123..."
  },
  "system_prompt": "You are...",
  "tools": [...],
  "env_vars": {
    "INFERENCE_TIMEOUT": "30",
    "MAX_TOKENS": "2048"
```

```
  },
  "metadata": {
    "created_date": "2025-12-01",
    "trained_on": ["retail", "surveillance"],
    "performance_notes": "20% faster inference on RTX 5070"
  }
}
```

10.3 Integrity Verification

- Control plane computes SHA256 of entire tarball at upload.
- Device downloads tarball and verifies SHA256 locally.
- Mismatch triggers rollback and alert.
- Later (v2): Add RSA signature + Merkle tree for tamper-proof bundles.

10.4 Device Disk Layout

```
/var/kernex/
+-- config.json (runtime config, device_id, control plane URL)
+-- agent_bundle_current/ (active bundle)
| +-- manifest.json
| +-- model.gguf
| +-- config.json
| +-- tools/
+-- agent_bundle_previous/ (rollback target)
| +-- manifest.json
| +-- ...
+-- staging/ (download + extract here before atomic move)
+-- logs/ (local log files, rotated)
| +-- agent.log.1, agent.log.2, ...
+-- pending_commands (JSON: commands received while offline)
+-- private_key (asymmetric signing, never transmitted)
```

10.5 Rollback Procedure (Atomic)

```
1. Device receives rollback command for v0.1
2. Verify /var/kernex/agent_bundle_previous/ exists + integrity valid
3. RENAME /var/kernex/agent_bundle_current -> /tmp/backup
4. RENAME /var/kernex/agent_bundle_previous -> /var/kernex/agent_bundle_current
5. RENAME /tmp/backup -> /var/kernex/agent_bundle_previous
6. SIGHUP agent process to reload
7. Verify agent restarted with old version
8. POST result to control plane
```

If step 2–7 fails: restore /tmp/backup and alert.

Chapter 11

Security Model

11.1 Threat Model

11.1.1 Attack Vectors

1. Attacker intercepts bundle download; injects malicious model.
2. Attacker gains device credentials; deploys unauthorized bundle.
3. Attacker modifies device private key; forges commands.
4. Attacker eavesdrops on device ↔ control plane traffic.
5. Attacker exploits API to access other organizations' bundles.
6. Insider modifies audit logs to hide malicious deployments.

11.2 Device Identity & Secrets

11.2.1 Device Registration (One-Time)

1. Device generates RSA 4096 keypair (private stored in /var/kernex/private_key; never transmitted).
2. Device POSTs public key + metadata to control plane over HTTPS.
3. Control plane issues device_id (UUID) + registration_token.
4. Device stores device_id and registration_token in /var/kernex/config.json.

11.2.2 Ongoing Authentication

- Device includes device_id in every HTTP request.
- Device computes RSA signature of request body.
- Control plane verifies signature using stored public key.

11.3 Bundle Integrity

11.3.1 MVP (v0): SHA256 Checksum

1. Control plane computes SHA256 hash of bundle tarball.
2. Device downloads bundle + checksum from control plane.
3. Device computes SHA256 locally and compares.
4. Mismatch triggers rollback + alert.

11.3.2 Production (v2): RSA Signature + Merkle Tree

1. Control plane signs bundle checksum: `sig = RSA_sign(hash, control_plane_private_key)`.
2. Device downloads bundle + signature.
3. Device verifies: `RSA_verify(sig, hash, control_plane_public_key)`.
4. Tamper-evident logs: each log entry includes hash of previous entry.

11.4 Transport Security (TLS)

- All control-plane ↔ device communication over HTTPS (TLS 1.3 minimum).
- Device pins control plane CA certificate (hardcoded in runtime binary).
- Prevents MITM attacks on bundle download and command delivery.

11.5 Least Privilege & Access Control

- Device API (local :8000) accessible only to localhost (or internal VPN).
- Control plane API requires user JWT (24-hour expiry).
- RBAC (v1): admin, ops, read-only roles.

- Device-level permissions: ops user A can deploy to devices tagged "region-us-west".
- Bundle access: users can only view/deploy bundles created by their organization.

11.6 Audit & Compliance

11.6.1 Audit Log (v1)

- Every deployment, rollback, config change logged: who, what, when, device, result.
- Stored in PostgreSQL with append-only semantics.
- Queryable by user, device, timestamp, action type.

11.6.2 Tamper-Evident Log (v2)

- Append-only log where each entry includes SHA256 hash of previous entry.
- Proof that logs have not been modified retroactively.
- Exportable for compliance audits (HIPAA, GDPR).

11.7 Practical Hardening Checklist

- ✓Enforce HTTPS everywhere (TLS 1.3+).
- ✓Use strong RSA 4096 keys; rotate annually.
- ✓Store bundle signing key in KMS (AWS Secrets Manager or HashiCorp Vault).
- ✓Pin device CA certificate; fail-safe on cert mismatch.
- ✓Implement rate limiting (100 requests/min per device).
- ✓Log all API calls with device_id, action, timestamp.
- ✓Encrypt sensitive data in transit and at rest.
- ✓Sanitize all user inputs; parameterized SQL queries.
- ✓No hardcoded credentials; use environment variables or secrets manager.
- ✓Regular security audits (quarterly in production).

Chapter 12

Reliability Model

12.1 State Machines

12.1.1 Device State Machine

```
online
  v (heartbeat not received for 30 mins)
offline
  v (heartbeat received)
online

online + memory > 90%
  v (alert)
online [degraded]
  v (memory freed)
online
```

12.1.2 Deployment State Machine

```
pending
  v (worker processes)
in_progress
  v (all devices report success)
success
  v (if user clicks rollback)
rolled_back

in_progress
  v (any device reports failure)
```

```
failed
  v (if user clicks retry)
pending
```

12.2 Crash Recovery & Last-Known-Good Design

12.2.1 Scenario: New Bundle Crashes Immediately

```
1. Device starts with bundle v1.0 (healthy)
2. Receives deployment command for v1.1
3. Downloads v1.1 to /var/kernex/staging/
4. Verifies checksum, signature OK
5. Atomic moves: v1.0 -> /var/kernex/previous/
6. Atomic moves: v1.1 (staging) -> /var/kernex/current/
7. Sends SIGHUP to agent process
8. Agent process reloads model + config (v1.1)
9. Agent crashes within 10 seconds (OOM, segfault, etc.)
10. systemd auto-restarts agent (max 3 retries in 5 mins)
11. Agent crashes again; systemd gives up
12. Runtime detects: too many crashes; marks agent_status = error
13. Runtime automatically reverts:
    - RENAME /var/kernex/previous/ -> /var/kernex/current/
    - Restarts agent with v1.0
14. Agent restarts successfully (v1.0)
15. Runtime POSTs to control plane:
    { device_id, deployment_id, status: 'failed_auto_rollback',
      reason: 'bundle_crashed_3_times' }
16. Control plane marks deployment as FAILED, v1.1 as unstable
```

12.3 Offline-First Behavior

12.3.1 Scenario: Device Loses Internet for 2 Hours

```
1. Device tries heartbeat; network unreachable
2. Queues heartbeat to /var/kernex/pending_commands (JSON)
3. Every 5 mins: retry heartbeat; fail silently
4. Update manager checks for pending commands; finds none
5. Agent continues running (last-known-good v1.0)
6. After 2 hours: network returns
```

```
7. Polling client resumes heartbeats successfully
8. Control plane response includes pending_commands []
9. Device executes all pending atomically (if any)
10. Clears pending queue
11. Returns to normal polling loop
```

12.3.2 Eventual Consistency

- Device state in control plane eventually consistent.
- UI may show device as "offline" for up to 60 seconds after reconnect.
- Logs may arrive out-of-order from multiple devices; reconcile by timestamp.

12.4 Rollout Safety

12.4.1 Staged Rollout (v1)

```
User deploys v2.0 to 1000 devices with canary enabled:
  1. Deploy to canary_group (5 devices, tagged "canary")
  2. Wait 1 hour; monitor error rate
  3. If error_rate < 1%: auto-promote to prod_group (995 devices)
  4. If error_rate > 5%: auto-rollback canary + alert user
```

12.4.2 Bandwidth Scheduling (v2)

```
User deploys v2.0 to 500 devices with schedule=staggered:
  1. Control plane computes slots:
     device-001 @ 00:00 UTC
     device-002 @ 00:05 UTC
     ...
     device-500 @ 02:00 UTC
  2. Device checks for pending commands every 60s
  3. At scheduled slot, device becomes eligible for deploy
  4. Download occurs during assigned 5-min window
  5. Total bandwidth spread over 2 hours instead of 5 minutes
  6. Network stays healthy; no congestion
```

Chapter 13

Observability

13.1 What to Log from Runtime

13.1.1 Agent Process Logs

- stdout/stderr from agent process (forwarded to /var/kernex/agent.log).
- Format: `[timestamp] [level] [component] message`.
- Levels: debug, info, warn, error.

13.1.2 Runtime Events

- Device startup: "Runtime started, device_id=..., version=..."
- Heartbeat: "Heartbeat sent, agent_version=..., memory=..., status=..."
- Command received: "Deploy command received, bundle=v1.2, device count=500"
- Bundle download started/completed: "Downloaded v1.2 (1024MB) in 180s"
- Bundle verification: "Checksum verified OK for v1.2"
- Atomic move: "Bundle swap: v1.0 -> previous, v1.2 -> current"
- Agent reload: "Signaled agent process to reload; waiting for restart..."
- Crash recovery: "Agent crashed 3 times; reverting to v1.0"

13.2 Metrics to Collect

13.2.1 Device Metrics

- Last heartbeat timestamp.

- Current bundle version.
- Device status (online/offline/degraded/error).
- Agent process memory usage (MB).
- Agent process CPU usage (%).
- Uptime (% in last 7 days).
- Last successful deployment (timestamp).

13.2.2 Agent-Specific Metrics

- Inference latency (p50, p95, p99 in milliseconds).
- Tokens per second (throughput).
- Errors per hour.
- Model memory footprint (MB).

13.2.3 Deployment Metrics

- Deployment duration (seconds).
- Success rate (%).
- Rollback count per bundle.
- Average time-to-rollback (seconds).

13.3 Minimal MVP Observability vs. Production

13.3.1 MVP (v0)

- Logs: stdout/stderr from device runtime, stored in local file, shipped to control plane.
- Metrics: Heartbeat (device_id, version, memory, CPU, status) every 60s.
- Dashboard: List devices; click to view latest logs; show heartbeat timeseries.
- Alerting: None (manual checks).

13.3.2 Production (v1+)

- Logs: Structured JSON logs; shipped to centralized log aggregator (ELK, Loki).
- Metrics: Prometheus format; scraped every 60s; stored in time-series DB.
- Tracing: Distributed traces for deployments (device → bundle download → extraction → reload).
- Dashboard: Grafana; multi-tenant; per-device and fleet-wide dashboards.
- Alerting: Alert on device offline, high error rate, deployment failure.
- Integrations: Send alerts to PagerDuty, Slack, etc.

13.4 Suggested Tooling

13.4.1 Logs (Pick One)

- **ELK Stack**: Elasticsearch + Logstash + Kibana (self-hosted, complex).
- **Loki**: Lightweight; designed for Kubernetes and edge (recommended for MVP+).
- **Datadog**: SaaS; fully managed; expensive at scale.

13.4.2 Metrics

- **Prometheus**: Scrape metrics; query with PromQL; widely adopted.
- **InfluxDB**: Time-series DB; good for high-cardinality metrics.
- **Datadog**: All-in-one; APM + logs + metrics.

13.4.3 Dashboards

- **Grafana**: Open-source; supports Prometheus, Loki, InfluxDB.
- **Datadog Dashboards**: If using Datadog.

Chapter 14

Tech Stack Recommendations

14.1 Two Viable Stacks

14.1.1 Stack A: Python-First (Recommended for Solo Founder)

Control Plane Backend

- **Framework**: FastAPI 0.100+
- **Async**: asyncio + uvicorn
- **Validation**: Pydantic v2
- **ORM**: SQLAlchemy (async)
- **Testing**: pytest + pytest-asyncio
- **Dev Tools**: Black (formatter), mypy (type checking), ruff (linter)

Control Plane Frontend

- **Framework**: Next.js 14+ (React SSR)
- **Styling**: Tailwind CSS
- **Components**: shadcn/ui
- **State**: TanStack Query (SWR + caching)
- **Charts**: Recharts or Plotly.js
- **Dev**: TypeScript, ESLint, Prettier

Database

- **Primary**: PostgreSQL 15+
- **Migrations**: Alembic (SQLAlchemy)
- **Hosting**: AWS RDS or Supabase (managed Postgres)

Object Storage

- **MVP**: MinIO (S3-compatible, runs locally or Docker)
- **Production**: AWS S3 or Backblaze B2
- **SDK**: boto3 (Python) or aws-sdk (JS)

Job Queue

- **MVP**: In-process asyncio.Queue
- **Pilot**: Redis (simple, reliable)
- **Production**: Redis or RabbitMQ
- **Worker**: Python asyncio loops or Celery

Device Runtime

- **Language**: Python 3.10+ (v0–v1), evolve to Rust (v2)
- **HTTP Client**: httpx (async)
- **Crypto**: cryptography library (RSA, SHA256)
- **Config**: pydantic-settings
- **Logging**: python-json-logger + loguru
- **Monitoring**: psutil (CPU/memory)
- **Packaging**: PyInstaller (single binary) or Debian .deb package

14.1.2 Stack B: TypeScript-First (Alternative)

Control Plane Backend

- **Framework**: Nest.js or Express
- **Runtime**: Node.js 20+
- **ORM**: Prisma (type-safe, auto-migrations)
- **Validation**: Zod or io-ts
- **Testing**: Jest
- **Dev Tools**: TypeScript, ESLint, Prettier

Frontend

Same as Stack A (Next.js).

Database & Storage

Same as Stack A (PostgreSQL + S3/MinIO).

Device Runtime

- **Language**: Python (due to llama.cpp bindings) or Go (compiled binary, fast).
- **Note**: TypeScript runtime is possible but overkill for edge; adds Node.js size overhead.

14.2 Recommended Starting Stack: Stack A (Python-First)

14.2.1 Rationale

1. **Cohesion**: Backend and runtime both Python; easier to share utilities (crypto, logging, config).
2. **Speed**: FastAPI is fast; Pydantic validation is simple.
3. **Solo Founder Friendly**: Fewer languages to manage; clear mental model.
4. **Ecosystem**: Rich ML/edge compute libraries in Python.
5. **Type Safety**: Pydantic + mypy provide type checking without verbosity of Java/Go.

14.3 Hosting Options

14.3.1 Development (MVP)

- **Laptop**: Local PostgreSQL + MinIO + Python dev server.
- **Docker Compose**: Single docker-compose.yml; spin up entire stack locally.

14.3.2 Pilot Stage

- **Control Plane API**: Fly.io or Render (easy deploy from Git; $0–$50/month).
- **Frontend**: Vercel (free for hobby; $0–$20/month).
- **Database**: AWS RDS (t3.micro free tier; then $20–$100/month) or Supabase ($0 tier, then $25/month).
- **Object Storage**: MinIO on same Fly/Render instance or AWS S3 ($0.023 per GB/month).

14.3.3 Production

- **Control Plane API**: AWS ECS (Fargate, $0.04 per vCPU/hour) or Kubernetes (self-hosted or managed EKS).
- **Frontend**: CloudFlare Pages or AWS CloudFront ($0.085 per GB).
- **Database**: AWS RDS (multi-AZ; $100–$500/month).
- **Object Storage**: AWS S3 (standard tier; $0.023 per GB/month).
- **CDN**: CloudFlare ($0.02 per GB).

14.4 Cost Estimates

14.4.1 MVP Stage (Weeks 1–10)

- Laptop + local dev: $0
- Domain (.com): $10/year
- Miscellaneous (Git, Slack free tier): $0
- **Total**: $1–$2/month

14.4.2 Pilot Stage (Weeks 11–32, 3 months)

- Fly.io + Render: $20–$50/month
- Postgres (AWS t3.micro + free tier): $0–$20/month
- S3 storage (assume 100GB bundles): $2–$5/month
- Datadog/monitoring: $0 (free tier)
- **Total**: $25–$75/month

14.4.3 Beta Stage (Weeks 33–52, 3 months, 50+ pilots)

- Fly.io (scaled): $100–$300/month
- Postgres (t3.small): $50–$100/month
- S3 storage (assume 500GB): $10–$20/month
- Datadog (if scaled): $100–$300/month
- **Total**: $260–$720/month

14.4.4 Production Stage (Months 9+, 100+ customers)

- API servers (3x ECS Fargate, t3.small): $300–$500/month
- Postgres (multi-AZ, db.r5.large): $500–$1000/month
- S3 storage (assume 1TB): $20–$50/month
- CloudFront CDN: $50–$200/month
- Datadog + monitoring: $300–$1000/month
- **Total**: $1170–$2750/month

Chapter 15

Repository & Directory Structure

15.1 Production-Grade Monorepo Layout

```
kernex/
+-- control-plane/
| +-- app/
| | +-- main.py (FastAPI entry point)
| | +-- api/
| | | +-- v1/
| | | | +-- devices.py
| | | | +-- bundles.py
| | | | +-- deployments.py
| | | | +-- logs.py
| | | +-- __init__.py
| | +-- models/ (Pydantic models)
| | | +-- device.py
| | | +-- bundle.py
| | | +-- deployment.py
| | +-- schemas/ (SQLAlchemy ORM)
| | | +-- device.py
| | | +-- bundle.py
| | | +-- deployment.py
| | +-- services/ (business logic)
| | | +-- deployment_service.py
| | | +-- bundle_service.py
| | | +-- device_service.py
| | +-- workers/ (background jobs)
| | | +-- deployment_worker.py
| | | +-- metrics_aggregator.py
| | | +-- cleanup_worker.py
```

```
| | +-- db/ (database)
| | | +-- session.py
| | | +-- migrations/ (Alembic)
| | +-- config.py (env vars, settings)
| | +-- auth.py (JWT, device auth)
| | +-- logging.py
| +-- tests/
| | +-- test_devices.py
| | +-- test_bundles.py
| | +-- test_deployments.py
| +-- requirements.txt (or pyproject.toml)
| +-- Dockerfile
| +-- README.md
|
+-- frontend/
| +-- app/
| | +-- page.tsx (home)
| | +-- layout.tsx
| | +-- devices/
| | | +-- page.tsx
| | | +-- [id]/
| | | +-- page.tsx
| | +-- bundles/
| | | +-- page.tsx
| | | +-- upload.tsx
| | +-- deployments/
| | +-- page.tsx
| | +-- [id]/
| | +-- page.tsx
| +-- components/
| | +-- DeviceList.tsx
| | +-- BundleUpload.tsx
| | +-- DeploymentForm.tsx
| | +-- Logs.tsx
| +-- hooks/ (custom React hooks)
| | +-- useDevices.ts
| | +-- useBundles.ts
| | +-- useDeployments.ts
| +-- lib/ (utilities)
| | +-- api.ts (API client)
| | +-- auth.ts (JWT handling)
```

```
| | +-- utils.ts
| +-- styles/ (Tailwind config)
| +-- package.json
| +-- tsconfig.json
| +-- next.config.js
| +-- Dockerfile
|
+-- runtime/
| +-- kernex/
| | +-- __main__.py (entry point)
| | +-- main.py (main loop)
| | +-- device/ (device management)
| | | +-- identity.py (RSA keys, device_id)
| | | +-- config.py (config loader)
| | | +-- info.py (hardware info)
| | +-- update/ (bundle updates)
| | | +-- manager.py (download, verify, swap)
| | | +-- integrity.py (SHA256, signatures)
| | | +-- atomic.py (atomic move operations)
| | +-- agent/ (agent process management)
| | | +-- launcher.py (spawn, signal handling)
| | | +-- monitor.py (health checks, restarts)
| | | +-- api.py (local HTTP server)
| | +-- polling/ (heartbeat + command polling)
| | | +-- client.py (main polling loop)
| | | +-- heartbeat.py (heartbeat payload)
| | | +-- commands.py (command processing)
| | +-- logging/ (local logging)
| | | +-- shipper.py (batch to control plane)
| | | +-- formatters.py (JSON format)
| | +-- config.py (env vars, settings)
| | +-- utils.py
| +-- tests/
| | +-- test_device.py
| | +-- test_update.py
| | +-- test_polling.py
| +-- requirements.txt
| +-- Dockerfile
| +-- systemd/
| | +-- kernex.service (systemd unit)
| +-- scripts/
```

```
| | +-- install.sh (setup script for RPi)
| | +-- uninstall.sh
| +-- README.md
|
+-- shared/
| +-- models.py (shared Pydantic models between control plane & runtime)
| +-- constants.py (API versions, timeouts)
| +-- utils.py (shared utilities)
|
+-- docs/
| +-- architecture.md
| +-- api-spec.md (OpenAPI)
| +-- bundle-spec.md
| +-- deployment-guide.md
| +-- troubleshooting.md
|
+-- infra/
| +-- docker-compose.yml (local dev environment)
| +-- kubernetes/ (later, production K8s manifests)
| | +-- api-deployment.yaml
| | +-- postgres-statefulset.yaml
| | +-- service.yaml
| +-- terraform/ (infrastructure as code, later)
| +-- main.tf
| +-- rds.tf
| +-- s3.tf
|
+-- scripts/
| +-- setup-dev.sh (initialize dev environment)
| +-- run-tests.sh (run full test suite)
| +-- build-runtime.sh (build device runtime binary)
| +-- deploy.sh (CI/CD integration)
|
+-- examples/
| +-- qwen-1.5b-bundle/ (example bundle)
| | +-- manifest.json
| | +-- config.json
| | +-- model.gguf
| +-- deployment-requests.json (example API calls)
| +-- device-bootstrap.sh (example device setup)
|
```

```
+-- .gitignore
+-- .env.example
+-- Makefile (common tasks: test, build, deploy)
+-- README.md (project overview)
+-- CHANGELOG.md
```

15.2 Configuration Management

15.2.1 Control Plane (.env.example)

```
# Control Plane Config
DATABASE_URL=postgresql://user:pass@localhost/kernex
REDIS_URL=redis://localhost:6379
S3_BUCKET=kernex-bundles
S3_REGION=us-east-1
AWS_ACCESS_KEY_ID=...
AWS_SECRET_ACCESS_KEY=...

# API Config
API_PORT=8000
API_HOST=0.0.0.0
LOG_LEVEL=info
JWT_SECRET=<random_secret>
JWT_EXPIRY_HOURS=24

# Control Plane URL (for devices)
CONTROL_PLANE_URL=https://api.kernex.io
CONTROL_PLANE_CERT_FINGERPRINT=<SHA256>
```

15.2.2 Device Runtime (.env.example)

```
# Device Runtime Config
DEVICE_ID=<generated_on_first_boot>
CONTROL_PLANE_URL=https://api.kernex.io
POLLING_INTERVAL=60
HEARTBEAT_TIMEOUT=5

# Paths
KERNEX_HOME=/var/kernex
AGENT_LOG_PATH=/var/kernex/logs/agent.log
```

```
10
11 # Agent Process
12 AGENT_TIMEOUT=30
13 MAX_RETRIES=3
14 ROLLBACK_ON_CRASHES=true
```

15.3 Secrets Management

- **MVP**: .env files (local development).
- **Pilot**: AWS Secrets Manager or HashiCorp Vault.
- **Production**: Managed secrets in deployment platform (AWS Secrets Manager, Heroku Config Vars, etc.).
- **Device Keys**: Encrypted at rest; never transmitted except public key at registration.

15.4 Code Standards

15.4.1 Python

- **Formatter**: Black (line length 100).
- **Linter**: ruff (include isort for imports).
- **Type Checker**: mypy (strict mode).
- **Docstrings**: Google style.
- **Pre-commit**: Add hooks for Black, ruff, mypy.

15.4.2 TypeScript

- **Formatter**: Prettier (line length 100).
- **Linter**: ESLint (recommended config).
- **Strict**: tsconfig.json with strict=true.
- **Pre-commit**: Add hooks for Prettier, ESLint.

15.5 Git Workflow

15.5.1 Branch Strategy

- **main**: Production-ready code; requires PR review.
- **develop**: Integration branch; feature branches merge here.
- **feature/...**: Feature branches; branched from develop.
- **fix/...**: Bugfix branches; branched from main (if hotfix) or develop.

15.5.2 Commit Convention

- **Format**: `<type>(<scope>): <message>`
- **Types**: feat, fix, docs, style, refactor, test, chore
- **Example**: `feat(device): implement offline command queueing`

Chapter 16

Development Workflow

16.1 Local Dev Setup (15 mins)

```
# Clone repo
git clone https://github.com/your-org/kernex.git
cd kernex

# Setup control plane
cd control-plane
python3.10 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
cp .env.example .env

# Setup frontend
cd ../frontend
npm install
cp .env.example .env.local

# Setup runtime
cd ../runtime
python3.10 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Run full stack locally
cd ..
docker-compose up -d

# Check health
```

```
curl http://localhost:8000/health
open http://localhost:3000
```

16.2 Docker Compose for Local Development

```
version: '3.8'

services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: kernex
      POSTGRES_USER: kernex_user
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7
    ports:
      - "6379:6379"

  minio:
    image: minio/minio
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    volumes:
      - minio_data:/data

  api:
    build:
      context: ./control-plane
      dockerfile: Dockerfile
    ports:
```

```
      - "8000:8000"
    environment:
      DATABASE_URL: postgresql://kernex_user:password@postgres:5432/kernex
      REDIS_URL: redis://redis:6379
      S3_ENDPOINT_URL: http://minio:9000
    depends_on:
      - postgres
      - redis
      - minio

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    environment:
      NEXT_PUBLIC_API_URL: http://localhost:8000

volumes:
  postgres_data:
  minio_data:
```

16.3 CI/CD Pipeline Design

16.3.1 GitHub Actions Workflow

```
name: Test & Build

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
        with:
          python-version: '3.10'
      - name: Install dependencies
```

```
      run: |
        pip install -r control-plane/requirements.txt
        pip install -r runtime/requirements.txt
    - name: Run tests
      run: |
        pytest control-plane/tests/
        pytest runtime/tests/
    - name: Lint (Black, ruff, mypy)
      run: |
        black --check control-plane runtime
        ruff check control-plane runtime
        mypy control-plane runtime

build:
  runs-on: ubuntu-latest
  needs: test
  if: github.ref == 'refs/heads/main'
  steps:
    - uses: actions/checkout@v2
    - name: Build API Docker image
      run: docker build -t kernex-api:latest ./control-plane
    - name: Build runtime binary
      run: bash scripts/build-runtime.sh
    - name: Push to registry
      run: |
        docker tag kernex-api:latest gcr.io/your-org/kernex-api:latest
        docker push gcr.io/your-org/kernex-api:latest
```

16.4 Testing Strategy

16.4.1 Unit Tests

- Test each function in isolation.
- Mock external dependencies (DB, S3, Redis).
- Target: 80%+ code coverage.

16.4.2 Integration Tests

- Spin up docker-compose; test API + DB + queue.

- Test deployment workflow end-to-end.
- Target: 3–5 critical workflows.

16.4.3 E2E Tests

- Deploy to test devices (Raspberry Pi + Jetson).
- Test enrollment, deploy, rollback, logs.
- Run daily; catch regressions early.

16.5 Release Process

16.5.1 Semantic Versioning

- **Major**: Breaking API/runtime changes.
- **Minor**: New features (backward-compatible).
- **Patch**: Bugfixes.

16.5.2 Release Steps

1. Tag commit on main: `git tag v0.2.0`
2. GitHub Actions builds + pushes images.
3. Create GitHub Release with notes.
4. Deploy to staging; run E2E tests.
5. Deploy to production.

16.6 Full Local Demo

```
# Terminal 1: Start stack
docker-compose up

# Terminal 2: Create device 1
curl -X POST http://localhost:8000/api/v1/devices/register \
  -H "Content-Type: application/json" \
  -d '{
    "public_key": "-----BEGIN RSA PUBLIC KEY-----\n...",
```

```
␣␣␣␣"device_type":␣"raspberry_pi"
␣␣}'
# Returns: { "device_id": "dev-001", "registration_token": "..." }

# Terminal 3: Start device runtime (simulated)
cd runtime
python -m kernex --device-id dev-001 --control-plane-url http://localhost:8000

# Terminal 1: Upload bundle
curl -X POST http://localhost:8000/api/v1/bundles \
  -F "file=@example-bundle.tar.gz" \
  -F "manifest=@manifest.json"
# Returns: { "bundle_id": "bundle-001", "version": "v0.2", "checksum": "abc123
    ..." }

# Terminal 1: Deploy
curl -X POST http://localhost:8000/api/v1/deployments \
  -H "Content-Type:␣application/json" \
  -d '{
␣␣␣␣"bundle_version":␣"v0.2",
␣␣␣␣"target_devices":␣["dev-001"]
␣␣}'
# Returns: { "deployment_id": "deploy-001", "status": "pending" }

# Watch device runtime: output shows "Deployment received", "Downloading bundle
    ", "Verifying", "Swapping", "Agent reloaded"

# Terminal 1: View logs via API
curl http://localhost:8000/api/v1/devices/dev-001/logs?limit=10

# Terminal 1: Rollback
curl -X POST http://localhost:8000/api/v1/deployments/deploy-001/rollback \
  -H "Content-Type:␣application/json" \
  -d '{␣"target_version":␣"v0.1"␣}'

# Watch device runtime: output shows "Rollback received", "Reverting", "Agent
    reloaded"
```

Chapter 17

MVP Build Plan (8 Weeks)

17.1 Overview

Solo founder, working 40–60 hrs/week (school + startup). Weekly deliverables are concrete; stop-conditions are explicit.

17.2 Week-by-Week Sprint

17.2.1 Week 1: Discovery & Foundation

Goal: Validate wedge; lock in first 3 pilot integrators.

- **Monday**: Finalize wedge (retail? logistics? industrial?). Document 1-pager.
- **Tuesday**: Cold outreach: 15 emails to integrators; target 2–5 responses.
- **Wednesday**: Conduct 2 customer interviews (30 mins each); validate pain.
- **Thursday**: Set up development environment (Git repo, venv, Docker).
- **Friday**: Sketch DB schema on paper; draft API endpoints list.
- **Success Metric**: 3 integrators say "yes, we'd pilot this"; schema approved by mentor.

17.2.2 Week 2: Backend Foundation

Goal: FastAPI skeleton + PostgreSQL basics working.

- **Monday**: Implement Device model + Devices table; POST /register-device endpoint.
- **Tuesday**: Add Bundle model + upload endpoint (store in local /tmp).

- **Wednesday**: Add Deployment model + deploy endpoint (no logic yet).
- **Thursday**: Add device authentication (RSA signature verification).
- **Friday**: Integration test: device registers + receives device_id.
- **Success Metric**: `curl -X POST http://localhost:8000/api/v1/devices/register` works; device gets device_id.

17.2.3 Week 3: Device Runtime Skeleton

Goal: Python runtime can register with control plane + poll.

- **Monday**: Device generates RSA keypair; stores in /var/kernex/.
- **Tuesday**: Polling client: every 60s, POST heartbeat to control plane.
- **Wednesday**: Heartbeat payload: device_id, agent_version, memory, CPU, status.
- **Thursday**: Add heartbeat endpoint to control plane; store in DB.
- **Friday**: End-to-end: device boots, registers, starts polling.
- **Success Metric**: Control plane logs show heartbeats arriving every 60s from simulated device.

17.2.4 Week 4: Bundle Upload & Versioning

Goal: Upload bundle; control plane stores + checksums.

- **Monday**: Web UI skeleton (Next.js); login page (dummy auth).
- **Tuesday**: Bundle upload page (HTML form).
- **Wednesday**: Upload endpoint: accept tarball, compute SHA256, store in S3/MinIO.
- **Thursday**: Bundle versioning: assign v0.1, v0.2, etc.
- **Friday**: List bundles endpoint; show in UI.
- **Success Metric**: Upload bundle via web UI; see it listed with version + checksum.

17.2.5 Week 5: Deploy Logic

Goal: Deploy bundle to device; device downloads + verifies.

- **Monday**: Deploy endpoint: accept bundle_version + device_id; create Deployment record.
- **Tuesday**: Device polling: check for pending deployments; download bundle.
- **Wednesday**: Bundle verification: device computes SHA256; compares with control plane.
- **Thursday**: Atomic bundle swap: staging → current (no actual agent process yet).
- **Friday**: End-to-end test: deploy v0.1 to device; verify files in /var/kernex/.
- **Success Metric**: Device successfully downloads, verifies, and swaps bundle.

17.2.6 Week 6: Rollback

Goal: Rollback to previous bundle in < 30 seconds.

- **Monday**: Keep previous bundle in /var/kernex/agent_bundle_previous/.
- **Tuesday**: Rollback endpoint: accept deployment_id + target_version.
- **Wednesday**: Device receives rollback command; swaps: previous → current.
- **Thursday**: Web UI: deployment detail page; rollback button.
- **Friday**: Integration test: deploy v0.1 → v0.2 → rollback to v0.1; measure time.
- **Success Metric**: Rollback completes in < 30 seconds; bundle verified correct.

17.2.7 Week 7: Logs & UI

Goal: View device logs in control plane dashboard.

- **Monday**: Device writes logs to /var/kernex/agent.log (JSON format).
- **Tuesday**: Log shipper: batch logs; POST to control plane every 60s.
- **Wednesday**: Control plane stores logs in Logs table.
- **Thursday**: Web UI: device detail page; show last 100 logs.
- **Friday**: Tail logs endpoint; stream logs in real-time (optional).
- **Success Metric**: Deploy bundle; see logs in UI within 60s of emission.

17.2.8 Week 8: Polish & First Pilot

Goal: MVP ready; deploy to first integrator pilot.

- **Monday**: End-to-end testing: deploy v1 → v2 → rollback 10 times; all succeed.
- **Tuesday**: Documentation: setup guide for device runtime; API spec.
- **Wednesday**: Demo script: 5-minute walk-through (upload bundle, deploy, rollback).
- **Thursday**: Record demo video; publish to GitHub.
- **Friday**: Deploy to test Raspberry Pi + Jetson; verify on real hardware.
- **Success Metric**: First pilot integrator successfully deploys bundle; provides feedback.

17.3 Stop Conditions (Pivot if Hit)

- **Week 2**: API skeleton doesn't compile or tests fail → rethink tech stack.
- **Week 4**: Integrators say "Ansible works fine for us" → pivot wedge or GTM.
- **Week 5**: Device registration flaky → simplify authentication.
- **Week 7**: Logs arriving 5+ minutes late → optimize shipper.

Chapter 18

Go-To-Market Plan

18.1 First 100 Users Strategy

18.1.1 Target: System Integrators & Agencies

Why? Low sales cycles, viral distribution, technical support-ready.

18.2 Distribution Loop 1: Cold Outreach

18.2.1 Email Template

```
Subject: Deploying edge AI to 50+ devices?

Hi [First Name],

I noticed [Company Name] has built several edge AI
projects (retail surveillance, industrial inspection, etc.).

I'm building Kernex---a control plane to deploy and rollback
AI models on edge devices. No SSH scripts. No field visits.
Instant rollback.

Current problem: Updating models to 50 devices takes 6 hours.
Rolling back takes 2 hours. Kernex does both in minutes.

Do you manage 30+ devices? Free 15-min call to see if it fits?

[Your name]
kernex.io
GitHub: github.com/your-handle
```

18.2.2 Outreach Volume

- **Week 1**: 15 emails (research integrators in retail, logistics, mfg).
- **Weeks 2–4**: 20 emails/week; cold approach; expect 2–5% reply.
- **Goal**: 5–7 calendar calls scheduled.

18.3 Distribution Loop 2: Community

18.3.1 Jetson Community Forum

- Answer 2–3 questions/week on model deployment.
- Mention: "We're building a tool for this; free beta available."

18.3.2 Edge AI Slack / Reddit

- r/MachineLearning, r/robotics, r/embedded: "Show HN: Kernex" post with demo video.
- Engage in threads; provide value (don't just promote).

18.3.3 Twitter / LinkedIn

- 2–3 posts/week (tips: "How to safely rollback models on 100 devices", case studies).
- Tag relevant accounts (NVIDIA, Jetson, edge AI influencers).

18.4 Distribution Loop 3: Powered by Kernex Runtime

When integrators deploy Kernex-managed solutions, their customer dashboard shows: "Powered by Kernex Runtime (learn more link)."

Customers curious → click link → land on Kernex marketing site → signup as operator account → Kernex converts.

18.5 Distribution Loop 4: Content

18.5.1 Blog Posts (Medium, Dev.to)

- **Post 1**: "Why 70% of Edge AI Pilots Fail—And How to Fix It" (problem narrative; hint at solution)

- **Post 2**: "Deploying GGUF Models to 100 Devices: Step-by-Step Guide" (genuine value; mention Kernex at end)
- **Post 3**: "The Complete Guide to Model Rollback on Edge" (technical deep dive; credibility builder)

18.5.2 Video Demo

- 2 minutes: "From model to 100 devices in 5 minutes"
- Upload to YouTube; embed on homepage + blog.

18.6 Conversion Path

```
Integrator sees Kernex (cold email / community / content)
  v
Clicks link; lands on kernex.io homepage
  v
Reads: problem statement + demo video
  v
Signs up for free trial (5 devices, 30 days)
  v
Onboarding email sequence (Day 1: setup guide; Day 3: video call offer; Day 7:
    check-in)
  v
Integrator deploys to test devices
  v
Integrator tries rollback; impressed
  v
After 30 days: Offer upgrade (unlock 100+ devices, group deployments)
  v
Convert to paid (\$15/device/month)
```

Chapter 19

Business Model (Later)

19.1 Pricing Hypothesis

19.1.1 Option A: Per-Device/Month (Recommended)

- **Free Tier**: 5 devices, 30-day trial, email support only.
- **Standard ($15/device/month)**: up to 1000 devices; 24h support response; basic analytics.
- **Pro ($25/device/month)**: unlimited devices; 4h support; policy engine; canary automation.
- **Enterprise (custom)**: 1000+ devices; dedicated account manager; self-host option; SLA (99.9% uptime).

19.1.2 Unit Economics

- **Typical Customer**: 50 devices; Standard tier; 1 year annual commitment (20% discount).
- **ACV**: $50 \times \$15 \times 12 \times 0.8 = \$7,200$/year
- **MRR**: \$600/month
- **COGS**: \$100/month (hosting + observability)
- **Gross Margin**: 83%
- **Expansion Potential**: Same customer 50 → 150 devices (Year 2); 3× NRR.

19.2 Free Trial to Paid Conversion

- **Week 1–2**: Integrator tries pilot; gets comfortable.
- **Week 3**: After first successful deployment + rollback; sales outreach: "Want to unlock 100+ devices?"
- **Week 4**: Offer: 3 months free for annual commitment.
- **Target Conversion**: 30–50% of pilots convert within 60 days.

19.3 Monetization Timeline

- **Weeks 1–20 (MVP–Pilot)**: Free; no charge.
- **Weeks 21–32 (Beta)**: Free trials; 30-day limit; collect email + usage data.
- **Weeks 33–40 (Monetization)**: Announce pricing; convert pilots; hit $5k MRR target.
- **Months 9+ (Growth)**: Expand pricing tiers; enterprise sales; hit $20k+ MRR.

Chapter 20

Team Plan

20.1 Starting Solo: Responsibilities Checklist

- ✓Technical: backend API, frontend UI, device runtime, DevOps, security.
- ✓Customer: cold outreach, pilot calls, onboarding, feedback collection.
- ✓Marketing: blog, Twitter, community presence, demo script.
- ✓Admin: GitHub issues, roadmap, documentation, commit reviews (self).

Time Allocation (MVP stage): 80% coding, 10% customer calls, 10% marketing.

20.2 Hiring Roadmap

20.2.1 After MVP (Week 10)

Hire: Contract Support Engineer (5–10 hrs/week)

- Respond to pilot customer support.
- Run onboarding calls.
- Gather feature feedback.
- Cost: $500–$1,000/month

20.2.2 After Phase 3 Pilots (Month 8)

Hire 1: Full-time Systems Engineer

- Harden device runtime (later: Rust rewrite).
- Scalability + reliability (99.9% SLA).

- Infrastructure + DevOps.
- Required: 3+ years backend systems; comfortable with edge/embedded.
- Cost: $120k–$150k/year

Hire 2: Full-time Full-Stack Engineer

- Frontend UI (dashboards, UX).
- Backend scaling (API performance, DB optimization).
- Required: 2+ years web full-stack; PostgreSQL + React + Python/Node.
- Cost: $100k–$130k/year

20.2.3 After PMF (Month 12)

Hire: Head of Sales / Partnerships

- Enterprise deals (contract negotiation).
- Channel partnerships (system integrators as resellers).
- Customer success (reduce churn, upsell).
- Required: B2B SaaS sales; technical buyers.
- Cost: $80k–$120k base + commission.

20.3 Operational Structure (No Legal Advice)

20.3.1 Company Formation

- **Entity**: Delaware LLC (cheapest, flexible).
- **Cost**: $200 filing fee + annual compliance.
- **Reason**: Separates personal from business; enables equity grants; easier fundraising.

20.3.2 Banking & Accounting

- Open business bank account (Mercury, Wise, or local bank).
- Keep personal + business finances separate.
- File annual taxes (1099-NEC for contractors; W-2 for yourself).
- Use startup-friendly accountant ($100–$300/month).

20.3.3 Visa Considerations (International Student)

- Consult your SJSU international student advisor.
- Possibility: Founder as self-employed (LLC); does not trigger full-time employment rules for F-1 visa.
- Alternative: If founder status is restricted, bring on co-founder who is unrestricted.
- Reality: Many student founders operate this way; stay compliant and document.

20.3.4 Equity & Co-Founders

- Early engineers: Offer equity (vesting 4 years, 1-year cliff).
- Standard split: Founder 50%, Eng 25%, Eng 25% (adjust timing of joins).
- Track with Carta or Pulley (free for early-stage).

Chapter 21

Funding Plan

21.1 Bootstrap Path (MVP–Pilot)

- **Phase 1 (MVP, 2.5 months)**: $0–$500 (personal laptop, free tier AWS).
- **Phase 2 (Pilot, 2.5 months)**: $2k–$5k (Fly.io, Postgres, S3).
- **Total**: <$5k personal investment.

21.1.1 When to Fundraise

- **Pre-seed trigger**: MVP working; 3–5 integrator pilots engaged; first 1–2 paying customers.
- **Target**: $25k–$100k from angels (5–10 checks of $5k–$10k).
- **Use**: Hire contractor support; scale infrastructure; marketing.

21.2 Pre-Seed Funding Path

- **Seed Round Goal**: $250k–$500k (after $10k+ MRR + 10+ paying customers).
- **Valuation**: $2M–$5M post-money.
- **Use**: Hire 2 full-time engineers; enterprise sales push; product expansion.

21.3 Milestone-Based Funding

Phase	**Milestone**	**Funding Stage**	**Amount**
MVP	2 devices working; rollback proven	None	\$0
Pilot	10 pilots; 5 paying; \$5k MRR	Pre-Seed	\$25k–\$100k
Beta	50+ users; 10 paying; NPS $>$ 40	Seed	\$250k–\$500k
Growth	100+ customers; \$50k MRR; NRR $>$ 110%	Series A	\$1M–\$5M

Chapter 22

Risks & Mitigations

22.1 Technical Risks

22.1.1 Device Fragmentation

Risk Runtime crashes on Jetson but works on RPi; must support all.

Mitigation Build test lab with 3 devices. Test every code change on all 3. Document device-specific quirks.

Early Warning Pilot reports "works on PC, crashes on customer Jetson."

22.1.2 Bandwidth Failures

Risk Device on 4G network; 2GB bundle upload times out or fails mid-transfer.

Mitigation Implement resumable HTTP downloads. Test on throttled WiFi (1Mbps). Add bandwidth scheduling (v2).

Early Warning Pilot times-out downloading bundles; requests workarounds.

22.1.3 Security Vulnerability

Risk Attacker forges bundle signature; injects malicious model.

Mitigation Use RSA 4096 + SHA256 (v0). Device pins CA cert (no MITM). Security audit in v2.

Early Warning Any suspicious device behavior; investigate immediately.

22.2 Market Risks

22.2.1 Integrators Don't Adopt

Risk Phase 1 MVP done; integrators uninterested; "Ansible works fine."

Mitigation Validate wedge hard in Phase 0. Co-build with 1 integrator. Publish case studies.

Early Warning By Week 6, if integrator says "not interested," pivot wedge or GTM.

22.2.2 Enterprise Competitor Launches

Risk AWS IoT or Google launches "agent management" feature; copies you.

Mitigation Ship fast. Build moat (delta updates, agent telemetry, ecosystem). Sell to integrators (less direct competition).

Early Warning Google announces "Vertex AI at Edge"; diversify revenue.

22.2.3 Pilots Don't Convert

Risk Phase 3: 50 free users; only 2 convert to paid; metrics weak.

Mitigation During pilots, sell value (RBAC, SLA). Segment users by usage; target high-usage for upgrade. Offer annual discount.

Early Warning Month 6 of free tier; $<10\%$ conversion; rethink pricing or product.

22.3 Founder Risks

22.3.1 Burnout

Risk Solo founder; balancing school + startup; exams, coursework, stress.

Mitigation Part-time (10–15 hrs/week) in semester; light course load or semester off. Set boundaries (no coding past midnight; weekends off). Hire support engineer in Phase 2.

Early Warning Missing deadlines; quality slipping; motivation declining; talk to mentor.

22.3.2 Visa Constraints

Risk F-1 visa restrictions limit work hours or company structure.

Mitigation Consult SJSU international student advisor. Consider LLC structure (check if OK). Hire co-founder if needed.

Early Warning Visa renewal coming; ask before taking action.

22.3.3 Competition

Risk Idea is copyable; well-funded competitor builds faster.

Mitigation Speed (first-mover advantage). Build integrator network (switching cost high). Ship moat features (delta updates, agent telemetry, ecosystem).

Early Warning Competitor launches; analyze (better?); decide pivot vs double-down.

Chapter 23

Moat & Long-Term Vision

23.1 Defensibility: Multi-Layer Moat

23.1.1 Layer 1: Agent-Aware Observability

- Monitor inference latency, token throughput, error rates—not just device CPU/memory.
- Hard to copy: requires deep LLM + edge + ops knowledge.
- Defensibility: 12–18 months.

23.1.2 Layer 2: Delta Updates

- Only send changed model weights (50MB vs 2GB).
- Complex: merkle trees, binary diffing, resumable transfers.
- Defensibility: 18–24 months.

23.1.3 Layer 3: Integrator Ecosystem

- "Powered by Kernex Runtime" distribution loop.
- First-mover wins; integrators ship with you → hard to displace.
- Defensibility: 24+ months.

23.1.4 Layer 4: Trust & Compliance

- SOC 2 certified; tamper-evident logs; enterprise compliance.
- Requires investment ($50k–$100k audit); 6–12 months to achieve.

- Defensibility: 18+ months.

23.1.5 Layer 5: Integrations Marketplace

- Pre-built connectors to Datadog, PagerDuty, Splunk.
- Network effect: more integrations → more valuable → more integrations.
- Defensibility: 24+ months.

23.2 5–10 Year Vision

23.2.1 Year 1: Product-Market Fit

- 50–100 integrator pilots.
- $10k–$20k MRR.
- Kernex recognized in edge AI communities.

23.2.2 Years 2–3: Scale

- 500+ devices across 50 customers.
- $100k–$250k MRR.
- Self-hosted option launched.
- Enterprise features (SSO, policy engine, advanced analytics).
- Team: 10–15 people.

23.2.3 Years 4–5: Enterprise

- 10k+ devices managed globally.
- $500k–$1M+ MRR.
- ISO 27001 + SOC 2 certified.
- Customer base: 60% integrators, 40% operators.
- Category ownership: “The VMware of Edge AI.”

23.2.4 Years 5–10: Platform Expansion

- Kernex becomes standard runtime for distributed agents (not just LLMs).
- 50+ ISVs building on Kernex.
- International expansion (EMEA, APAC).
- Adjacent products: Kernex Analytics, Policy Engine.
- Potential exit: Acquisition by cloud provider or IPO.

Chapter 24

Learning Path for Gourav Mukherjee

24.1 Required Concepts

24.1.1 System Design

- State machines (device state, deployment state).
- Eventual consistency (logs from multiple devices, out-of-order arrival).
- Distributed systems basics (atomicity, idempotency, retry semantics).

24.1.2 Security

- RSA cryptography (keypair generation, signing, verification).
- TLS/HTTPS (certificate pinning, MITM prevention).
- OAuth2 / JWT (token-based auth).
- Least privilege (RBAC, multi-tenancy).

24.1.3 Operations

- HTTP (polling, idempotency, resumable downloads with Range headers).
- REST API design (versioning, error codes, pagination).
- Database design (schema, migrations, indexes, query optimization).
- Observability (logging, metrics, tracing).

24.1.4 Infrastructure

- Docker (containerization, images, volumes).
- Postgres (basics, ACID, JSONB, migrations).
- S3 / object storage (object model, lifecycle policies).
- Redis (in-memory store, queueing).

24.1.5 DevOps

- Git workflows (branching, PRs, commit conventions).
- CI/CD (GitHub Actions, automated testing).
- Secrets management (env vars, vaults).
- Monitoring (health checks, alerting).

24.2 2-Week Crash Course

24.2.1 Days 1–2: HTTP & REST Design

- Study REST principles (resource-oriented, stateless, HATEOAS).
- Learn HTTP methods (GET, POST, PATCH, DELETE), status codes, idempotency.
- Implement 3 simple FastAPI endpoints; test with curl.
- Read: RESTful Web Services by Leonard Richardson (Chapters 1–3).

24.2.2 Days 3–4: PostgreSQL Basics

- Learn relational data modeling (entities, relationships, normalization).
- Create Devices, Bundles, Deployments tables.
- Learn SQL (SELECT, INSERT, UPDATE, JOIN, indexes).
- Learn SQLAlchemy ORM (models, sessions, queries).
- Practice: build schema migrations with Alembic.

24.2.3 Days 5–6: OAuth2 / JWT

- Understand OAuth2 flow (authorization, token exchange).
- Learn JWT structure (header, payload, signature).
- Implement JWT-based auth in FastAPI.
- Test: login endpoint + protected endpoint.

24.2.4 Days 7–8: Cryptography (RSA, SHA256)

- Understand RSA (public/private keypairs, sign/verify).
- Understand SHA256 (hashing, collision resistance).
- Implement: generate keypair, sign payload, verify signature.
- Use Python cryptography library.
- Practice: device registration flow with key exchange.

24.2.5 Days 9–10: Secure OTA Updates

- Study: update patterns (verification, rollback, atomicity).
- Read: "Lessons Learned in Software Update Systems for Autonomous Vehicles" (USENIX).
- Implement: atomic file move (staging → current).
- Test: handle failure scenarios (download interrupted, checksum mismatch).

24.2.6 Days 11–12: Async Python

- Learn asyncio basics (coroutines, await, async for).
- Learn aiohttp (async HTTP client).
- Implement device polling loop (async).
- Practice: concurrent polling from 5 devices.

24.2.7 Days 13–14: Docker & Deployment

- Write Dockerfile for control plane API.
- Write docker-compose.yml (API + Postgres + Redis).
- Deploy locally; verify all services healthy.
- Deploy to Fly.io or Render; verify health checks.

24.3 Reading List

- **HTTP**: "HTTP/2 Explained" (by Daniel Stenberg).
- **REST**: "RESTful Web Services" (by Leonard Richardson, Sam Ruby).
- **Databases**: "Designing Data-Intensive Applications" (by Martin Kleppmann, Chapters 1–5).
- **Security**: "The Web Application Hacker's Handbook" (Stuttard, Pinto, Chapters 1–3).
- **DevOps**: "The Phoenix Project" (by Gene Kim, Kevin Behr, George Spafford).
- **Distributed Systems**: "Designing Distributed Systems" (by Brendan Burns).

Chapter 25

Simplified "Vibe Coding" Build Guide

25.1 Core Mental Model

Think of Kernex as "four truth systems" that must stay in sync:

1. **Device Identity**: Who is this device? (device_id, public key, hardware info)
2. **Release State**: What code should run? (which bundle version is "desired" vs "actual")
3. **Bundle Integrity**: Is code authentic? (checksums, signatures)
4. **Logs/Heartbeat**: Is it healthy? (does device report success/failure?)

If these four stay consistent, Kernex works. If they diverge, debug one at a time.

25.2 State Machines in Plain English

25.2.1 Device State Machine

```
Device boots
  $\rightarrow$ (sends heartbeat every 60s)
  $\rightarrow$ Control plane receives it
  $\rightarrow$ Control plane marks device as "online"

If device goes 30+ mins without heartbeat
  $\rightarrow$ Control plane marks device as "offline"

Device reconnects
  $\rightarrow$ Sends heartbeat
  $\rightarrow$ Control plane marks device as "online"
```

25.2.2 Deployment State Machine

```
User clicks "Deploy v2 to device-001"
  $\rightarrow$ Deployment created in DB (status = "pending")

Device polls and sees new deployment
  $\rightarrow$ (status changes to "in_progress")

Device downloads v2, verifies checksum
  $\rightarrow$ Device swaps: v1 -> previous, v2 -> current
  $\rightarrow$ Device reports success

Control plane receives success
  $\rightarrow$ Deployment status = "success"

If user clicks rollback
  $\rightarrow$ Device swaps: previous -> current
  $\rightarrow$ Deployment status = "rolled_back"
```

25.3 Vertical-Slice Approach (Suggested Build Order)

Don't build all backends, then all frontends, then runtimes.

Instead, build **one vertical slice at a time**:

25.3.1 Slice 1: Device Registration

What to build:

- Device: Generate RSA keypair locally.
- Device: POST /register endpoint with public key.
- Control Plane: Accept POST; save device; return device_id.
- Device: Store device_id locally.

How to test:

```
curl -X POST http://localhost:8000/api/v1/devices/register \
  -d '{...}'
# See device appears in Postgres
```

25.3.2 Slice 2: Heartbeat Loop

What to build:

- Device: Every 60s, POST heartbeat (device_id, status, memory, CPU).
- Control Plane: Accept heartbeat; update last_heartbeat timestamp in DB.
- Web UI: Simple page listing devices + last heartbeat time.

How to test:

```
Watch device logs; see "Heartbeat sent" every 60s
Visit web UI; see device appear; see timestamp update every 60s
```

25.3.3 Slice 3: Bundle Upload

What to build:

- Web UI: File upload form (accept tarball + manifest.json).
- Control Plane: POST /bundles endpoint; save to S3; compute SHA256.
- Web UI: List all bundles; show version + checksum + size.

How to test:

```
Upload bundle.tar.gz via web UI
curl http://localhost:8000/api/v1/bundles
# See bundle appear with correct checksum
```

25.3.4 Slice 4: Deploy

What to build:

- Web UI: Dropdown to select bundle version + select device + click Deploy.
- Control Plane: POST /deployments; create Deployment record; add to queue.
- Device: Check for pending deployments during heartbeat.
- Device: Download bundle from control plane; verify SHA256.
- Device: Atomic swap (staging → current).
- Device: POST deployment result back to control plane.

How to test:

```
Deploy v1 via web UI
Watch device logs: "Deployment␣received", "Downloading", "Verifying",
"Swapping", "Success"
curl http://localhost:8000/api/v1/deployments/{id}
# See status = "success"
```

25.3.5 Slice 5: Rollback

What to build:

- Device: Keep previous bundle in /var/kernex/agent_bundle_previous/.
- Web UI: Deployment detail page; show rollback button.
- Control Plane: POST /deployments/id/rollback endpoint.
- Device: Receive rollback command; atomic swap (previous → current).

How to test:

```
Deploy v1 -> v2 -> rollback to v1
Measure time: should be < 30 seconds
Verify v1 running correctly
```

25.4 MVP Demo Checklist

- ✓Device A + B register (see device_ids in DB)
- ✓Upload bundle v0.1 (see checksum computed)
- ✓Deploy v0.1 to both devices (see in$_{p}rogress$→ success)
- ✓Devices report healthy heartbeat (v0.1 running)
- ✓Upload bundle v0.2
- ✓Deploy v0.2 to both devices (see download, verify, swap)
- ✓Devices report healthy heartbeat (v0.2 running)
- ✓Click rollback; revert to v0.1 ($<$ 30 secs)
- ✓Devices report healthy heartbeat (v0.1 running again)
- ✓View logs from both devices in web UI

25.5 What NOT to Vibe-Code Blindly

These need careful thought; don't just copy LLM output:

- **Atomic swap logic**: If you move current → previous, then move staging → current, but the second move fails, you've lost the previous bundle. Use proper file system operations (rename is atomic; don't use mv + rm).
- **RSA signature verification**: Don't implement this yourself. Use established cryptography library. Signature verification has many pitfalls (padding attacks, etc.).
- **Auth tokens**: Don't store passwords in plaintext. Use bcrypt for hashing. Use JWT libraries (don't roll your own).
- **Database migrations**: Use Alembic (SQLAlchemy migrations). Don't write raw SQL ALTER statements; they can cause data loss.
- **Retry logic**: If download fails, don't retry infinitely. Use exponential backoff. Cap at 60 seconds.

25.6 LLM Prompts for Safe Scaffolding

When asking LLMs (Claude, GPT, etc.) to generate boilerplate, give these constraints:

```
Prompt Template:
"Generate a Python FastAPI endpoint for [purpose].
Constraints:
- Use Pydantic for input validation
- Include type hints for all parameters and return types
- Add basic error handling (raise HTTPException)
- No hardcoded values; use env vars via settings.py
- Include a unit test using pytest
- Assume async function
- Do NOT implement authentication; assume it's middleware
Example use case: [concrete example]"
```

Why constraints matter: Broad prompts generate over-engineered code. Specific constraints yield scoped, safe scaffolding.

25.7 Example: Naive vs. Safe Device Registration

25.7.1 Naive Approach (DON'T DO THIS)

```
@app.post("/register")
def register_device(req):
    device_id = uuid.uuid4()
    db.execute(f"INSERT␣INTO␣devices␣VALUES␣('{device_id}',
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣'{req.public_key}')") # SQL INJECTION!
    return {"device_id": device_id}
```

25.7.2 Safe Approach (DO THIS)

```
from pydantic import BaseModel
from sqlalchemy import insert

class DeviceRegisterRequest(BaseModel):
    public_key: str
    device_type: str

@app.post("/api/v1/devices/register")
async def register_device(
    req: DeviceRegisterRequest,
    session: Session = Depends(get_session)
):
    try:
        device_id = uuid.uuid4()
        device = Device(
            id=device_id,
            device_id=str(device_id),
            public_key=req.public_key,
            device_type=req.device_type
        )
        session.add(device)
        session.commit()
        return {
            "device_id": str(device_id),
            "registration_token": generate_token()
        }
    except SQLAlchemyError as e:
        session.rollback()
        raise HTTPException(status_code=500, detail="DB␣error")
```

Key differences:

- Pydantic validation (automatic input validation).
- SQLAlchemy ORM (no SQL injection).
- Async function.
- Error handling.
- Type hints.

Appendix A

Glossary

Agent An LLM or ensemble of LLMs running locally on a device with access to tools (APIs, file access).

Agent Bundle Versioned package: model weights (GGUF), system prompt, tool definitions, config, manifest. Signed cryptographically.

Canary Deployment Rolling out to small subset (canary group) first; auto-promote if healthy.

Control Plane Cloud-hosted service managing device fleets, bundles, deployments, observability.

Delta Update Transmitting only changed weights (50MB vs 2GB).

Device Runtime Lightweight agent running on edge device; handles polling, updates, agent management.

GGUF Quantized model format (llama.cpp); efficient for local inference.

Heartbeat Periodic signal (60s) from device: status, memory, CPU, version.

OTA Over-the-air deployment; update without physical device access.

Rollback Revert to previous bundle version; atomic, < 30 seconds.

Tamper-Evident Log Append-only log where each entry hashes previous entry; cannot modify retroactively.

Appendix B

MVP Requirements Checklist

B.1 Control Plane

- ✓PostgreSQL schema (Devices, Bundles, Deployments, Heartbeats, Logs, Users, Organizations).
- ✓FastAPI app (main.py, config, async setup).
- ✓Device endpoints: POST /register, GET /devices, GET /devices/id, POST /heartbeat.
- ✓Bundle endpoints: POST /upload, GET /bundles, GET /bundles/id (streaming).
- ✓Deployment endpoints: POST /deployments, GET /deployments, POST /rollback.
- ✓Logs endpoints: POST /batch, GET /devices/id/logs.
- ✓Authentication: device signatures + user JWT.
- ✓Database ORM (SQLAlchemy), migrations (Alembic).
- ✓React UI (Next.js): dashboard, device list, bundle upload, deployment form, logs viewer.
- ✓Object storage: MinIO or local filesystem (MVP).
- ✓Job queue: in-process asyncio.Queue (MVP) or Redis (pilot).
- ✓Error handling, logging, tests (pytest).

B.2 Device Runtime

- ✓Device generates RSA keypair; stores private key locally.
- ✓Device registration: POST public key; receive device_id.
- ✓Polling client: every 60s, POST heartbeat; GET commands.
- ✓Bundle download: streaming download, resumable on failure.
- ✓Bundle verification: compute SHA256 locally; compare with control plane.
- ✓Atomic bundle swap: staging → current, previous backup.
- ✓Agent process management: launch, signal, restart on crash.
- ✓Log shipper: batch logs; POST to control plane.
- ✓Disk layout: /var/kernex/config, bundles, staging, logs.
- ✓Systemd unit file (optional: supervisor).
- ✓Installation script (setup script for RPi).
- ✓Tests (pytest).

Appendix C

Example Manifests & Configs

C.1 Bundle Manifest

```
{
  "version": "1.2.0",
  "runtime_version_min": "0.1.0",
  "model": {
    "name": "qwen-1.5b-instruct",
    "format": "gguf",
    "size_mb": 1024,
    "quantization": "Q4_K_M",
    "sha256": "abc123..."
  },
  "system_prompt": "You are a helpful assistant.",
  "tools": [
    {
      "name": "search",
      "description": "Search the web",
      "endpoint": "/api/search"
    }
  ],
  "env_vars": {
    "INFERENCE_TIMEOUT": "30",
    "MAX_TOKENS": "2048"
  },
  "metadata": {
    "created_date": "2025-12-27",
    "trained_on": ["retail", "surveillance"]
  }
}
```

C.2 Runtime Config

```
{
  "device_id": "device-12345",
  "control_plane_url": "https://api.kernex.io",
  "polling_interval": 60,
  "heartbeat_timeout": 5,
  "agent_timeout": 30,
  "max_retries": 3,
  "rollback_on_crashes": true,
  "log_level": "info"
}
```

C.3 Deployment JSON

```
{
  "id": "deploy-abc123",
  "org_id": "org-xyz",
  "bundle_version": "1.2.0",
  "target_devices": ["device-001", "device-002", "device-003"],
  "status": "in_progress",
  "created_at": "2025-12-27T10:00:00Z",
  "created_by_user_id": "user-123",
  "description": "Shoplifting model v1.2 with improved night detection"
}
```

Appendix D

Example Failure Scenarios

D.1 Scenario 1: Bundle Checksum Mismatch

```
Device downloads bundle v1.2
Device computes SHA256: "abc123"
Control plane checksum: "def456"
Mismatch!

Device action:
  1. Log error: "Checksum mismatch for v1.2"
  2. Delete staging directory
  3. POST deployment result: { status: "failed",
                               reason: "checksum_mismatch" }
  4. Agent continues running v1.1 (previous)

Control plane action:
  1. Marks Deployment as FAILED
  2. Marks Bundle v1.2 as "unstable"
  3. Alerts user: "Deployment to device-001 failed"
  4. User investigates bundle integrity
```

D.2 Scenario 2: Agent Crashes After Deploy

```
Device deploys v1.2 successfully
Device reloads agent with v1.2
Agent crashes (OOM, segfault, etc.)

Device action:
  1. systemd detects crash; auto-restarts agent
```

```
  2. Agent crashes again (crash count = 2)
  3. systemd restarts again (crash count = 3)
  4. After 3 crashes in 5 mins: systemd gives up
  5. Runtime detects systemd failure
  6. Runtime performs automatic rollback:
     - Move v1.1 (previous) $\rightarrow$ current
     - Restart agent with v1.1
  7. Agent starts successfully
  8. Runtime POSTs: { deployment_id, status: "failed",
                      reason: "auto_rollback_due_to_crashes" }

Control plane action:
  1. Marks Deployment as FAILED + ROLLED_BACK
  2. Marks Bundle v1.2 as "unstable"
  3. Device status returns to v1.1
  4. User sees: "Deployment failed; auto-reverted to v1.1"
```

D.3 Scenario 3: Device Offline for 2 Hours

```
Device on 4G network; network unstable
Device tries to heartbeat; request times out
Polling client logs: "Heartbeat failed; will retry"

Every 5 mins: retry heartbeat; fail silently

Control plane side:
  1. After 30 mins without heartbeat: mark device "offline"
  2. UI shows device as offline (gray badge)

User deploys v1.2 to device-001 (marked offline)
Deployment is queued; device gets "pending" status

Device network reconnects (2 hours later)
Device resumes polling successfully

Device receives: { commands: [deploy_v1.2] }
Device downloads, verifies, swaps
Device reports success

Control plane:
```

```
22  1. Marks deployment as SUCCESS
23  2. Marks device as ONLINE (green badge)
```

Appendix E

Next 7 Days Checklist

Goal: Start building Kernex. Validate wedge. Get first 3 pilots interested.

E.1 Day 1 (Friday, Dec 27)

- ✓Finalize wedge: which vertical? (retail, logistics, industrial, other?)
- ✓Document 1-pager: wedge definition + ICP + why.
- ✓Create GitHub repo: `kernex-control-plane`, `kernex-device-runtime`, README.
- ✓Set up dev environment: Python 3.10 venv, PostgreSQL local, Docker.
- ✓Schedule 15-min call with 1 mentor to reality-check wedge + timeline.

E.2 Day 2 (Saturday, Dec 28)

- ✓Research: identify 3 competitors (IoT hubs, K3s, DIY). 1-pager on gaps.
- ✓Sketch DB schema on paper: Devices, Bundles, Deployments, Heartbeats. Photo it.
- ✓Draft API spec: 12–15 endpoints (device register, bundle upload, deploy, rollback, heartbeat, logs).

E.3 Day 3 (Sunday, Dec 29)

- ✓Cold outreach: Send 10 personalized emails to integrators.
- ✓Track responses: spreadsheet with email sent date, response status.
- ✓Sketch landing page for kernex.io (Figma or paper sketch).

E.4 Day 4 (Monday, Dec 30)

- ✓Code: FastAPI skeleton (`main.py`, 3 endpoints stub).
- ✓Code: Pydantic models for Device, Bundle, Deployment.
- ✓Verify: `uvicorn app.main:app -reload` runs without errors.
- ✓Verify: `curl http://localhost:8000/health` returns 200.

E.5 Day 5 (Tuesday, Dec 31)

- ✓Database: PostgreSQL running; create kernex DB.
- ✓Code: SQLAlchemy models (Device, Bundle, Deployment tables).
- ✓Code: Alembic migration script (create tables).
- ✓Verify: `alembic upgrade head`; tables created.

E.6 Day 6 (Wednesday, Jan 1)

- ✓Device runtime: Python skeleton (`__main__.py`, imports).
- ✓Device: Generate RSA keypair; save to /var/kernex/private_key.
- ✓Device: Polling loop stub (logs "polling..." every 60s).
- ✓Verify: Device boots; runs for 5 mins; polls logged.

E.7 Day 7 (Thursday, Jan 2)

- ✓Summary: Write 1-page "Week 1 Progress Report" (what validated, learned, next steps).
- ✓Share with mentor for feedback.
- ✓Update GitHub README with current status, next week plan.
- ✓Plan Week 2: registration + heartbeat flow.

Success Criteria for Week 1:

- 3 integrators say “yes, we’d pilot this.”
- FastAPI + Postgres running locally.

- Device runtime boots without errors.
- GitHub repo public with README + architecture diagram.

KERNEX

Primary Architecture & Execution Documentation

Author: **Gourav Mukherjee**

Version: 1.0

Status: Living Document

This document represents the initial canonical specification for Kernex. It is expected to evolve as real-world pilots, failures, and operational feedback inform the system design.

Next Intended Actions

- Validate the MVP assumptions with real edge deployments
- Build and demo the two-device deploy + rollback flow
- Iterate only on features proven necessary by users
- Avoid premature optimization or feature expansion

Kernex is not a product of perfection, but of iteration under real constraints. Stability, operability, and trust are the north stars.

—

End of Document