Kamiwaza AI Platform

Offline Installation Guide

Version 0.7.0

Table of Contents

Administrator Guide

1. Authentication & Access Control

Kamiwaza provides enterprise-grade authentication built on **Keycloak** with OpenID Connect (OIDC) and JWT token validation.

1.1 Authentication Architecture

```
User → Keycloak (IdP) → JWT Token → Traefik → ForwardAuth → API Services
                                        ↓
                                  [Validated] → Access Granted
                                        ↓
                                  [Rejected] → 401/403 Error
```

Components:

- **Keycloak**: Identity provider managing users, authentication, and token issuance
- **ForwardAuth Service**: Validates JWT tokens and enforces access policies
- **Traefik**: Reverse proxy routing requests through ForwardAuth middleware
- **RBAC Policy Engine**: YAML-based endpoint access control

1.2 Authentication Modes

Kamiwaza supports two operational modes:

Mode	Use Case	Configuration
With Authentication	Production, staging, secure environments	`KAMIWAZA_USE_AUTH=true`
Bypass Mode	Local development, debugging	`KAMIWAZA_USE_AUTH=false`

To enable authentication:

```
# In env.sh or environment
export KAMIWAZA_USE_AUTH=true
bash startup/kamiwazad.sh restart
```

⚠ **Warning:** Bypass mode (`KAMIWAZA_USE_AUTH=false`) disables all authentication. Use only in secure development environments.

1.3 Token-Based Authentication

Kamiwaza uses **RS256 JWT tokens** with asymmetric cryptographic signatures.

Token Lifecycle:

1. **Acquisition**: User authenticates with Keycloak via username/password or SSO
2. **Validation**: ForwardAuth validates token signature against JWKS endpoint
3. **Authorization**: User roles checked against RBAC policy
4. **Expiration**: Access tokens expire (default: 1 hour), require refresh
5. **Revocation**: Logout invalidates tokens

Token Delivery Methods:

- HTTP `Authorization: Bearer <token>` header (recommended for APIs)
- Secure HTTP-only cookie (automatic for browser sessions)

---

2. User Management

2.1 Accessing Keycloak Admin Console

Default Credentials (change immediately in production):

- **URL:** http://localhost:8080 (http://localhost:8080) (or your configured Keycloak URL)
- **Username:** `admin`
- **Password:** Set via `KEYCLOAK_ADMIN_PASSWORD` environment variable

Production Setup:

```
# Set secure admin password in env.sh
export KEYCLOAK_ADMIN_PASSWORD="<strong-random-password>"
```

2.2 Creating User Accounts

Via Keycloak Admin Console:

1. Navigate to **Users** in left sidebar
2. Click **Add User**
3. Fill in required fields:
   - **Username** (required)
   - **Email** (required for password reset)
   - **First Name / Last Name** (optional)
4. Toggle **Email Verified** to `ON`
5. Click **Save**
6. Go to **Credentials** tab
7. Set temporary or permanent password
8. Assign roles (see Role Management below)

Pre-configured Test Users:

Username	Password	Roles	Use Case
`testuser`	`testpass`	viewer	Read-only testing
`testadmin`	`testpass`	admin	Administrative testing

⚠ **Important:** Remove or secure test users before production deployment.

2.3 User Roles and Permissions

Kamiwaza defines three primary roles:

Role	Permissions	Typical Users
admin	Full access: read, write, delete, configure	System administrators, platform operators
user	Standard access: read, write (no delete/admin)	Data scientists, developers, analysts
viewer	Read-only access	Auditors, observers, stakeholders

Assigning Roles:

1. Navigate to **Users** → Select user
2. Go to **Role Mappings** tab
3. Under **Realm Roles**, select appropriate roles
4. Click **Add selected**
5. Changes take effect immediately (no logout required)

2.4 Password Policies

Configuring Password Requirements:

1. Navigate to **Realm Settings** → **Security Defenses** → **Password Policy**
2. Add policies:
    - **Minimum Length**: 12 characters (recommended)
    - **Uppercase Characters**: Require at least 1
    - **Lowercase Characters**: Require at least 1
    - **Digits**: Require at least 1
    - **Special Characters**: Require at least 1
    - **Not Username**: Prevent username as password
    - **Password History**: Prevent last 3 passwords
    - **Expire Password**: 90 days (recommended)

Password Reset Flow:

1. User clicks "Forgot Password" on login page
2. Keycloak sends password reset email
3. User follows link and sets new password
4. New password must meet policy requirements

⚠️ **Important:** Configure SMTP settings in Keycloak for email-based password reset to function.

3. Role-Based Access Control (RBAC)

3.1 RBAC Policy File

Access control is defined in **YAML policy files** that map endpoints to required roles.

Default Location:

- Host installs: `$KAMIWAZA_ROOT/config/auth_gateway_policy.yaml`
- Docker installs: Mounted at `/app/config/auth_gateway_policy.yaml`

Policy File Structure:

```yaml
version: 1
env: production
default_deny: true  # Block all endpoints unless explicitly allowed

roles:
  - id: admin
    description: "Full system access"
  - id: user
    description: "Standard user access"
  - id: viewer
    description: "Read-only access"

endpoints:
  # Model Management
  - path: "/api/models*"
    methods: ["GET"]
    roles: ["viewer", "user", "admin"]

  - path: "/api/models*"
    methods: ["POST", "PUT", "DELETE"]
    roles: ["user", "admin"]

  # Cluster Management (Admin-only)
  - path: "/api/cluster*"
    methods: ["*"]
    roles: ["admin"]

  # Vector Database (User and Admin)
  - path: "/api/vectordb*"
    methods: ["GET"]
    roles: ["viewer", "user", "admin"]

  - path: "/api/vectordb*"
    methods: ["POST", "PUT", "DELETE"]
    roles: ["user", "admin"]

  # Public endpoints (no auth required)
  - path: "/health"
    methods: ["GET"]
    roles: ["*"]  # Public

  - path: "/docs"
    methods: ["GET"]
    roles: ["*"]  # Public API documentation
```

3.2 Path Matching Rules

Wildcard Patterns:

- `*` matches zero or more characters within a path segment
- `**` matches across multiple path segments

- Patterns are case-sensitive

Examples:

- `/api/models*` matches `/api/models`, `/api/models/123`, `/api/models/search`
- `/api/*/health` matches `/api/models/health`, `/api/cluster/health`
- `/api/**` matches all paths under `/api/`

3.3 Hot Reload (No Restart Required)

The RBAC policy file is automatically reloaded when modified:

1. Edit `auth_gateway_policy.yaml`
2. Save the file
3. Changes take effect within seconds
4. Monitor logs for reload confirmation:

```
INFO: Policy reloaded successfully from /app/config/auth_gateway_policy.yaml
```

⚠️ **Important:** Invalid YAML syntax will prevent reload and retain the previous valid configuration.

3.4 Adding Custom Endpoints

Example: Protecting a new analytics endpoint

```yaml
endpoints:
  # Add new analytics endpoint
  - path: "/api/analytics/reports*"
    methods: ["GET"]
    roles: ["user", "admin"]

  - path: "/api/analytics/reports*"
    methods: ["POST", "DELETE"]
    roles: ["admin"]
```

Testing Access Control:

```
# Get token for viewer role (should be denied POST)
VIEWER_TOKEN=$(curl -s -X POST
http://localhost:8080/realms/kamiwaza/protocol/openid-connect/token \
  -d "grant_type=password" \
  -d "client_id=kamiwaza-platform" \
  -d "username=testuser" \
  -d "password=testpass" | jq -r .access_token)

# Test (expect 403 Forbidden)
curl -H "Authorization: Bearer $VIEWER_TOKEN" \
  -X POST http://localhost:7777/api/analytics/reports

# Get token for admin role (should succeed)
ADMIN_TOKEN=$(curl -s -X POST
http://localhost:8080/realms/kamiwaza/protocol/openid-connect/token \
  -d "grant_type=password" \
  -d "client_id=kamiwaza-platform" \
  -d "username=testadmin" \
  -d "password=testpass" | jq -r .access_token)

# Test (expect 200 OK)
curl -H "Authorization: Bearer $ADMIN_TOKEN" \
  -X POST http://localhost:7777/api/analytics/reports
```

4. Identity Provider Integration

4.1 Keycloak Configuration

Realm: `kamiwaza` **Client ID:** `kamiwaza-platform`

Client Configuration Settings:

Setting	Value	Purpose
Access Type	Public (SPA) or Confidential (backend)	Authentication flow type
Valid Redirect URIs	`https://your-domain.com/*`	Allowed OAuth callback URLs
Web Origins	`https://your-domain.com`	CORS configuration
Direct Access Grants	Enabled (dev), Disabled (prod)	Password grant for testing

4.2 OAuth 2.0 / OpenID Connect Integration

Kamiwaza supports standard OIDC authentication flows.

Environment Configuration:

```
# Keycloak OIDC Settings
AUTH_GATEWAY_KEYCLOAK_URL=https://auth.yourdomain.com
AUTH_GATEWAY_KEYCLOAK_REALM=kamiwaza
AUTH_GATEWAY_KEYCLOAK_CLIENT_ID=kamiwaza-platform

# JWT Validation
AUTH_GATEWAY_JWT_ISSUER=https://auth.yourdomain.com/realms/kamiwaza
AUTH_GATEWAY_JWT_AUDIENCE=kamiwaza-platform
AUTH_GATEWAY_JWKS_URL=https://auth.yourdomain.com/realms/kamiwaza/protocol/openid-
connect/certs
```

OIDC Discovery Endpoint:

```
https://auth.yourdomain.com/realms/kamiwaza/.well-known/openid-configuration
```

4.3 SAML Integration

Configure SAML Identity Provider in Keycloak:

1. Navigate to **Identity Providers** in Keycloak admin console
2. Select **SAML v2.0**
3. Configure SAML settings:
    - **Single Sign-On Service URL**: Your IdP's SSO endpoint
    - **Single Logout Service URL**: Your IdP's logout endpoint
    - **NameID Policy Format**: `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`
    - **Principal Type**: Subject NameID
4. Upload IdP metadata XML or configure manually
5. Map SAML attributes to Keycloak user attributes
6. Enable identity provider in login flow

Attribute Mapping Example:

```
SAML Attribute          → Keycloak Attribute
-----------------         ------------------
email                   → email
firstName               → firstName
lastName                → lastName
memberOf                → roles
```

4.4 LDAP / Active Directory Integration

Configure LDAP Federation:

1. Navigate to **User Federation** → **Add provider** → **ldap**
2. Configure connection settings:
   - **Connection URL**: `ldap://ldap.company.com:389` or `ldaps://` for SSL
   - **Bind DN**: `cn=admin,dc=company,dc=com`
   - **Bind Credential**: LDAP admin password
3. Configure LDAP search settings:
   - **Users DN**: `ou=users,dc=company,dc=com`
   - **User Object Classes**: `inetOrgPerson, organizationalPerson`
   - **Username LDAP attribute**: `uid` or `sAMAccountName` (AD)
   - **RDN LDAP attribute**: `uid` or `cn`
   - **UUID LDAP attribute**: `entryUUID` or `objectGUID` (AD)
4. Save and test connection
5. Synchronize users: **Synchronize all users** button

Active Directory Specific Settings:

- **Vendor**: Active Directory
- **Username LDAP attribute**: `sAMAccountName`
- **RDN LDAP attribute**: `cn`
- **UUID LDAP attribute**: `objectGUID`
- **User Object Classes**: `person, organizationalPerson, user`

Role Mapping from LDAP Groups:

1. Go to **Mappers** tab in LDAP federation
2. Create new mapper: **group-ldap-mapper**

- **Mapper Type**: `group-ldap-mapper`
- **LDAP Groups DN**: `ou=groups,dc=company,dc=com`
- **Group Name LDAP Attribute**: `cn`
- **Group Object Classes**: `groupOfNames`
- **Membership LDAP Attribute**: `member`
- **Mode**: `READ_ONLY` or `LDAP_ONLY`

3. Map LDAP groups to Keycloak roles in **Role Mappings**

4.5 Single Sign-On (SSO) Setup

Google SSO Integration:

1. Create OAuth 2.0 credentials in Google Cloud Console (https://console.cloud.google.com/apis/credentials)
2. Configure authorized redirect URI:

```
https://auth.yourdomain.com/realms/kamiwaza/broker/google/endpoint
```

3. In Keycloak, navigate to **Identity Providers** → **Google**
4. Enter **Client ID** and **Client Secret** from Google Console
5. Save and enable

Environment Configuration:

```
# Google SSO
GOOGLE_CLIENT_ID=your-google-client-id
GOOGLE_CLIENT_SECRET=your-google-client-secret
```

Microsoft Azure AD / Office 365:

1. Register application in Azure Portal (https://portal.azure.com)
2. Configure redirect URI: `https://auth.yourdomain.com/realms/kamiwaza/broker/oidc/endpoint`
3. In Keycloak, add **OpenID Connect v1.0** provider
4. Configure with Azure AD settings:

  - **Authorization URL**: `https://login.microsoftonline.com/{tenant}/oauth2/v2.0/authorize`
  - **Token URL**: `https://login.microsoftonline.com/{tenant}/oauth2/v2.0/token`
  - **Client ID**: Azure application ID
  - **Client Secret**: Azure client secret

Testing SSO:

1. Navigate to Kamiwaza login page
2. Click SSO provider button (Google, Azure, etc.)
3. Authenticate with external identity provider
4. First-time users automatically create Keycloak account
5. Subsequent logins use existing account

---

5. Security Configuration

5.1 JWT Token Configuration

Token Security Settings:

```
# JWT Validation (in env.sh)
AUTH_GATEWAY_JWT_AUDIENCE=kamiwaza-platform  # Required audience claim
AUTH_GATEWAY_JWT_ISSUER=https://auth.yourdomain.com/realms/kamiwaza
AUTH_GATEWAY_JWKS_URL=https://auth.yourdomain.com/realms/kamiwaza/protocol/openid-
connect/certs

# Security Hardening
AUTH_REQUIRE_SUB=true  # Require 'sub' claim (user ID) in tokens
AUTH_EXPOSE_TOKEN_HEADER=false  # Don't expose tokens in response headers
(production)
AUTH_ALLOW_UNSIGNED_STATE=false  # Require signed OIDC state parameter
(production)
```

Token Algorithms:

- **Supported**: RS256 (RSA with SHA-256) - asymmetric cryptography
- **Not Supported**: HS256, ES256, or other algorithms

5.2 Session Management

Access Token Expiration:

Configure in Keycloak: **Realm Settings → Tokens**

- **Access Token Lifespan**: 1 hour (default), 5-15 minutes (high security)
- **Refresh Token Lifespan**: 30 days (default)
- **SSO Session Idle**: 30 minutes
- **SSO Session Max**: 10 hours

Session Timeout Configuration:

```
# In env.sh
AUTH_GATEWAY_TOKEN_LEEWAY=30  # Clock skew tolerance (seconds)
AUTH_GATEWAY_JWKS_CACHE_TTL=300  # JWKS cache duration (5 minutes)
```

Best Practices:

- Short-lived access tokens (5-15 minutes) for high-security environments
- Longer refresh tokens (days) for user convenience
- Implement token refresh in client applications
- Use secure, HTTP-only cookies for browser sessions

5.3 HTTPS Enforcement

Production HTTPS Requirements:

Kamiwaza enforces HTTPS in production and CI environments when `CI=true` or `KAMIWAZA_ENV=production`.

TLS Configuration:

1. Obtain SSL/TLS certificates (Let's Encrypt, commercial CA, etc.)
2. Configure Traefik with TLS:

```
# traefik-dynamic.yml
tls:
  certificates:
    - certFile: /certs/your-domain.crt
      keyFile: /certs/your-domain.key
  options:
    default:
      minVersion: VersionTLS12
      cipherSuites:
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

3. Update environment:

```
AUTH_GATEWAY_KEYCLOAK_URL=https://auth.yourdomain.com
KAMIWAZA_HTTPS=true
```

5.4 Rate Limiting (Optional - Requires Redis)

Rate limiting requires Redis configuration:

```
# Redis connection for rate limiting
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_DB=0
```

Rate Limit Configuration:

```
# In auth_gateway_policy.yaml
rate_limits:
  - path: "/api/models*"
    requests_per_minute: 100
    per_user: true

  - path: "/api/auth/token"
    requests_per_minute: 10
    per_ip: true
```

6. Monitoring & Troubleshooting

6.1 Health Checks

Auth Service Health Endpoint:

```
curl http://localhost:7777/health
```

Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "uptime": 3600.5,
  "KAMIWAZA_USE_AUTH": true,
  "jwks_cache_status": "healthy"
}
```

Keycloak Health Check:

```
curl http://localhost:8080/health/ready
```

6.2 Log Monitoring

Auth Service Logs:

```
# Docker deployments
docker logs kamiwaza-api -f | grep AUTH

# Host deployments
tail -f $KAMIWAZA_LOG_DIR/kamiwaza.log | grep AUTH
```

Important Log Events:

- `AUTH_FAILED` - Authentication failure with reason

- `ACCESS_DENIED` - Authorization denial with path/method/roles
- `JWKS_REFRESHED` - JWKS key cache refresh
- `POLICY_RELOADED` - RBAC policy file reload
- `TOKEN_VALIDATED` - Successful token validation

Keycloak Logs:

```
docker logs kamiwaza-keycloak -f
```

6.3 Common Issues and Solutions

Issue: 401 Unauthorized on All Requests

Symptoms: All API requests return 401 even with valid tokens

Troubleshooting:

1. **Check if auth is enabled:**

```
echo $KAMIWAZA_USE_AUTH  # Should be 'true'
```

2. **Verify Keycloak is running:**

```
docker ps | grep keycloak
curl http://localhost:8080/health/ready
```

3. **Check JWT issuer matches:**

```
# Decode your token
echo $TOKEN | cut -d. -f2 | base64 -d | jq .iss

# Compare with configuration
echo $AUTH_GATEWAY_JWT_ISSUER
```

4. **Verify JWKS endpoint is accessible:**

```
curl $AUTH_GATEWAY_JWKS_URL
```

Solution:

- Ensure `AUTH_GATEWAY_JWT_ISSUER` matches token issuer exactly
- Verify Keycloak realm name is correct
- Check network connectivity to Keycloak

Issue: 403 Forbidden (Valid Token)

Symptoms: Token is valid but access denied

Troubleshooting:

1. **Check user roles in token:**

   ```
   echo $TOKEN | cut -d. -f2 | base64 -d | jq .realm_access.roles
   ```

2. **Verify RBAC policy allows access:**

   ```
   cat $KAMIWAZA_ROOT/config/auth_gateway_policy.yaml
   ```

3. **Check policy file syntax:**

   ```
   # Invalid YAML prevents policy reload
   yamllint $KAMIWAZA_ROOT/config/auth_gateway_policy.yaml
   ```

Solution:

- Add required roles to user in Keycloak
- Update RBAC policy to allow endpoint/method/role combination
- Fix YAML syntax errors and reload policy

Issue: Token Expired Too Quickly

Symptoms: Tokens expire after minutes instead of expected duration

Troubleshooting:

1. **Check token lifespan in Keycloak:**
    - Navigate to **Realm Settings** → **Tokens**
    - Verify **Access Token Lifespan** setting
2. **Check token claims:**

```
echo $TOKEN | cut -d. -f2 | base64 -d | jq '.exp - .iat'
# Result is token lifetime in seconds
```

Solution:

- Increase **Access Token Lifespan** in Keycloak (for development)
- Implement token refresh in client applications
- Use refresh tokens for long-lived sessions

Issue: Google/SSO Login Not Working

Symptoms: SSO redirect fails or returns error

Troubleshooting:

1. **Check redirect URI configuration:**
    - Verify redirect URI in Google/Azure console matches Keycloak exactly
    - Format: `https://auth.yourdomain.com/realms/kamiwaza/broker/{provider}/endpoint`
2. **Verify client secret is set:**

```
echo $GOOGLE_CLIENT_SECRET  # Should not be empty
```

3. **Check Keycloak identity provider logs:**

```
docker logs kamiwaza-keycloak -f | grep -i broker
```

Solution:

- Update authorized redirect URIs in OAuth provider console
- Ensure client secret is configured in Keycloak
- Enable identity provider in Keycloak authentication flow

6.4 Diagnostic Commands

Test Token Generation:

```
# Get token from Keycloak
TOKEN=$(curl -s -X POST http://localhost:8080/realms/kamiwaza/protocol/openid-connect/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=password" \
  -d "client_id=kamiwaza-platform" \
  -d "username=testuser" \
  -d "password=testpass" | jq -r .access_token)

# Decode token to inspect claims
echo $TOKEN | cut -d. -f2 | base64 -d | jq .
```

Test Token Validation:

```
# Test ForwardAuth validation endpoint directly
curl -v -H "Authorization: Bearer $TOKEN" \
  -H "X-Forwarded-Uri: /api/models" \
  -H "X-Forwarded-Method: GET" \
  http://localhost:7777/auth/validate
```

Verify JWKS Endpoint:

```
# Fetch public keys for signature validation
curl http://localhost:8080/realms/kamiwaza/protocol/openid-connect/certs | jq .
```

Check RBAC Policy:

```
# View current policy
cat $KAMIWAZA_ROOT/config/auth_gateway_policy.yaml

# Watch for policy reload events
tail -f $KAMIWAZA_LOG_DIR/kamiwaza.log | grep POLICY_RELOADED
```

Appendix A: Environment Variable Reference

Core Authentication

Variable	Description	Default
`KAMIWAZA_USE_AUTH`	Enable/disable authentication	`true`
`AUTH_GATEWAY_JWT_ISSUER`	Expected JWT issuer URL	-
`AUTH_GATEWAY_JWT_AUDIENCE`	Expected JWT audience claim	-
`AUTH_GATEWAY_JWKS_URL`	JWKS endpoint for key fetching	-
`AUTH_GATEWAY_POLICY_FILE`	Path to RBAC policy file	`$KAMIWAZA_ROOT/config/auth_gateway_policy`

Keycloak Configuration

Variable	Description	Default	Required
`AUTH_GATEWAY_KEYCLOAK_URL`	Keycloak base URL	`http://localhost:8080`	Yes
`AUTH_GATEWAY_KEYCLOAK_REALM`	Keycloak realm name	`kamiwaza`	Yes
`AUTH_GATEWAY_KEYCLOAK_CLIENT_ID`	OAuth client ID	`kamiwaza-platform`	Yes
`KEYCLOAK_ADMIN_PASSWORD`	Keycloak admin password	`admin`	Yes

Security Hardening

Variable	Description	Default	Required
`AUTH_REQUIRE_SUB`	Require 'sub' claim in tokens	`false`	No
`AUTH_EXPOSE_TOKEN_HEADER`	Expose token in response headers	`true`	No
`AUTH_ALLOW_UNSIGNED_STATE`	Allow unsigned OIDC state	`true` (dev only)	No
`AUTH_GATEWAY_TOKEN_LEEWAY`	Clock skew tolerance (seconds)	`30`	No
`AUTH_GATEWAY_JWKS_CACHE_TTL`	JWKS cache duration (seconds)	`300`	No

External Identity Providers

Variable	Description	Default	Required
`GOOGLE_CLIENT_ID`	Google OAuth client ID	-	For Google SSO
`GOOGLE_CLIENT_SECRET`	Google OAuth client secret	-	For Google SSO

Appendix B: RBAC Policy Examples

Example 1: Tiered Access by Service

```
version: 1
env: production
default_deny: true

roles:
  - id: admin
    description: "System administrators"
  - id: data_scientist
    description: "Data scientists and ML engineers"
  - id: analyst
    description: "Business analysts and viewers"

endpoints:
  # Model Management - Scientists can create/edit, analysts read-only
  - path: "/api/models*"
    methods: ["GET"]
    roles: ["admin", "data_scientist", "analyst"]

  - path: "/api/models*"
    methods: ["POST", "PUT", "DELETE"]
    roles: ["admin", "data_scientist"]

  # Model Serving - Scientists can deploy, analysts can query
  - path: "/api/serving/deployments*"
    methods: ["GET"]
    roles: ["admin", "data_scientist", "analyst"]

  - path: "/api/serving/deploy"
    methods: ["POST"]
    roles: ["admin", "data_scientist"]

  - path: "/api/serving/generate"
    methods: ["POST"]
    roles: ["admin", "data_scientist", "analyst"]

  # Cluster Management - Admin-only
  - path: "/api/cluster*"
    methods: ["*"]
    roles: ["admin"]

  # Public endpoints
  - path: "/health"
    methods: ["GET"]
    roles: ["*"]
```

Example 2: Read-Write Separation

```yaml
version: 1
env: production
default_deny: true

roles:
  - id: admin
  - id: editor
  - id: reader

endpoints:
  # Read endpoints - All authenticated users
  - path: "/api/models"
    methods: ["GET"]
    roles: ["admin", "editor", "reader"]

  - path: "/api/vectordb/collections"
    methods: ["GET"]
    roles: ["admin", "editor", "reader"]

  # Write endpoints - Editors and admins only
  - path: "/api/models"
    methods: ["POST", "PUT"]
    roles: ["admin", "editor"]

  - path: "/api/vectordb/collections"
    methods: ["POST", "PUT"]
    roles: ["admin", "editor"]

  # Delete endpoints - Admins only
  - path: "/api/models*"
    methods: ["DELETE"]
    roles: ["admin"]

  - path: "/api/vectordb/collections*"
    methods: ["DELETE"]
    roles: ["admin"]
```

System Requirements

Base System Requirements

Supported Operating Systems & Architecture

- **Linux**:
  - Ubuntu 24.04 and 22.04 LTS via .deb package installation (x64/amd64 architecture only)
  - Redhat Enterprise Linux (RHEL) 9
- **Windows**: 11 (x64 architecture) via WSL with MSI installer
- **macOS**: 12.0 or later, Apple Silicon (ARM64) only (community edition only)

CPU Requirements

- **Architecture**:
  - Linux: x64/amd64 (64-bit)
  - macOS: ARM64 (Apple Silicon) only
  - Windows: x64 (64-bit)
- **Minimum Cores**: 8+ cores
- **Recommended Cores**: 16+ cores for CPU-based inference workloads

Core Software Requirements

- **Python**: Python 3.10 for tarball installations; Python 3.12 for .deb/.msi installations
- **Docker**: Docker Engine with Compose v2
- **Node.js**: 22.x (installed via NVM during setup)
- **Browser**: Chrome Version 141+ (tested and recommended)
- **GPU Support**: NVIDIA GPU with compute capability 7.0+ (Linux only) or NVIDIA RTX/Intel Arc (Windows via WSL)

Memory Requirements

System RAM

- **Minimum**: 16GB RAM
- **Recommended**: 32GB+ RAM for CPU-based inference workloads
- **GPU Workloads**: 16GB+ system RAM (32GB+ recommended)

GPU Memory (vRAM)

- **GPU Inference**: 16GB+ vRAM required
- **Recommended**: 32GB+ vRAM for optimal GPU inference performance

Windows (WSL-based) Specific

- **Minimum**: 16GB RAM
- **Recommended**: 32GB+ RAM
- **Memory Allocation**: 50-75% of system RAM dedicated to Kamiwaza during installation

Storage Requirements

Storage Performance

- **Required**: SSD (Solid State Drive)
- **Preferred**: NVMe SSD for optimal performance
- **Minimum**: SATA SSD
- **Note**: Models weights can be on a separate HDD but loads time will increase

Storage Capacity

Linux/macOS

- **Minimum**: 100GB free disk space
- **Recommended**: 200GB+ free disk space
- **Enterprise Edition**: Additional space for /opt/kamiwaza persistence

Windows

- **Minimum**: 100GB free disk space
- **Recommended**: 200GB+ free space on SSD

- **WSL**: Automatically manages Ubuntu 24.04 installation space

📋 **For detailed Windows storage and configuration requirements, see the Windows Installation Guide.**

Hardware Recommendation Tiers

Kamiwaza is a distributed AI platform built on Ray that supports both CPU-only and GPU-accelerated inference. Hardware requirements vary significantly based on:

- **Model size**: From 0.6B to 70B+ parameters
- **Deployment scale**: Single-node development vs multi-node production
- **Inference engine**: LlamaCpp (CPU/GPU), VLLM (GPU), MLX (Apple Silicon)
- **Workload type**: Interactive chat, batch processing, RAG pipelines

GPU Memory Requirements by Model Size

The table below provides real-world GPU memory requirement estimates for representative models at different scales. These estimates assume FP8 and include overhead for context windows and batch processing.

Model Example	Parameters	Minimum vRAM	Notes
GPT-OSS 20B	20B	24GB	Includes weights + 1-batch max context; fits 1x 24GB GPU (e.g., L4/RTX 4090)
GPT-OSS 120B	120B	80GB	~40GB weights + 1-batch max context; 1x H100/H200 or 2x A100 80GB recommended
Qwen 3 235B A22B	235B	150GB	~120GB weights + 1-batch max context; 2x H200 (282GB) or 2x B200 (384GB) ideal for max context
Qwen 3-VL 235B A22B	235B	150GB	Same base minimum (includes 1-batch max context); budget +20-30% vRAM for high-res vision inputs

Key Considerations:

- **Minimum vRAM**: FP8 weights + 1-batch allocation at your target max context
- **Headroom**: For longer contexts, larger batch sizes, and concurrency, budget additional vRAM beyond minimums
- **Vision Workloads**: Image/video processing adds overhead; budget 20-30% more for vision-language models
- **Tensor Parallelism**: Distributing large models (120B+) across multiple GPUs requires high-bandwidth interconnects (NVLink 3.0+)

Tier 1: Development & Small Models

Use Case: Local development, testing, small to medium model deployment (up to 13B parameters)

Hardware Specifications:

- **CPU:** 8-16 cores / 16-32 threads
- **RAM:** 32GB (16GB minimum)
- **Storage:** 200GB NVMe SSD (100GB minimum)
- **GPU:** Optional - Single GPU with 16-24GB VRAM
  - NVIDIA RTX 4090 (24GB)
  - NVIDIA RTX 4080 (16GB)
  - NVIDIA T4 (16GB)
- **Network:** 1-10 Gbps

Workload Capacity:

- Low-volume workloads: 1-10 concurrent requests (supports dozens of interactive users)
- Development, testing, and proof-of-concept deployments
- Light production workloads

Tier 2: Production - Medium to Large Models

Use Case: Production deployment of medium to large models (13B-70B parameters), high throughput

Hardware Specifications:

- **CPU:** 32 cores / 64 threads
- **RAM:** 128-256GB system RAM

- **Storage:** 1-2TB NVMe SSD
- **GPU:** 1-4 GPUs with 40GB+ VRAM each
    - 1-4x NVIDIA B200 (192GB HBM3e)
    - 1-4x NVIDIA H200 (141GB HBM3e)
    - 1-4x NVIDIA RTX 6000 Pro Blackwell (48GB)
    - 1-2x NVIDIA H100 (80GB)
    - 1-4x NVIDIA A100 (40GB or 80GB)
    - 1-2x NVIDIA L40S (48GB)
    - 2-4x NVIDIA A10G (24GB) for tensor parallelism
- **Network:** 25-40 Gbps

Workload Capacity:

- Medium-scale production: 100s to 1,000+ concurrent requests (supports thousands of interactive users)
- Example: Per-GPU batch size of 32 across 8 GPUs = 256 concurrent requests; batch size of 128 = 1,024 requests
- Production chat applications
- Complex RAG pipelines with embedding generation
- Batch inference

Tier 3: Enterprise Multi-Node Cluster

Use Case: Enterprise deployment with multiple models, high availability, horizontal scaling, 99.9%+ SLA

Cluster Architecture:

Head Node (Control Plane):

- **CPU:** 16 cores / 32 threads
- **RAM:** 64GB
- **Storage:** 500GB NVMe SSD
- **GPU:** Same class as worker nodes (homogeneous cluster recommended)
- **Role:** Ray head, API gateway, scheduling, monitoring (head performs minimal extra work; Ray backend load is distributed across nodes)

Worker Nodes (3+ nodes for HA):

- **CPU:** 32-64 cores / 64-128 threads per node
- **RAM:** 256-512GB per node
- **Storage:** 2TB NVMe SSD per node (local cache)
- **GPU:** 4-8 GPUs per node (same class as head node)
- **Network:** 40-100 Gbps (InfiniBand for HPC workloads)

> Note: For Enterprise Edition production clusters, avoid non-homogeneous hardware (e.g., GPU-less head nodes). Each node participates in data plane duties (Traefik gateway, HTTP proxying, etc.), so matching GPU capabilities simplifies scheduling and maximizes throughput.

Shared Storage:

- High-performance NAS or distributed filesystem (Lustre, CephFS)
- 10TB+ capacity, NVMe-backed
- 10+ GB/s aggregate sequential throughput
- Low-latency access (< 5ms) from all nodes

Workload Capacity:

- Multiple models deployed simultaneously
- High-scale production: 1,000–10,000+ concurrent requests (supports tens of thousands of interactive users)
- Batch sizes scale with GPU count and model size; smaller requests enable higher throughput per GPU
- High availability with automatic failover
- Horizontal auto-scaling based on load
- Production SLAs (99.9% uptime)

Cloud Provider Instance Mapping

AWS EC2 Instance Types

Tier	Instance Type	vCPU	RAM	GPU	Storage
Tier 1: CPU-only	`m6i.2xlarge`	8	32GB	None	200GB gp3
Tier 1: With GPU	`g5.xlarge`	4	16GB	1x A10G (24GB)	200GB gp3
Tier 1: Alternative	`g5.2xlarge`	8	32GB	1x A10G (24GB)	200GB gp3
Tier 2: Multi-GPU	`g5.12xlarge`	48	192GB	4x A10G (96GB)	2TB gp3
Tier 2: Alternative	`p4d.24xlarge`	96	1152GB	8x A100 (320GB)	2TB gp3
Tier 3: All Nodes	`p4d.24xlarge`	96	1152GB	8x A100 (320GB)	2TB gp3

Notes:

- Use `gp3` SSD volumes (not `gp2`) for better performance/cost
- For Tier 3 shared storage: Amazon FSx for Lustre or EFS (with Provisioned Throughput)
- Use Placement Groups for low-latency multi-node clusters (Tier 3)
- H100 instances (`p5.48xlarge`) available in limited regions for highest performance
- Latest options: Emerging `p6`/`p6e` families with H200/B200/Grace-Blackwell are rolling out in select regions; map to Tier 2/3 as available.

Google Cloud Platform (GCP) Instance Types

Tier	Machine Type	vCPU	RAM	GPU	Storage
Tier 1: CPU-only	`n2-standard-8`	8	32GB	None	200GB SSD
Tier 1: With GPU	`n1-standard-8` + `1x T4`	8	30GB	1x T4 (16GB)	200GB SSD
Tier 1: Alternative	`g2-standard-8` + `1x L4`	8	32GB	1x L4 (24GB)	200GB SSD
Tier 2: Multi-GPU	`a2-highgpu-4g`	48	340GB	4x A100 (160GB)	2TB SSD
Tier 2: Alternative	`g2-standard-48` + `4x L4`	48	192GB	4x L4 (96GB)	2TB SSD
Tier 3: All Nodes	`a2-highgpu-8g`	96	680GB	8x A100 (320GB)	2TB SSD

Notes:

- Use `pd-ssd` or `pd-balanced` persistent disks (not `pd-standard`)
- For Tier 3 shared storage: Filestore High Scale tier (up to 10 GB/s)
- Use Compact Placement for low-latency multi-node clusters (Tier 3)
- L4 GPUs (24GB) available as cost-effective alternative to A100
- Latest options: Blackwell/H200 classes are entering preview/limited availability; consider AI Hypercomputer offerings as they launch.

Microsoft Azure Instance Types

Tier	VM Size	vCPU	RAM	GPU	Storage
Tier 1: CPU-only	`Standard_D8s_v5`	8	32GB	None	200GB Premium SSD
Tier 1: With GPU	`Standard_NC4as_T4_v3`	4	28GB	1x T4 (16GB)	200GB Premium SSD
Tier 1: Alternative	`Standard_NC6s_v3`	6	112GB	1x V100 (16GB)	200GB Premium SSD
Tier 2: H100 (recommended)	`Standard_NC40ads_H100_v5`	40	320GB	1x H100 (80GB)	2TB Premium SSD
Tier 2: H100 Multi-GPU	`Standard_NC80adis_H100_v5`	80	640GB	2x H100 (160GB)	2TB Premium SSD
Tier 2: A100 Multi-GPU	`Standard_NC96ads_A100_v4`	96	880GB	4x A100 (320GB)	2TB Premium SSD
Tier 2: A100 Alternative	`Standard_NC48ads_A100_v4`	48	440GB	2x A100 (160GB)	2TB Premium SSD
Tier 3: H100 (recommended)	`Standard_ND96isr_H100_v5`	96	1900GB	8x H100 (640GB)	2TB Premium SSD
Tier 3: A100 Alternative	`Standard_ND96asr_v4`	96	900GB	8x A100 (320GB)	2TB Premium SSD

Notes:

- Use Premium SSD (not Standard HDD or Standard SSD)

- For Tier 3 shared storage: Azure NetApp Files Premium or Ultra tier
- Use Proximity Placement Groups for low-latency multi-node clusters (Tier 3)
- NDm A100 v4 series offers InfiniBand networking for HPC workloads
- Latest options: Blackwell/H200-based VM families are announced/rolling out; align Tier 2/3 to those SKUs where available.

Windows-Specific Prerequisites

- Windows Subsystem for Linux (WSL) installed and enabled
- Administrator access required for initial setup
- Windows Terminal (recommended for optimal WSL experience)

Dependencies & Components

Required System Packages

See platform-specific installation instructions

NVIDIA Components (Linux GPU Support)

- NVIDIA Driver (550-server recommended)
- NVIDIA Container Toolkit
- nvidia-docker2

Windows Components (Automated via MSI Installer)

- Windows Subsystem for Linux (WSL 2)
- Ubuntu 24.04 LTS (automatically downloaded and configured)
- Docker Engine (configured within WSL)
- GPU drivers and runtime (automatically detected and configured)
- Node.js 22 (via NVM within WSL environment)

Docker Configuration Requirements

- Docker Engine with Compose v2
- User must be in docker group
- Swarm mode (Enterprise Edition)

- Docker data root configuration (configurable)

Required Directory Structure

Enterprise Edition

Note this is created by the installer and present in cloud marketplace images.

```
/etc/kamiwaza/
├── config/
├── ssl/      # Cluster certificates
└── swarm/    # Swarm tokens

/opt/kamiwaza/
├── containers/  # Docker root (configurable)
├── logs/
├── nvm/        # Node Version Manager
└── runtime/    # Runtime files
```

Community Edition

We recommend `${HOME}/kamiwaza` or something similar for `KAMIWAZA_ROOT`.

```
$KAMIWAZA_ROOT/
├── env.sh
├── runtime/
└── logs/
```

Network Configuration

Network Bandwidth Requirements

Single Node Deployment

Network Bandwidth:

- **Minimum:** 1 Gbps (for model downloads, API traffic)
- **Recommended:** 10 Gbps (for high-throughput inference)

Considerations:

- Internet bandwidth for downloading models from HuggingFace (one-time)
- Client API traffic for inference requests/responses
- Monitoring and logging egress

Multi-Node Cluster

Inter-Node Network:

- **Minimum:** 10 Gbps Ethernet
- **Recommended:** 25-40 Gbps Ethernet or InfiniBand
- **Latency:** < 1ms between nodes (same datacenter/availability zone)

Why It Matters:

- Ray distributed scheduling requires low-latency communication
- Tensor parallelism transfers large model shards between GPUs
- Shared storage access impacts model loading performance

Required Kernel Modules (Enterprise Edition Linux Only)

Required modules for Swarm container networking:

- overlay
- br_netfilter

System Network Parameters (Enterprise Edition Linux Only)

These will be set by the installer.

```
# Required sysctl settings for Swarm networking
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
```

Community Edition Networking

- Uses standard Docker bridge networks

- No special kernel modules or sysctl settings required
- Simplified single-node networking configuration

Detailed Storage Requirements

Capacity Planning

Component	Minimum	Recommended	Notes
Operating System	20GB	50GB	Ubuntu/RHEL base + dependencies
Kamiwaza Platform	50GB	50GB	Python environment, Ray, services
Model Storage	50GB	500GB+	Depends on number and size of models
Database	10GB	50GB	CockroachDB for metadata
Vector Database	10GB	100GB+	For embeddings (if enabled)
Logs & Metrics	10GB	50GB	Rotated logs, Ray dashboard data
Scratch Space	20GB	100GB	Temporary files, downloads, builds
Total	**170GB**	**900GB+**	

Storage Performance Requirements

Local Storage (Single Node)

Storage Type:

- **Minimum:** SATA SSD (500 MB/s sequential read)
- **Recommended:** NVMe SSD (2000+ MB/s sequential read)
- **Note:** HDD: Only recommended for non-dynamic model loads and low KV cache usage - model load times can be very long (15+ minutes); models are in memory after load

Performance Targets:

- **Sequential Read:** 2000+ MB/s (model loading)
- **Sequential Write:** 1000+ MB/s (model downloads, checkpoints)
- **4K Random Read IOPS:** 50,000+ (database, concurrent access)
- **4K Random Write IOPS:** 20,000+ (database writes, logs)

Why It Matters:

- 7B model (14GB): Loads in ~7 seconds on NVMe vs ~28 seconds on SATA SSD
- Concurrent model loads across Ray workers stress random read performance
- Database query performance directly tied to IOPS

Shared Storage (Multi-Node Clusters)

Network Filesystem Requirements:

- **Protocol:** NFSv4, Lustre, CephFS, or S3-compatible object storage
- **Network Bandwidth:** 10 Gbps minimum, 40+ Gbps for production
- **Network Latency:** < 5ms between nodes and storage
- **Sequential Throughput:** 5+ GB/s aggregate (10+ GB/s for large clusters)

Object Storage (Alternative):

- S3-compatible API (AWS S3, GCS, MinIO, etc.)
- Local caching layer recommended for frequently accessed models
- Consider bandwidth costs for cloud object storage

Shared Storage Options:

Solution	Use Case	Throughput	Cost Profile
NFS over NVMe	Small clusters (< 5 nodes)	1-5 GB/s	Low (commodity hardware)
AWS FSx for Lustre	AWS multi-node clusters	1-10 GB/s	Medium (pay per GB/month + throughput)
GCP Filestore High Scale	GCP multi-node clusters	Up to 10 GB/s	Medium-High
Azure NetApp Files Ultra	Azure multi-node clusters	Up to 10 GB/s	High
CephFS	On-premises clusters	5-20 GB/s	Medium (requires Ceph cluster)
Object Storage + Cache	Cost-optimized	Varies	Low storage, high egress

Storage Configuration by Edition

Enterprise Edition Requirements

- Primary mountpoint for persistent storage (/opt/kamiwaza)
- Scratch/temporary storage (auto-configured)
- For Azure: Additional managed disk for persistence
- Shared storage for multi-node clusters (see Shared Storage Options above)

Community Edition

- Local filesystem storage
- Configurable paths via environment variables
- Single-node storage only (no shared storage required)

Special Considerations

Apple Silicon (M-Series)

MLX Engine Support:

- Kamiwaza supports Apple Silicon via the MLX inference engine
- Unified memory architecture (shared CPU/GPU RAM)
- Excellent performance for models up to 13B parameters; reasonable performance for larger models when context is appropriately restricted and RAM is available.
- All M-series chips work in approximately the same way, but newer chips (e.g., M4) offer substantially higher performance than older versions
- Ultra chips (Mac Studio/Mac Pro models) typically offer 50-80% more performance than Pro versions

Notes:

- No tensor parallelism support (single chip only)
- Not for production use; like-for-like API, UI, capabilities.
- Community edition only; single node only (Enterprise edition not available on macOS)

Important Notes

- **System Impact**: Network and kernel configurations can affect other services
- **Security**: Certificate generation and management for cluster communications
- **GPU Support**: Available on Linux (NVIDIA GPUs) and Windows (NVIDIA RTX, Intel Arc via WSL)
- **Storage**: Enterprise Edition requires specific storage configuration
- **Network**: Enterprise Edition requires specific network ports for cluster communication
- **Docker**: Custom Docker root configuration may affect other containers
- **Windows Edition**: Requires WSL 2 and will create a dedicated Ubuntu 24.04 instance
- **Administrator Access**: Windows installation requires administrator privileges for initial setup

Additional Considerations

Network Ports

Linux/macOS Enterprise Edition

- 443/tcp: HTTPS primary access
- 51100-51199/tcp: Deployment ports for model instances (will also be used for 'App Garden' in the future)

Windows Edition

- 443/tcp: HTTPS primary access (via WSL)
- 61100-61299/tcp: Reserved ports for Windows installation

Version Compatibility

- Docker Engine: 20.10 or later
- NVIDIA Driver: 450.80.02 or later
- ETCD: 3.5 or later
- Node.js: 22.x (installed automatically)

Installing Kamiwaza

Before You Begin

Please review the System Requirements before proceeding with installation. This document covers:

- Supported operating systems and versions
- Hardware requirements (CPU, RAM, storage)
- Required system packages and dependencies
- Network and storage configuration
- GPU support requirements

Installation Workflows

Linux

Ubuntu .deb Package Installation (for Ubuntu 24.04 Noble)

1. Add Kamiwaza repository to APT sources

```
echo "deb [signed-by=/usr/share/keyrings/kamiwaza-archive-keyring.gpg]
https://packages.kamiwaza.ai/ubuntu/ noble main" | sudo tee
/etc/apt/sources.list.d/kamiwaza.list
```

2. Import and install Kamiwaza GPG signing key

```
curl -fsSL https://packages.kamiwaza.ai/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/kamiwaza-archive-keyring.gpg
```

3. Update package database and install Kamiwaza

```
sudo apt update
sudo apt upgrade
sudo apt install kamiwaza
```

4. Verify service starts (see Quickstart)

RHEL .rpm Package Installation (for RHEL 9)

For offline/air-gapped RHEL installations, see the comprehensive Red Hat Offline Installation Guide.

Other Linux Distros via Tarball

1. Follow the consolidated guide: Linux/macOS tarball installation
2. Ensure Docker Engine (with Compose v2), Python 3.10, and Node.js 22 are available (installer may configure as needed)
3. Run `install.sh --community`
4. Access via browser at `https://localhost`

Community Edition on macOS

Only Community Edition is supported on macOS.

1. Follow the consolidated guide: Linux/macOS tarball installation
2. Ensure Docker Engine (with Compose v2), Python 3.10, and Node.js 22 are available (installer may configure as needed)
3. Run `install.sh --community`
4. Access via browser at `https://localhost`

Community Edition on Windows

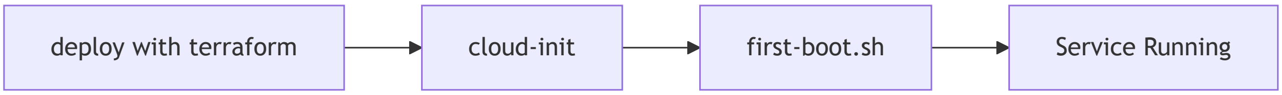
Use the MSI installer for a streamlined WSL2-based setup. See the Windows Installation Guide for prerequisites, GPU support, and step-by-step instructions.

Steps:

1. Download: `KamiwazaInstaller-[version]-[arch].msi`
2. Install: Run the MSI (reboot when prompted)
3. Launch: Start Menu → "Kamiwaza Start"

Enterprise Edition Deployment

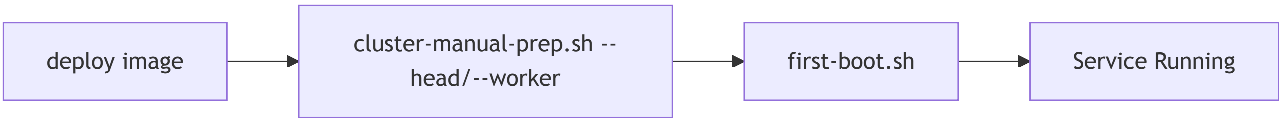
A. Terraform Deployment (Recommended)



Key Points:

- Terraform handles complete cluster setup
- cloud-init automatically runs first-boot.sh
- Service starts automatically via systemd

B. Manual Cluster Deployment



Key Points:

- Requires manual cluster setup via cluster-manual-prep.sh
- Must specify correct role (`--head` or `--worker --head-ip=<IP>`)
- Service starts automatically via systemd

Updating Kamiwaza

Windows

- Download new MSI installer and run to update existing installation
- Restart if prompted for GPU changes

Linux/macOS

- Download new package
- Run installation script again
- Service will restart automatically

Uninstallation

Windows

- Windows Settings → Add or Remove Programs -> (three dots on side) Uninstall

Linux/macOS

- Remove package via package manager
- Clean up any remaining configuration files

Windows GPU Setup Guide

Overview

Kamiwaza supports hardware acceleration on Windows through WSL2 with the following GPU configurations:

- **NVIDIA GPUs** (RTX series, GTX series, Quadro series)
- **Intel Arc GPUs** (A3xx, A5xx, A7xx series)
- **Intel Integrated GPUs** (UHD Graphics, Iris Xe)

Prerequisites

System Requirements

- Windows 11 (Build 22000 or later)
- WSL2 enabled and updated
- Latest GPU drivers installed
- Compatible GPU hardware

WSL2 Requirements

- WSL2 kernel version 5.10.60.1 or later
- Windows 11 with GPU virtualization support
- GPU drivers with WSL2 compatibility

NVIDIA GPU Setup

Supported Hardware

- **RTX 40 Series**: RTX 4090, RTX 4080, RTX 4070 Ti, RTX 4070, RTX 4060 Ti, RTX 4060
- **RTX 30 Series**: RTX 3090, RTX 3080, RTX 3070, RTX 3060 Ti, RTX 3060
- **RTX 20 Series**: RTX 2080 Ti, RTX 2080, RTX 2070, RTX 2060

- **GTX 16 Series**: GTX 1660 Ti, GTX 1660, GTX 1650
- **GTX 10 Series**: GTX 1080 Ti, GTX 1080, GTX 1070, GTX 1060

Driver Requirements

- **Minimum**: NVIDIA Driver 470.82 or later
- **Recommended**: NVIDIA Driver 535.98 or later
- **Latest**: Download from NVIDIA Driver Downloads (https://www.nvidia.com/Download/index.aspx)

Installation Steps

1. Install NVIDIA Drivers

1. Download the latest driver for your GPU
2. Run the installer as Administrator
3. Restart your computer
4. Verify installation: `nvidia-smi` in Command Prompt

2. Install NVIDIA CUDA Toolkit for WSL

```
# In WSL (Ubuntu 24.04)
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64/cuda-keyring_1.1-1_all.deb
sudo dpkg -i cuda-keyring_1.1-1_all.deb
sudo apt-get update
sudo apt-get -y install cuda-toolkit-12-4
```

3. Verify GPU Access in WSL

```
# Check if GPU is visible
nvidia-smi

# Expected output:
# +-----------------------------------------------------------------------------+
# | NVIDIA-SMI 535.98                 Driver Version: 535.98                    |
# |-------------------------------+----------------------+----------------------+
# | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
# | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
# |                               |                      |               MIG M. |
# |===============================+======================+======================|
# |   0  NVIDIA GeForce RTX 4090  On   | 00000000:01:00.0  Off |
N/A |
# |  0%    45C    P8    25W /  450W |      0MiB /  24576MiB |      0%      Default
|
# |                               |                      |                  N/A |
# +-------------------------------+----------------------+----------------------+
```

Configuration Files

.wslconfig (Windows)

```
[wsl2]
gpuSupport=true
memory=16GB
processors=8
```

Environment Variables (WSL)

```
# Add to ~/.bashrc
export CUDA_HOME=/usr/local/cuda
export PATH=$PATH:$CUDA_HOME/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CUDA_HOME/lib64
```

Intel Arc GPU Setup

Supported Hardware

- **Arc A7 Series**: A770, A750
- **Arc A5 Series**: A580, A570, A560, A550
- **Arc A3 Series**: A380, A370, A350, A310

Driver Requirements

- **Minimum**: Intel Arc Driver 31.0.101.4502 or later
- **Recommended**: Latest Intel Arc Driver
- **Download**: Intel Arc Driver Downloads (https://www.intel.com/content/www/us/en/download/785597/intel-arc-iris-xe-graphics-whql-windows.html)

Installation Steps

1. Install Intel Arc Drivers

1. Download the latest Intel Arc driver
2. Run the installer as Administrator
3. Restart your computer
4. Verify installation in Device Manager

2. Install Intel OpenCL Runtime and oneAPI (Recommended)

For optimal Intel GPU performance, install Intel's oneAPI toolkit:

```
# In WSL (Ubuntu 24.04)
# Add Intel's GPG key
wget -O- https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
PRODUCTS.PUB | \
  gpg --dearmor | sudo tee /usr/share/keyrings/oneapi-keyring.gpg > /dev/null

# Add the oneAPI repository
echo "deb [signed-by=/usr/share/keyrings/oneapi-keyring.gpg]
https://apt.repos.intel.com/oneapi all main" | \
  sudo tee /etc/apt/sources.list.d/oneAPI.list

# Update and install Intel OpenCL runtime and oneAPI
sudo apt update
sudo apt install -y intel-opencl-icd intel-basekit

# Configure permissions
sudo usermod -a -G render $USER
newgrp render
```

3. Alternative: Install OpenCL Runtime Only

If you prefer not to install the full oneAPI toolkit:

```
# Install OpenCL loader and tools
sudo apt-get update
sudo apt-get install -y ocl-icd-libopencl1 ocl-icd-opencl-dev opencl-headers
clinfo

# Add Intel Graphics PPA for latest drivers
sudo apt-get install -y software-properties-common
sudo add-apt-repository -y ppa:kobuk-team/intel-graphics
sudo apt-get update
sudo apt-get install -y libze-intel-gpu1 libze1 intel-opencl-icd
```

4. Verify GPU Access in WSL

```
# Check OpenCL availability
clinfo | grep "Platform Name"

# Check GPU devices
clinfo | grep "Device Name"

# Expected output:
# Platform Name                                 Intel(R) OpenCL
# Device Name                                   Intel(R) Arc(TM) A770 Graphics
```

Configuration Files

.wslconfig (Windows)

```
[wsl2]
gpuSupport=true
memory=16GB
processors=8
```

Environment Variables (WSL)

```
# Add to ~/.bashrc
export INTEL_OPENCL_CONFIG=/etc/OpenCL/vendors/intel.icd

# For oneAPI users, source the environment
echo 'source /opt/intel/oneapi/setvars.sh' >> ~/.bashrc
```

Intel Integrated GPU Setup

Supported Hardware

- **12th Gen Intel**: UHD Graphics 730, UHD Graphics 770
- **13th Gen Intel**: UHD Graphics 770, UHD Graphics 730
- **14th Gen Intel**: UHD Graphics 770, UHD Graphics 730
- **Intel Iris Xe**: Integrated graphics in 11th-14th gen processors

Driver Requirements

- **Minimum**: Intel Graphics Driver 30.0.101.1190 or later
- **Recommended**: Latest Intel Graphics Driver
- **Download**: Intel Graphics Driver Downloads (https://www.intel.com/content/www/us/en/download/785597/intel-arc-iris-xe-graphics-whql-windows.html)

Installation Steps

1. Install Intel Graphics Drivers

1. Download the latest Intel Graphics driver
2. Run the installer as Administrator
3. Restart your computer
4. Verify installation in Device Manager

2. Install Intel OpenCL Runtime

```
# In WSL (Ubuntu 24.04)
sudo apt-get update
sudo apt-get install -y intel-opencl-icd
```

3. Verify GPU Access in WSL

```
# Check OpenCL availability
clinfo | grep "Platform Name"

# Check GPU devices
clinfo | grep "Device Name"

# Expected output:
# Platform Name                                Intel(R) OpenCL
# Device Name                                  Intel(R) UHD Graphics 770
```

GPU Detection Scripts

Automatic Detection (PowerShell)

```
# detect_gpu.ps1
$gpuInfo = Get-WmiObject -Class Win32_VideoController | Select-Object Name,
AdapterRAM, DriverVersion

foreach ($gpu in $gpuInfo) {
    if ($gpu.Name -match "NVIDIA") {
        Write-Host "NVIDIA GPU detected: $($gpu.Name)"
        # Run NVIDIA setup
    }
    elseif ($gpu.Name -match "Intel.*Arc") {
        Write-Host "Intel Arc GPU detected: $($gpu.Name)"
        # Run Intel Arc setup
    }
    elseif ($gpu.Name -match "Intel.*UHD|Intel.*Iris") {
        Write-Host "Intel Integrated GPU detected: $($gpu.Name)"
        # Run Intel Integrated setup
    }
}
```

GPU Setup Scripts

NVIDIA Setup (setup_nvidia_gpu.sh)

```
#!/bin/bash
# setup_nvidia_gpu.sh

echo "Setting up NVIDIA GPU acceleration..."

# Install CUDA toolkit
sudo apt-get update
sudo apt-get install -y cuda-toolkit-12-4

# Configure environment
echo 'export CUDA_HOME=/usr/local/cuda' >> ~/.bashrc
echo 'export PATH=$PATH:$CUDA_HOME/bin' >> ~/.bashrc
echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CUDA_HOME/lib64' >> ~/.bashrc

# Test GPU access
nvidia-smi

echo "NVIDIA GPU setup complete!"
```

Intel Arc Setup (setup_intel_arc_gpu.sh)

```bash
#!/bin/bash
# setup_intel_arc_gpu.sh

echo "Setting up Intel Arc GPU acceleration..."

# Install oneAPI for optimal performance
wget -O- https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB | \
  gpg --dearmor | sudo tee /usr/share/keyrings/oneapi-keyring.gpg > /dev/null

echo "deb [signed-by=/usr/share/keyrings/oneapi-keyring.gpg] https://apt.repos.intel.com/oneapi all main" | \
  sudo tee /etc/apt/sources.list.d/oneAPI.list

sudo apt update
sudo apt install -y intel-opencl-icd intel-basekit

# Configure permissions
sudo usermod -a -G render $USER
newgrp render

# Configure environment
echo 'source /opt/intel/oneapi/setvars.sh' >> ~/.bashrc
echo 'export INTEL_OPENCL_CONFIG=/etc/OpenCL/vendors/intel.icd' >> ~/.bashrc

# Test GPU access
clinfo | grep "Device Name"

echo "Intel Arc GPU setup complete!"
```

Intel Integrated Setup (setup_intel_integrated_gpu.sh)

```
#!/bin/bash
# setup_intel_integrated_gpu.sh

echo "Setting up Intel Integrated GPU acceleration..."

# Install OpenCL runtime
sudo apt-get update
sudo apt-get install -y intel-opencl-icd

# Configure environment
echo 'export INTEL_OPENCL_CONFIG=/etc/OpenCL/vendors/intel.icd' >> ~/.bashrc

# Test GPU access
clinfo | grep "Device Name"

echo "Intel Integrated GPU setup complete!"
```

Advanced Intel GPU Setup for AI Workloads

Building llama.cpp with Intel GPU Support

For optimal Intel GPU performance with AI models, build llama.cpp with SYCL support:

```
# Install build dependencies
sudo apt-get install -y build-essential cmake libcurl4-openssl-dev

# Clone llama.cpp
git clone https://github.com/ggerganov/llama.cpp.git
cd llama.cpp

# Source oneAPI environment (required for SYCL build)
source /opt/intel/oneapi/setvars.sh

# Build with SYCL support
rm -rf build
mkdir -p build && cd build
cmake .. -DGGML_SYCL=ON -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx
make -j$(nproc)
```

Testing Intel GPU Acceleration

```
# Download a sample model
mkdir -p ../models
cd ../models
wget https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct-GGUF/resolve/main/qwen2.5-0.5b-instruct-q8_0.gguf

# Test inference with GPU offloading
cd ../build
source /opt/intel/oneapi/setvars.sh
./bin/llama-cli \
    -m ../models/qwen2.5-0.5b-instruct-q8_0.gguf \
    -p "Hello, how are you?" \
    -n 128 \
    -ngl 999  # Offload all layers to GPU
```

Troubleshooting

Common GPU Issues

GPU Not Detected in WSL

```
# Check WSL version
wsl --list --verbose

# Ensure WSL2 is being used
wsl --set-version Ubuntu-24.04 2

# Check GPU support
wsl --status
```

Driver Compatibility Issues

1. **Update Windows**: Ensure Windows 11 is fully updated
2. **Update WSL**: `wsl --update`
3. **Reinstall drivers**: Remove and reinstall GPU drivers
4. **Check compatibility**: Verify GPU supports WSL2 virtualization

Intel GPU Specific Issues

```
# Check OpenCL installation
clinfo

# Verify oneAPI environment (if installed)
source /opt/intel/oneapi/setvars.sh
sycl-ls

# Check permissions
groups $USER
# Should show 'render' in the list
```

Performance Issues

1. **Memory allocation**: Increase WSL memory in .wslconfig
2. **Processor allocation**: Allocate more CPU cores
3. **GPU memory**: Ensure sufficient GPU VRAM
4. **Background processes**: Close unnecessary applications

GPU Status Verification

NVIDIA GPU

```
# Check GPU status
nvidia-smi

# Check CUDA availability
nvcc --version

# Test CUDA functionality
cuda-install-samples-12.4.sh ~
cd ~/NVIDIA_CUDA-12.4_Samples/1_Utilities/deviceQuery
make
./deviceQuery
```

Intel GPU

```
# Check OpenCL availability
clinfo

# Check GPU information
lspci | grep -i vga

# Test OpenCL functionality
sudo apt-get install -y ocl-icd-opencl-dev
```

Performance Optimization

WSL Configuration (.wslconfig)

```
[wsl2]
gpuSupport=true
memory=32GB
processors=16
swap=8GB
localhostForwarding=true
```

Environment Optimization

```
# Add to ~/.bashrc
export CUDA_CACHE_DISABLE=0
export CUDA_CACHE_MAXSIZE=1073741824
export INTEL_OPENCL_CONFIG=/etc/OpenCL/vendors/intel.icd

# For oneAPI users
echo 'source /opt/intel/oneapi/setvars.sh' >> ~/.bashrc
```

GPU Memory Management

- **NVIDIA**: Use `nvidia-smi` to monitor GPU memory usage
- **Intel**: Monitor through Windows Task Manager
- **Optimization**: Close unnecessary GPU applications

Support and Resources

Official Documentation

- NVIDIA CUDA Documentation (https://docs.nvidia.com/cuda/)
- Intel OpenCL Documentation (https://www.intel.com/content/www/us/en/developer/tools/opencl/overview.html)
- Intel oneAPI Documentation (https://www.intel.com/content/www/us/en/developer/tools/oneapi/overview.html)
- Microsoft WSL GPU Support (https://docs.microsoft.com/en-us/windows/wsl/tutorials/gpu-compute)

Community Resources

- NVIDIA Developer Forums (https://forums.developer.nvidia.com/)
- Intel Community Forums (https://community.intel.com/)
- WSL GitHub Issues (https://github.com/microsoft/WSL/issues)

Troubleshooting Tools

- **GPU-Z**: Detailed GPU information and monitoring
- **MSI Afterburner**: GPU monitoring and overclocking
- **HWiNFO**: Comprehensive system information
- **Windows Performance Monitor**: System performance analysis

---

Last Updated: October 3rd, 2025 **Version**: Compatible with Kamiwaza v0.5.1 **Support**: Contact our support team

Quickstart

Get up and running with Kamiwaza in just a few minutes! This guide will walk you through starting the platform, deploying your first AI model, and launching a real application from the App Garden.

Prerequisites

Before you begin, make sure you have:

- Kamiwaza installed and configured (see our Installation Guide)
- At least 16GB of available RAM
- A stable internet connection for downloading models

Step 1: Start Kamiwaza

First, let's get the Kamiwaza platform running on your system.

For Community Edition (Ubuntu .deb package)

If you installed via the .deb package, Kamiwaza should start automatically as a system service. You can check the status with:

```
kamiwaza status
```

If it's not running, start it with:

```
kamiwaza start
```

For Manual Installations

Navigate to your Kamiwaza installation directory and start the platform:

```
cd /path/to/kamiwaza
bash startup/kamiwazad.sh start
```

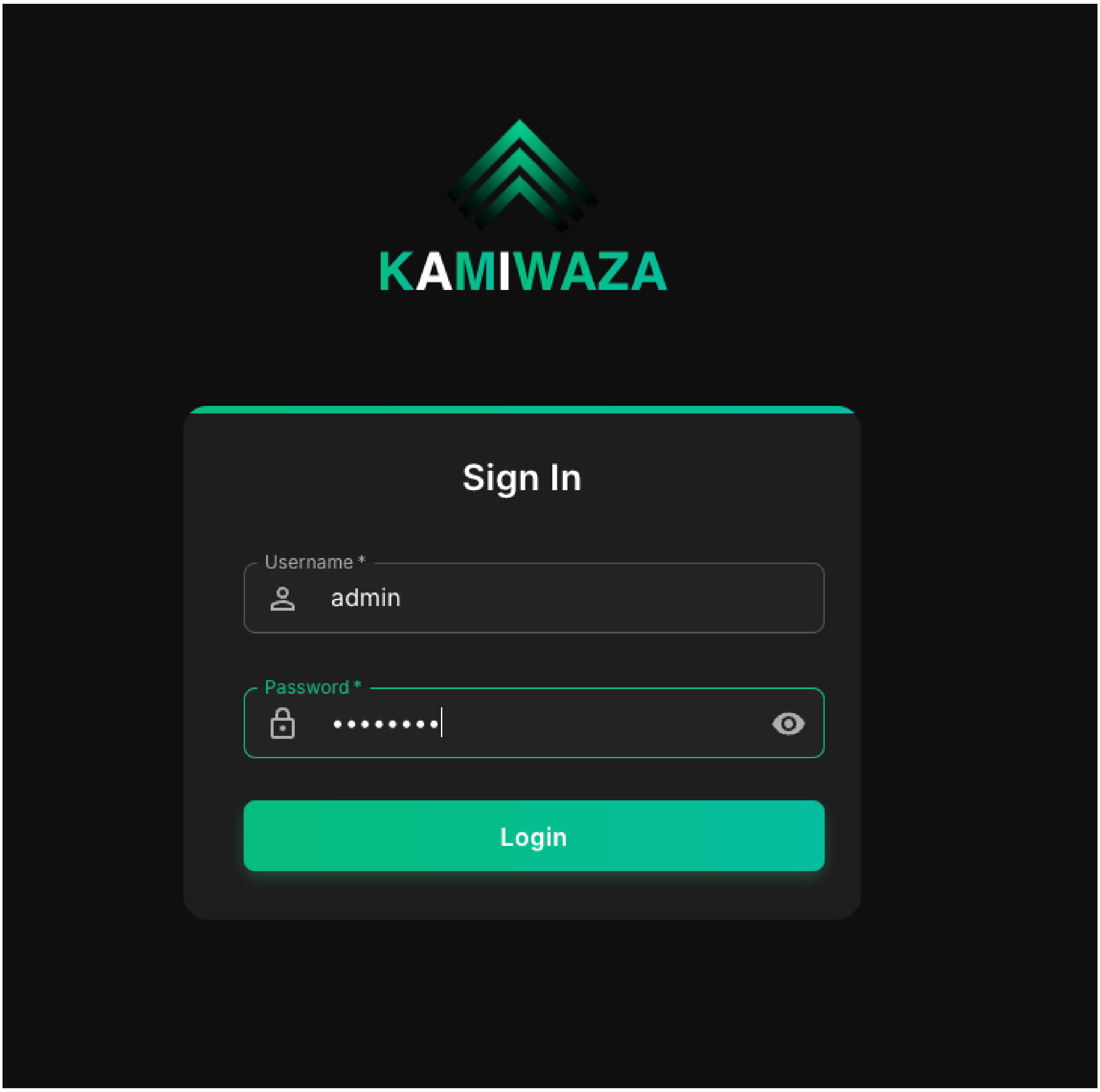
Verify Kamiwaza is Running

Open your web browser and navigate to:

- **Frontend**: https://localhost (https://localhost)
- **API Documentation**: http://localhost/api/docs (http://localhost/api/docs)

You should see the Kamiwaza interface load successfully. Use the following credentials in the Sign In screen:

- **Username:** `admin`
- **Password:** `kamiwaza`



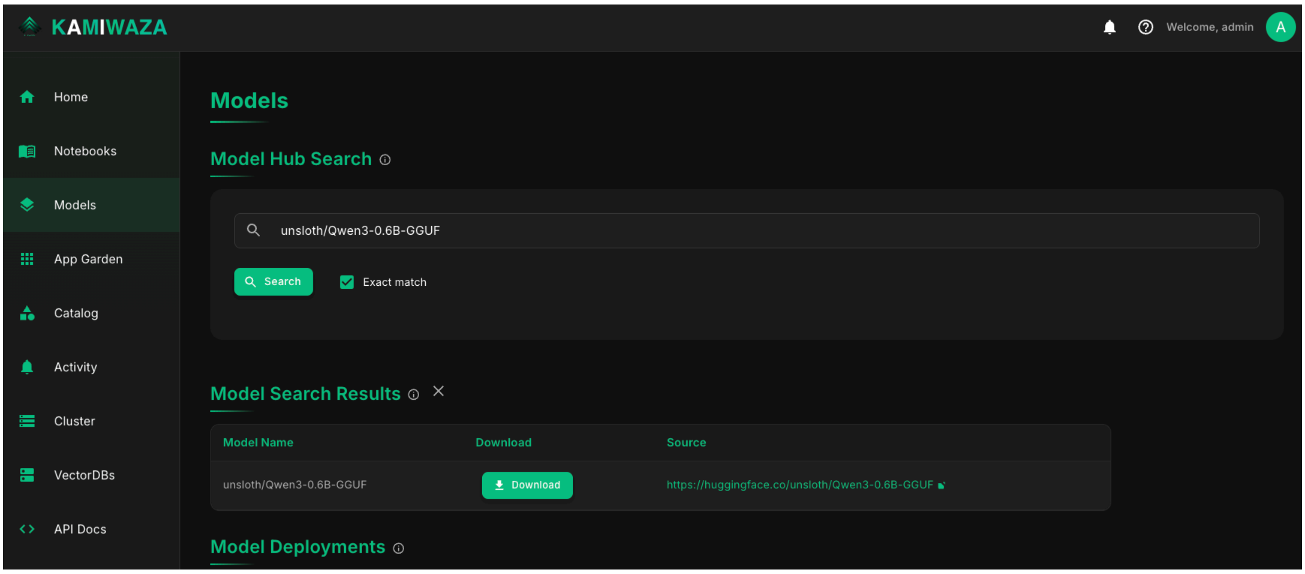
Step 2: Deploy Your First Model

Now let's deploy a small, fast language model that's perfect for getting started.

Access the Models Section

1. In the Kamiwaza frontend, navigate to the **Models** section
2. Enter `unsloth/Qwen3-0.6B-GGUF` into the search field, click `Exact match`, then click Search.
   - Click the "Download" button, unselect all the files, just select the box next to `Qwen3-0.6B-Q6_K.gguf - 495.11 MB`

- Click "Download Selected Files".
- You should now see the file downloading in a new pop-up modal.

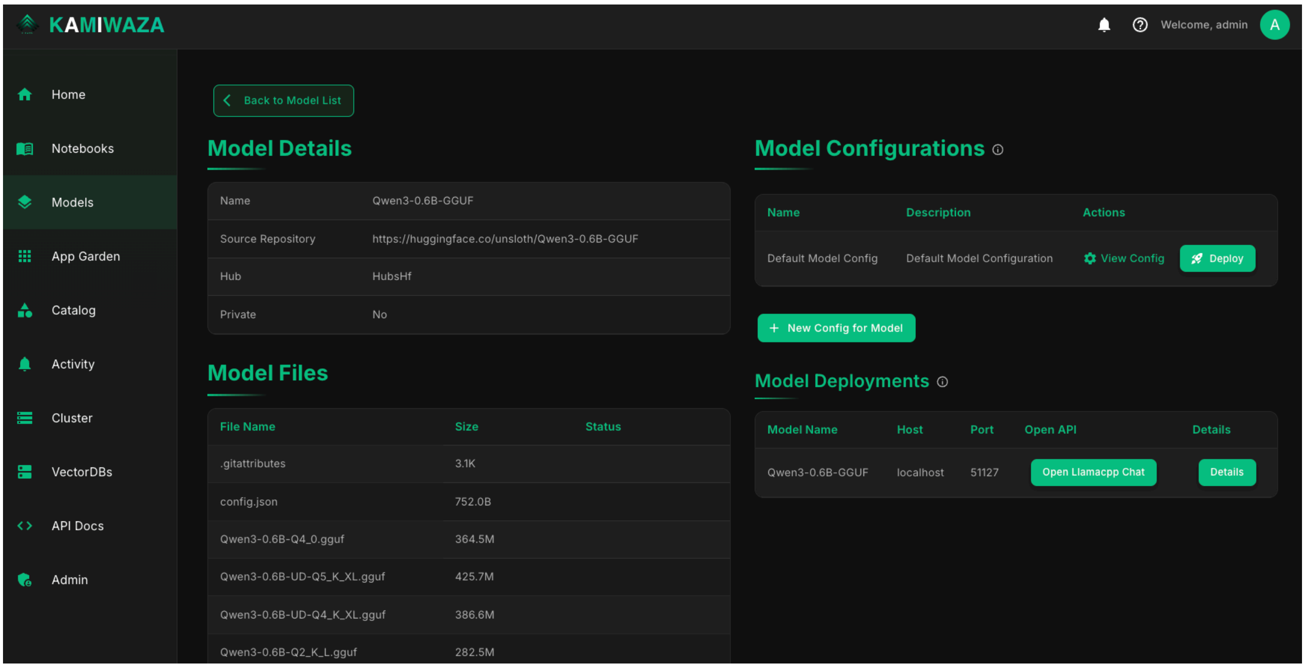


unsloth/Qwen3-0.6B-GGUF

☑ Exact match

Model Name	Download	Source
unsloth/Qwen3-0.6B-GGUF	Download	https://huggingface.co/unsloth/Qwen3-0.6B-GGUF

3. Deploy the model.
   - Scroll down to the "Your Models" section, and click the name of the model (`Qwen3-0.6B-GGUF`)
   - On the next screen, look for the "Deploy" button in the upper right (under Model Configurations) and click it.
   - After a few moments, you will see the model appear under "Model Deployments."



Name	Qwen3-0.6B-GGUF
Source Repository	https://huggingface.co/unsloth/Qwen3-0.6B-GGUF
Hub	HubsHf
Private	No

Name	Description	Actions
Default Model Config	Default Model Configuration	View Config Deploy

File Name	Size	Status
.gitattributes	3.1K	
config.json	752.0B	
Qwen3-0.6B-Q4_0.gguf	364.5M	
Qwen3-0.6B-UD-Q5_K_XL.gguf	425.7M	
Qwen3-0.6B-UD-Q4_K_XL.gguf	386.6M	
Qwen3-0.6B-Q2_K_L.gguf	282.5M	

Model Name	Host	Port	Open API	Details
Qwen3-0.6B-GGUF	localhost	51127	Open Llamacpp Chat	Details

Review

For the first deployment, we deployed a modern large language model:

- **Model**: `Qwen3 0.6B` (very light wieght, suitable for basic conversation)
    - Note that this is a GGUF-quantized version of the model suitable for CPU-based inference
- **Engine**: `llamacpp` (CPU-friendly)

Step 3: Launch an AI Chatbot from the App Garden

Now let's use your deployed model in a real application from the App Garden.

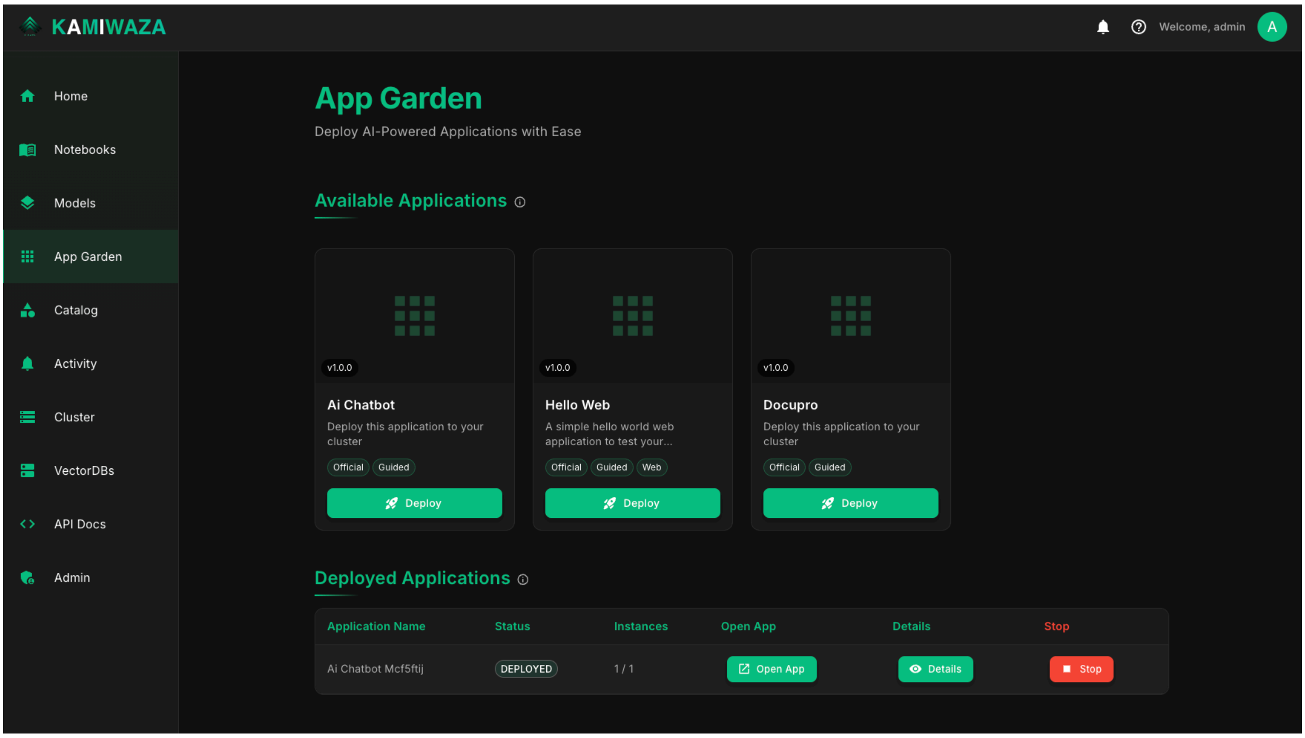
Access the App Garden

1. Navigate to the **App Garden** section in the Kamiwaza interface
2. Browse the available applications
3. Look for the **"AI Chatbot"** app

[Screenshot placeholder: App Garden interface showing available apps]

Configure and Launch the Chatbot

1. Click on the **AI Chatbot** app's "Deploy" button
2. In the configuration screen, keep the default configuration, and click **"Deploy"**
3. Click **"Launch App"**



App Garden

Deploy AI-Powered Applications with Ease

Available Applications

Ai Chatbot
Deploy this application to your cluster

Hello Web
A simple hello world web application to test your...

Docupro
Deploy this application to your cluster

Deployed Applications

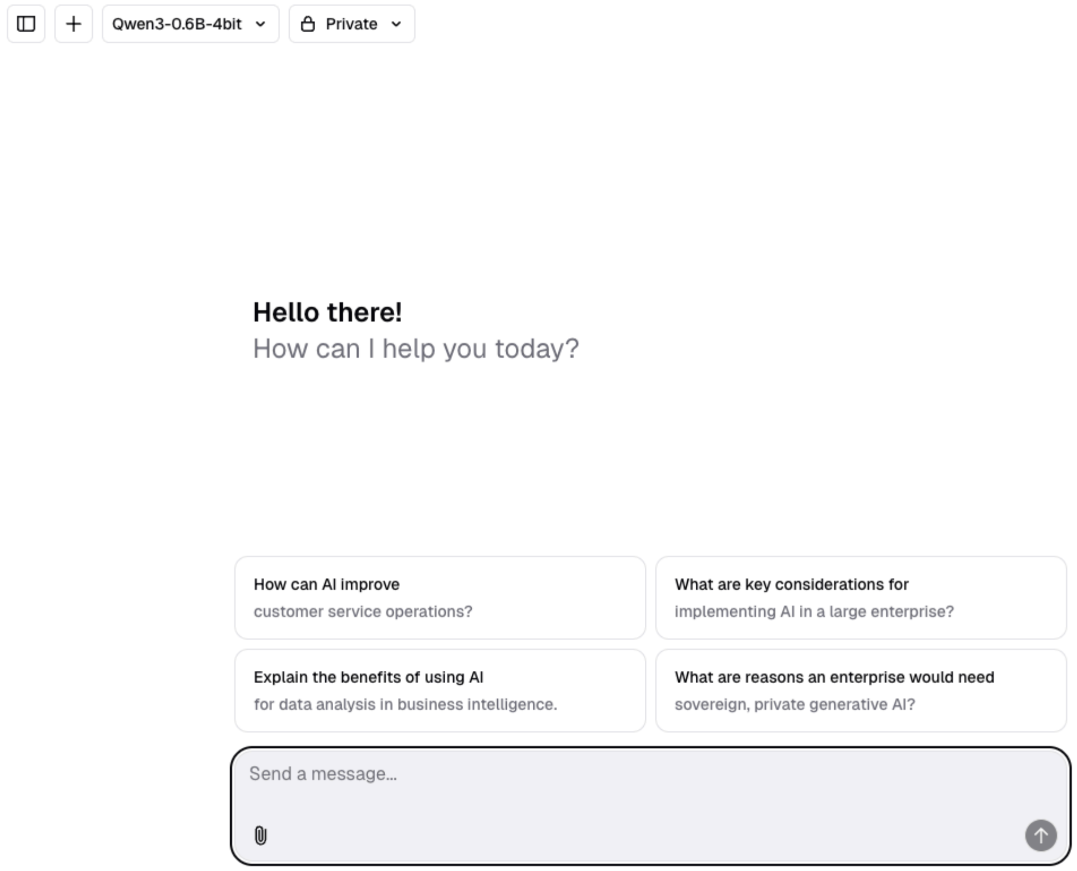
Application Name	Status	Instances	Open App	Details	Stop
Ai Chatbot Mcf5ftij	DEPLOYED	1 / 1	Open App	Details	Stop

Access Your Running Chatbot

- Once launched, click the **"Open App"** button in the deployed app listing,
- This will open the app in a new browser tab.
- It will automatically use your deployed model.

Try Your Chatbot

You now have a fully functional AI-powered chat application!



Hello there!
How can I help you today?

How can AI improve customer service operations?

What are key considerations for implementing AI in a large enterprise?

Explain the benefits of using AI for data analysis in business intelligence.

What are reasons an enterprise would need sovereign, private generative AI?

Send a message...

Next Steps

Congratulations! You've successfully: ✅ Started Kamiwaza
✅ Deployed your first AI model
✅ Launched a real application from the App Garden

Explore More

Now that you have the basics down, here are some next steps to explore:

- **Try Different Models**: Deploy larger, more capable models
- **Explore More Apps**: Check out other applications in the App Garden
- **Learn the Architecture**: Understand how Kamiwaza works under the hood
- **Use the SDK**: Build custom applications using the Kamiwaza Python SDK
- **Set Up Your Data**: Connect your own data sources for RAG applications

Need Help?

If you run into any issues:

- Check the troubleshooting section for common problems
- Join our Discord community (https://discord.gg/cVGBS5rD2U) for real-time help
- Contact our support team (https://portal.kamiwaza.ai/_hcms/mem/login?redirect_url=https%3A%2F%2Fportal.kamiwaza.ai%2Ftickets-view)

What You've Learned

In this quickstart, you've experienced the core Kamiwaza workflow:

1. **Model Management**: How to deploy and test AI models
2. **App Garden**: How to launch pre-built applications
3. **Integration**: How models and apps work together seamlessly

This same pattern scales from simple chatbots to complex enterprise AI applications. Welcome to Kamiwaza!

Release Notes