PyTaskforce Enterprise Features Guide

Version: 1.0 **Last Updated:** January 2026 **Target Audience:** Technical Decision-Makers, Architects, Business Stakeholders

Table of Contents

1. Executive Summary

1.1 What is PyTaskforce Enterprise?

PyTaskforce Enterprise is a production-ready, multi-agent orchestration framework designed for organizations that need to deploy AI agents at scale with enterprise-grade security, compliance, and operational controls.

Built on Clean Architecture principles, PyTaskforce enables autonomous AI agents that:

- **Plan and execute** complex tasks through LLM-driven reasoning (ReAct loop)
- **Decompose work** into manageable steps via TodoList patterns
- **Leverage tools** through an extensible execution framework
- **Maintain context** via session state persistence and long-term memory

1.2 Key Benefits at a Glance

Benefit	Description
Multi-Tenant Isolation	Complete data separation between organizations with tenant-scoped sessions
Enterprise Security	JWT/OAuth2 authentication, API key management, role-based access control
Compliance Ready	Built-in support for SOC 2, ISO 27001, and GDPR requirements
Operational Visibility	Usage tracking, cost reporting, SLA monitoring with Prometheus export

Benefit	Description
Governed Agent Lifecycle	Version control, approval workflows, and audit trails for all agents
Flexible Deployment	CLI for development, REST API for production, Docker/Kubernetes ready

1.3 Target Audiences and Use Cases

For CTOs and Architects:

- Multi-tenant SaaS platforms integrating AI agents
- Internal knowledge assistants with department-level isolation
- Customer service automation requiring compliance certification
- Regulated industries (finance, healthcare, legal) with strict audit requirements

For Business Stakeholders:

- Reduce operational costs through AI automation
- Maintain compliance posture without sacrificing innovation
- Gain visibility into AI usage and spending across the organization
- Protect sensitive data while enabling AI-powered workflows

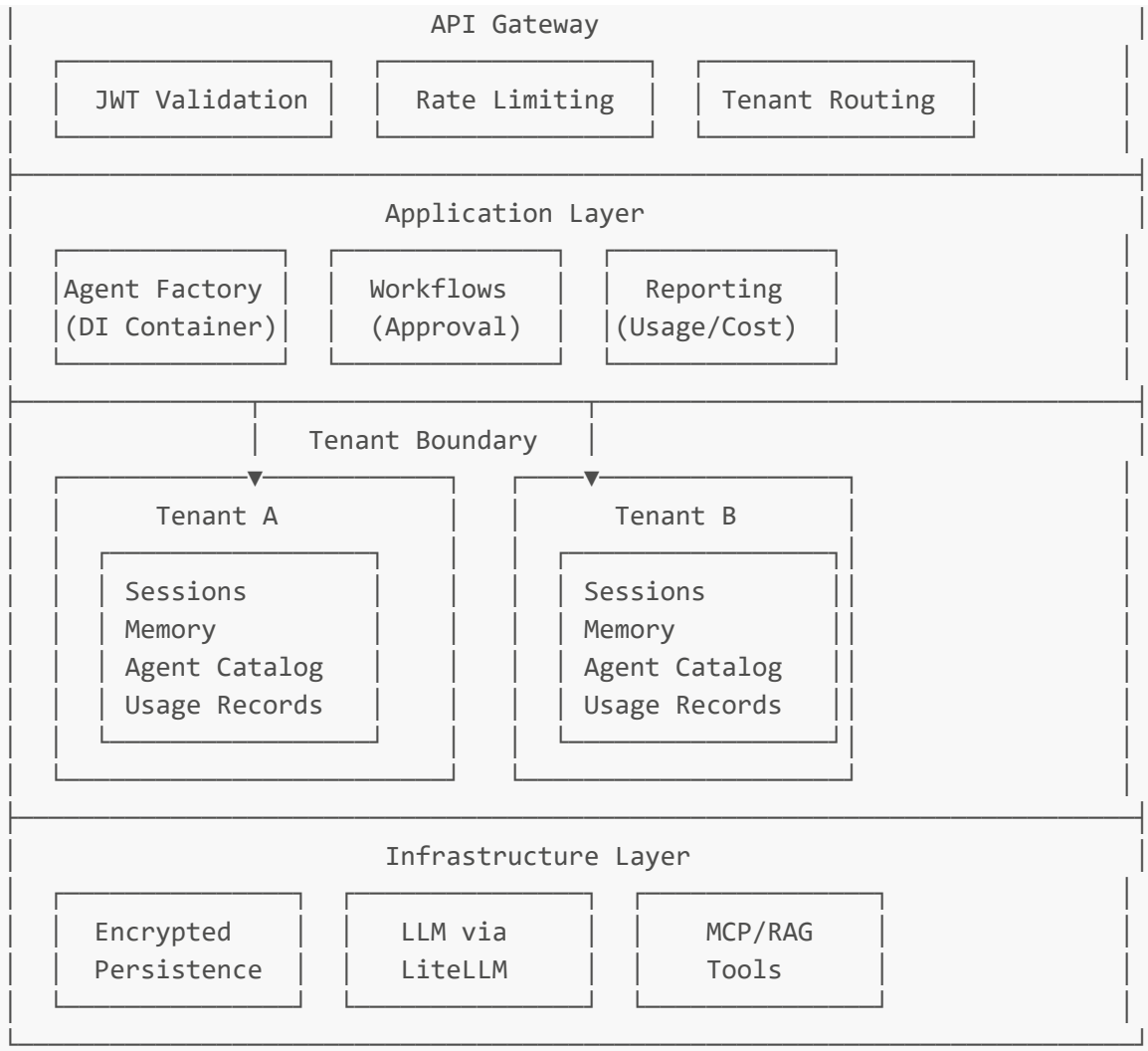
1.4 Quick Decision Matrix

Requirement	PyTaskforce Enterprise
Multi-tenant deployment	Yes - Complete tenant isolation
SSO/OAuth2 integration	Yes - JWT and API key support
SOC 2 / ISO 27001 compliance	Yes - Built-in evidence collection
GDPR data processing records	Yes - Automatic Article 30 reports
Usage-based billing support	Yes - Per-tenant cost tracking
Agent version control	Yes - Semantic versioning with approval workflows
Data encryption at rest	Yes - AES-256 with per-tenant keys
Prometheus/Grafana integration	Yes - Native metrics export

2. Architecture Overview

2.1 Multi-Tenant Architecture

PyTaskforce implements a shared-nothing approach to tenant isolation, ensuring complete data separation:



2.2 Security Model (Defense in Depth)

PyTaskforce implements multiple security layers:

```
Layer 1: Network Security
├── TLS encryption in transit
├── IP allowlisting (optional)
└── Rate limiting per tenant

Layer 2: Authentication
├── JWT/OAuth2 token validation
├── API key authentication
└── Token expiration and refresh

Layer 3: Authorization
├── Role-based access control (RBAC)
├── Resource-level permissions
└── Policy engine enforcement

Layer 4: Data Protection
├── Encryption at rest (AES-256)
├── Per-tenant encryption keys
```

```
└── Memory access control lists (ACLs)

Layer 5: Audit & Compliance
├── Structured audit logging
├── Evidence chain tracking
└── Compliance report generation
```

2.3 Deployment Options

Option	Description	Best For
On-Premise	Full deployment within your infrastructure	Maximum data control, air-gapped environments
Cloud (AWS/Azure/GCP)	Containerized deployment with managed services	Scalability, managed infrastructure
Hybrid	Core on-premise, LLM calls to cloud providers	Balance of control and capability

3. Identity & Access Management

3.1 Multi-Tenant Isolation

Every operation in PyTaskforce carries tenant and user context, ensuring complete isolation between organizations.

Tenant Context

```python
from taskforce.core.interfaces.identity import TenantContext

# Tenant represents an organization or customer
tenant = TenantContext(
    tenant_id="acme-corp",
    name="Acme Corporation",
    settings={
        "max_agents": 100,
        "max_concurrent_sessions": 50,
        "features": ["rag", "long_term_memory", "approval_workflows"],
        "data_retention_days": 90
    },
    metadata={
        "tier": "enterprise",
        "contract_start": "2025-01-01",
        "primary_contact": "admin@acme.com"
    }
)
```

User Context

```python
from taskforce.core.interfaces.identity import UserContext, Permission

# User represents an authenticated person or service account
user = UserContext(
    user_id="user-12345",
    tenant_id="acme-corp",
    username="alice.johnson",
    email="alice@acme.com",
    roles={"admin", "agent_designer"},
    permissions={
        Permission.AGENT_CREATE,
        Permission.AGENT_READ,
        Permission.AGENT_UPDATE,
        Permission.AGENT_EXECUTE,
        Permission.USER_MANAGE,
    },
    attributes={
        "department": "Engineering",
        "team": "AI Platform"
    }
)

# Check permissions programmatically
if user.has_permission(Permission.AGENT_CREATE):
    # User can create agents
    pass

if user.is_admin():
    # User has admin role
    pass
```

Session Namespacing

All sessions are namespaced by tenant to prevent data leakage:

```python
# Sessions are automatically prefixed with tenant_id
session_id = f"{tenant.tenant_id}:session-{uuid4()}"
# Result: "acme-corp:session-abc123..."

# State manager enforces tenant isolation
await state_manager.save_state(
    session_id="acme-corp:session-abc123",
    state_data={"mission": "Analyze Q4 sales data", "step": 1}
)

# Attempting to access another tenant's session will fail
await state_manager.load_state("other-tenant:session-xyz")  # Returns None
```

3.2 Authentication

JWT/OAuth2 Integration

PyTaskforce supports JWT tokens from any OAuth2/OIDC provider (Auth0, Okta, Azure AD, etc.).

```
from taskforce.infrastructure.auth.jwt_provider import (
    JWTIdentityProvider,
    JWTConfig,
    create_test_jwt  # For development/testing only
)

# Configure JWT validation
config = JWTConfig(
    secret_key="your-secret-key",           # For HS256
    # public_key="-----BEGIN PUBLIC...",    # For RS256
    algorithms=["HS256"],
    issuer="https://auth.yourcompany.com",
    audience="taskforce-api",
    leeway_seconds=30,                       # Clock skew tolerance

    # Claim mapping (customize for your IdP)
    tenant_claim="org_id",                   # Claim containing tenant ID
    roles_claim="roles",                     # Claim containing user roles
    username_claim="preferred_username",
    email_claim="email"
)

provider = JWTIdentityProvider(config)

# Validate incoming token
async def authenticate_request(token: str):
    user_context = await provider.validate_token(token)
    if user_context is None:
        raise UnauthorizedError("Invalid or expired token")
    return user_context
```

Creating Test Tokens (Development Only):

```
# Generate a test JWT for development
token = create_test_jwt(
    user_id="user-123",
    tenant_id="acme-corp",
    roles=["admin", "agent_designer"],
    secret_key="your-secret-key",
    expires_in_seconds=3600,
    # Additional claims
    email="alice@acme.com",
    department="Engineering"
```

```
)
# Use token in Authorization header: Bearer {token}
```

API Key Management

For service-to-service communication and programmatic access:

```python
from taskforce.infrastructure.auth.api_key_provider import APIKeyProvider
from datetime import datetime, timezone, timedelta

provider = APIKeyProvider()

# Create an API key for a service account
plaintext_key, record = provider.create_api_key(
    tenant_id="acme-corp",
    user_id="svc-data-pipeline",
    name="Data Pipeline Service Key",
    roles={"operator"},                    # Limited role
    expires_at=datetime.now(timezone.utc) + timedelta(days=365),
    metadata={
        "created_by": "admin@acme.com",
        "purpose": "ETL pipeline agent execution"
    }
)

print(f"API Key: {plaintext_key}")  # tf_Abc123... (store securely!)
print(f"Key ID: {record.key_id}")   # Used for management

# Validate an API key
async def authenticate_api_key(api_key: str):
    user_context = await provider.validate_api_key(api_key)
    if user_context is None:
        raise UnauthorizedError("Invalid API key")
    return user_context

# List keys for a tenant (for admin dashboard)
keys = provider.list_api_keys("acme-corp")
for key in keys:
    print(f"  {key.name}: expires {key.expires_at}, active={key.is_active}")

# Revoke a compromised key
provider.revoke_api_key(key_id="abc123")
```

SSO Integration Patterns

Azure AD Integration:

```
# configs/enterprise.yaml
authentication:
```

```
  type: jwt
  jwt:
    issuer: "https://login.microsoftonline.com/{tenant-id}/v2.0"
    audience: "api://{application-id}"
    algorithms: ["RS256"]
    jwks_uri: "https://login.microsoftonline.com/{tenant-id}/discovery/v2.0/keys"
    tenant_claim: "tid"
    roles_claim: "roles"
```

Okta Integration:

```
authentication:
  type: jwt
  jwt:
    issuer: "https://{your-domain}.okta.com"
    audience: "api://taskforce"
    algorithms: ["RS256"]
    jwks_uri: "https://{your-domain}.okta.com/oauth2/v1/keys"
    tenant_claim: "org_id"
    roles_claim: "groups"
```

3.3 Role-Based Access Control (RBAC)

Pre-defined System Roles

PyTaskforce includes five system roles covering common enterprise patterns:

Role	Description	Key Permissions
admin	Full tenant administration	All permissions within tenant scope
agent_designer	Create and manage agents	`agent:*`, `tool:read`, `session:read`
operator	Execute agents and manage sessions	`agent:execute/read`, `session:*`, `tool:*`, `memory:read/write`
auditor	Compliance and audit access	`*:read`, `audit:read`
viewer	Basic read-only access	`agent:read`, `session:read`, `tool:read`

Complete Permission Matrix

```
from taskforce.core.interfaces.identity import Permission, SYSTEM_ROLES

# All available permissions
class Permission(Enum):
    # Agent permissions
    AGENT_CREATE = "agent:create"
    AGENT_READ = "agent:read"
```

```
    AGENT_UPDATE = "agent:update"
    AGENT_DELETE = "agent:delete"
    AGENT_EXECUTE = "agent:execute"

    # Session permissions
    SESSION_CREATE = "session:create"
    SESSION_READ = "session:read"
    SESSION_DELETE = "session:delete"

    # Tool permissions
    TOOL_EXECUTE = "tool:execute"
    TOOL_READ = "tool:read"

    # Memory permissions
    MEMORY_READ = "memory:read"
    MEMORY_WRITE = "memory:write"
    MEMORY_DELETE = "memory:delete"

    # Admin permissions
    TENANT_MANAGE = "tenant:manage"
    USER_MANAGE = "user:manage"
    ROLE_MANAGE = "role:manage"
    POLICY_MANAGE = "policy:manage"
    AUDIT_READ = "audit:read"

    # System permissions
    SYSTEM_CONFIG = "system:config"
    SYSTEM_METRICS = "system:metrics"
```

Custom Role Creation

```
from taskforce.core.interfaces.identity import Role, Permission

# Create a custom role for your organization
data_analyst_role = Role(
    role_id="custom:data_analyst",
    name="Data Analyst",
    description="Can execute data analysis agents and view results",
    permissions={
        Permission.AGENT_READ,
        Permission.AGENT_EXECUTE,
        Permission.SESSION_CREATE,
        Permission.SESSION_READ,
        Permission.MEMORY_READ,
    },
    tenant_id="acme-corp",      # Tenant-specific role
    is_system_role=False,
    metadata={
        "created_by": "admin@acme.com",
        "department": "Analytics"
    }
```

```
)

# Register with role manager
await role_manager.create_role(
    name=data_analyst_role.name,
    description=data_analyst_role.description,
    permissions=data_analyst_role.permissions,
    tenant_id="acme-corp"
)
```

Policy Engine Usage

```
from taskforce.core.interfaces.identity import PolicyEngineProtocol,
PolicyDecision

# The policy engine evaluates access requests
async def check_agent_access(
    policy_engine: PolicyEngineProtocol,
    user: UserContext,
    agent_id: str,
    action: Permission
) -> bool:
    decision: PolicyDecision = await policy_engine.evaluate(
        user=user,
        action=action,
        resource_type=ResourceType.AGENT,
        resource_id=agent_id,
        context={"requested_at": datetime.now(timezone.utc).isoformat()}
    )

    if not decision.allowed:
        logger.warning(
            "access.denied",
            user_id=user.user_id,
            action=action.value,
            agent_id=agent_id,
            reason=decision.reason
        )

    return decision.allowed

# Using decorators for route protection (FastAPI example)
from functools import wraps

def require_permission(permission: Permission):
    def decorator(func):
        @wraps(func)
        async def wrapper(*args, user: UserContext, **kwargs):
            if not user.has_permission(permission):
                raise HTTPException(
                    status_code=403,
```

```
                    detail=f"Permission denied: {permission.value} required"
                )
            return await func(*args, user=user, **kwargs)
        return wrapper
    return decorator

@router.post("/agents")
@require_permission(Permission.AGENT_CREATE)
async def create_agent(request: CreateAgentRequest, user: UserContext):
    # Only users with AGENT_CREATE permission reach here
    pass
```

---

4. Agent Lifecycle Management

4.1 Agent Catalog

The Agent Catalog provides a centralized registry for managing agent definitions across your organization.

Version Management with Semantic Versioning

```
from taskforce.application.catalog import AgentCatalog, VersionStatus

catalog = AgentCatalog()

# Create a new agent in the catalog
entry = catalog.create_agent(
    tenant_id="acme-corp",
    name="Customer Support Agent",
    description="Handles customer inquiries via email and chat",
    category="support",
    tags={"production", "nlp", "customer-facing"},
    owner_id="alice",
    initial_definition={
        "tools": ["semantic_search", "send_email"],
        "system_prompt": "You are a helpful customer support agent for Acme
Corp.",
        "max_iterations": 10,
        "temperature": 0.7
    }
)

print(f"Agent created: {entry.name}")
print(f"Agent ID: {entry.agent_id}")
print(f"Initial version: {entry.versions[0].version_number}")  # 0.1.0 (DRAFT)
```

Creating New Versions

```
# Create version 1.0.0 with enhanced capabilities
new_version = catalog.create_version(
    agent_id=entry.agent_id,
    version_number="1.0.0",
    definition={
        "tools": ["semantic_search", "send_email", "create_ticket"],
        "system_prompt": """You are a helpful customer support agent for Acme
Corp.

When you cannot resolve an issue directly, create a support ticket for escalation.
Always cite sources when referencing documentation.""",
        "max_iterations": 15,
        "temperature": 0.5,
        "rag_config": {
            "index": "support-docs",
            "max_results": 5
        }
    },
    created_by="alice",
    changelog="Added ticket creation capability and improved RAG integration"
)

print(f"Version created: {new_version.version_number}")
print(f"Status: {new_version.status.value}")  # draft
print(f"Config hash: {new_version.config_hash}")  # Used for change detection
```

Categorization and Tagging

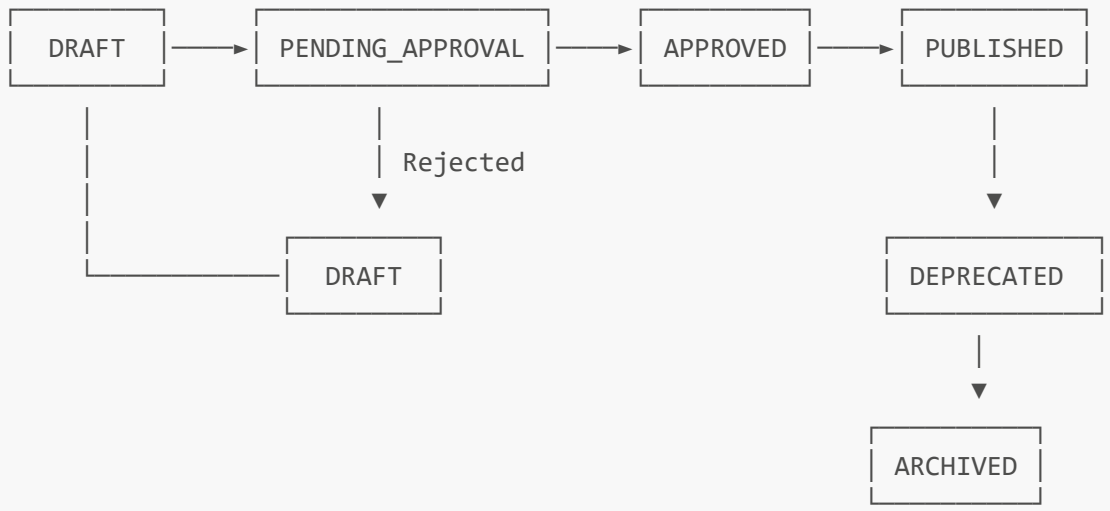
```
# Search agents by category
support_agents = catalog.list_agents(
    tenant_id="acme-corp",
    category="support"
)

# Search by tags (any match)
nlp_agents = catalog.list_agents(
    tenant_id="acme-corp",
    tags={"nlp", "natural-language"}
)

# Full-text search across name, description, and tags
results = catalog.search_agents(
    tenant_id="acme-corp",
    query="customer support",
    limit=10
)

for agent in results:
    current = agent.get_current_version()
    print(f"  {agent.name} v{current.version_number if current else 'N/A'}")
```

```
    print(f"    Category: {agent.category}")
    print(f"    Tags: {', '.join(agent.tags)}")
```

4.2 Approval Workflows

Agent versions follow a governed lifecycle requiring explicit approvals before production deployment.

Version Lifecycle States



Complete Workflow Example

```
from taskforce.application.workflows import WorkflowManager, ApprovalStatus
from taskforce.application.workflows.approval import RequestType, ApprovalWorkflow

# Initialize workflow manager
manager = WorkflowManager()

# (Optional) Register a custom workflow with stricter requirements
strict_workflow = ApprovalWorkflow(
    workflow_id="strict-publish",
    tenant_id="acme-corp",
    name="Strict Agent Publishing",
    description="Requires 2 approvers for production agents",
    request_type=RequestType.AGENT_PUBLISH,
    required_approvers=2,                          # Need 2 approvals
    allowed_approver_roles={"admin", "agent_designer"},
    auto_expire_hours=48,                          # Request expires in 48h
    notify_on_create=True,
    notify_on_action=True
)
manager.register_workflow(strict_workflow)

# Step 1: Submit version for approval
```

```
catalog.submit_for_approval(entry.agent_id, new_version.version_id)
print(f"Submitted: {new_version.status.value}")  # pending_approval

# Step 2: Create approval request
request = manager.create_request(
    tenant_id="acme-corp",
    request_type=RequestType.AGENT_PUBLISH,
    resource_type="agent",
    resource_id=entry.agent_id,
    requester_id="alice",
    title="Publish Customer Support Agent v1.0.0",
    description="Ready for production deployment after QA testing",
    payload={
        "version_id": new_version.version_id,
        "version_number": new_version.version_number,
        "changelog": new_version.changelog
    }
)

print(f"Request ID: {request.request_id[:8]}...")
print(f"Status: {request.status.value}")           # pending
print(f"Expires: {request.expires_at}")            # 48 hours from now
print(f"Required approvers: {request.required_approvers}")

# Step 3: Approver reviews pending requests
pending = manager.list_pending_for_approver(
    tenant_id="acme-corp",
    user_id="bob",
    user_roles={"admin", "agent_designer"}
)

print(f"Pending requests for Bob: {len(pending)}")
for req in pending:
    print(f"  - {req.title} (by {req.requester_id})")

# Step 4: First approval
result = manager.approve_request(
    request_id=request.request_id,
    user_id="bob",
    comment="Reviewed and tested - looks good!"
)
print(f"Approved by Bob: {result}")
print(f"Current approvers: {request.current_approvers}")  # {'bob'}
print(f"Status: {request.status.value}")  # still pending (need 2)

# Step 5: Second approval completes the request
result = manager.approve_request(
    request_id=request.request_id,
    user_id="carol",
    comment="LGTM - approved for production"
)
print(f"Approved by Carol: {result}")              # True
print(f"Status: {request.status.value}")           # approved
```

```
# Step 6: Complete the approval in the catalog
catalog.approve_version(entry.agent_id, new_version.version_id,
approved_by="carol")
catalog.publish_version(entry.agent_id, new_version.version_id)

print(f"Final status: {new_version.status.value}")  # published
print(f"Current version: {entry.current_version}")
```

Event Callbacks and Notifications

```
# Register callbacks for workflow events
def on_request_approved(request):
    """Called when a request is fully approved."""
    # Automatically publish the approved version
    if request.request_type == RequestType.AGENT_PUBLISH:
        version_id = request.payload.get("version_id")
        catalog.approve_version(request.resource_id, version_id,
approved_by="system")
        catalog.publish_version(request.resource_id, version_id)

    # Send notification
    notify_slack(
        channel="#agent-releases",
        message=f"Agent {request.resource_id} has been approved and published!"
    )

def on_request_rejected(request):
    """Called when a request is rejected."""
    notify_email(
        to=request.requester_id,
        subject=f"Request Rejected: {request.title}",
        body=f"Your request was rejected. Reason: {request.actions[-1].comment}"
    )

manager.on_event("request_approved", on_request_approved)
manager.on_event("request_rejected", on_request_rejected)
```

---

5. Secure Memory Governance

5.1 Memory Access Control

PyTaskforce provides fine-grained access control for all memory resources, including conversation history, tool results, and persistent knowledge.

Sensitivity Levels

```
from taskforce.core.domain.memory_acl import SensitivityLevel

# Four sensitivity levels in ascending order of restriction
class SensitivityLevel(Enum):
    PUBLIC = "public"         # Accessible to all authenticated users in tenant
    INTERNAL = "internal"     # Accessible to team members
    CONFIDENTIAL = "confidential"  # Restricted access, explicit grants required
    RESTRICTED = "restricted"      # Highly restricted, audit required
```

Memory Scopes

```
from taskforce.core.domain.memory_acl import MemoryScope

class MemoryScope(Enum):
    GLOBAL = "global"     # Visible across tenants (system data only)
    TENANT = "tenant"     # Visible within tenant
    PROJECT = "project"   # Visible within project
    SESSION = "session"   # Visible within session only
    PRIVATE = "private"   # Visible only to owner
```

Permissions

```
from taskforce.core.domain.memory_acl import MemoryPermission

class MemoryPermission(Enum):
    READ = "read"           # Can read memory content
    WRITE = "write"         # Can modify memory content
    DELETE = "delete"       # Can delete memory
    REFERENCE = "reference" # Can reference in citations/evidence
    SHARE = "share"         # Can share with other users
    ADMIN = "admin"         # Full control including ACL management
```

Complete ACL Example

```
from taskforce.core.domain.memory_acl import (
    MemoryACLManager,
    MemoryPermission,
    SensitivityLevel,
    MemoryScope
)

manager = MemoryACLManager()

# Create an ACL for a conversation memory resource
acl = manager.create_acl(
```

```
    resource_id="memory-conv-12345",
    resource_type="conversation",
    owner_id="alice",
    tenant_id="acme-corp",
    sensitivity=SensitivityLevel.INTERNAL,
    scope=MemoryScope.TENANT
)

print(f"ACL created: {acl.acl_id}")
print(f"Sensitivity: {acl.sensitivity.value}")  # internal
print(f"Scope: {acl.scope.value}")              # tenant
print(f"Owner: {acl.owner_id}")

# Owner always has full access
can_read = manager.check_access(
    resource_id="memory-conv-12345",
    user_id="alice",
    tenant_id="acme-corp",
    roles={"user"},
    permission=MemoryPermission.READ
)
print(f"Owner can read: {can_read}")  # True

can_admin = manager.check_access(
    resource_id="memory-conv-12345",
    user_id="alice",
    tenant_id="acme-corp",
    roles={"user"},
    permission=MemoryPermission.ADMIN
)
print(f"Owner can admin: {can_admin}")  # True

# Other users in same tenant can read (INTERNAL + TENANT scope)
bob_can_read = manager.check_access(
    resource_id="memory-conv-12345",
    user_id="bob",
    tenant_id="acme-corp",
    roles={"user"},
    permission=MemoryPermission.READ
)
print(f"Bob can read (same tenant): {bob_can_read}")  # True

# Grant specific write access to a user
entry = manager.grant_access(
    resource_id="memory-conv-12345",
    principal_type="user",
    principal_id="charlie",
    permissions={MemoryPermission.READ, MemoryPermission.WRITE},
    granted_by="alice"
)

charlie_can_write = manager.check_access(
    resource_id="memory-conv-12345",
    user_id="charlie",
```

```
    tenant_id="acme-corp",
    roles={"user"},
    permission=MemoryPermission.WRITE
)
print(f"Charlie can write (explicit grant): {charlie_can_write}")  # True

# Grant access to entire role
manager.grant_access(
    resource_id="memory-conv-12345",
    principal_type="role",
    principal_id="data_analyst",
    permissions={MemoryPermission.READ, MemoryPermission.REFERENCE},
    granted_by="alice"
)

# User with data_analyst role can read
analyst_can_read = manager.check_access(
    resource_id="memory-conv-12345",
    user_id="david",
    tenant_id="acme-corp",
    roles={"data_analyst"},
    permission=MemoryPermission.READ
)
print(f"Data analyst can read: {analyst_can_read}")  # True

# Cross-tenant access is blocked
other_tenant_read = manager.check_access(
    resource_id="memory-conv-12345",
    user_id="eve",
    tenant_id="other-corp",  # Different tenant
    roles={"admin"},
    permission=MemoryPermission.READ
)
print(f"Other tenant can read: {other_tenant_read}")  # False

# List all resources a user can access
accessible = manager.list_user_resources(
    user_id="charlie",
    tenant_id="acme-corp",
    roles={"user"}
)
print(f"Charlie can access: {accessible}")
```

5.2 Data Encryption

At-Rest Encryption (AES-256)

```
from taskforce.infrastructure.persistence.encryption import (
    DataEncryptor,
    KeyManager,
    EncryptionKey
```

```
)

# Initialize encryption with master key from environment
key_manager = KeyManager(
    master_key_env="TASKFORCE_ENCRYPTION_KEY"
)

encryptor = DataEncryptor(key_manager)

# Encrypt sensitive data
sensitive_data = {
    "user_query": "What is my account balance?",
    "tool_results": {"balance": 12345.67},
    "pii_detected": True
}

encrypted = encryptor.encrypt_json(
    data=sensitive_data,
    tenant_id="acme-corp",
    algorithm="fernet"  # or "aes-gcm" for AEAD
)

print(f"Encrypted length: {len(encrypted)} bytes")
# Output includes: key_id:algorithm:ciphertext

# Decrypt when needed
decrypted = encryptor.decrypt_json(
    encrypted_data=encrypted,
    tenant_id="acme-corp"
)
print(f"Decrypted: {decrypted}")
```

Key Management

```
# Each tenant gets a derived key from the master key
tenant_key = key_manager.get_or_create_tenant_key("acme-corp")

print(f"Key ID: {tenant_key.key_id}")        # tenant:acme-corp:v1
print(f"Algorithm: {tenant_key.algorithm}")   # fernet
print(f"Version: {tenant_key.version}")       # 1

# Key rotation (for compliance or after suspected compromise)
new_key = key_manager.rotate_tenant_key("acme-corp")
print(f"New Key ID: {new_key.key_id}")        # tenant:acme-corp:v2
print(f"New Version: {new_key.version}")      # 2
# Note: Old data must be re-encrypted with new key
```

EncryptedStateManager Usage

```
from taskforce.infrastructure.persistence.encryption import DataEncryptor
from taskforce.infrastructure.persistence.file_state_manager import
FileStateManager

# Create an encrypted state manager wrapper
class EncryptedStateManager:
    def __init__(self, base_manager: FileStateManager, encryptor: DataEncryptor,
tenant_id: str):
        self.base = base_manager
        self.encryptor = encryptor
        self.tenant_id = tenant_id

    async def save_state(self, session_id: str, state_data: dict) -> None:
        encrypted = self.encryptor.encrypt_json(state_data, self.tenant_id)
        # Store encrypted bytes
        await self.base.save_state(session_id, {"encrypted":
encrypted.decode('latin-1')})

    async def load_state(self, session_id: str) -> Optional[dict]:
        raw = await self.base.load_state(session_id)
        if raw and "encrypted" in raw:
            encrypted = raw["encrypted"].encode('latin-1')
            return self.encryptor.decrypt_json(encrypted, self.tenant_id)
        return raw

# Usage
encrypted_manager = EncryptedStateManager(
    base_manager=FileStateManager(work_dir=".taskforce"),
    encryptor=encryptor,
    tenant_id="acme-corp"
)

await encrypted_manager.save_state(
    "session-123",
    {"mission": "Handle sensitive data", "pii": {"ssn": "XXX-XX-1234"}}
)
```

5.3 Retention Policies

```
from taskforce.core.domain.memory_acl import ScopePolicy, MemoryScope,
SensitivityLevel

# Define retention policies per scope
session_policy = ScopePolicy(
    policy_id="session_policy",
    name="Session Data Policy",
    description="Short-term retention for session data",
    scope=MemoryScope.SESSION,
    sensitivity_levels={
        SensitivityLevel.PUBLIC,
```

```
        SensitivityLevel.INTERNAL,
        SensitivityLevel.CONFIDENTIAL
    },
    max_retention_days=30,
    requires_audit=False
)

tenant_policy = ScopePolicy(
    policy_id="tenant_policy",
    name="Tenant Data Policy",
    description="Medium-term retention for tenant data",
    scope=MemoryScope.TENANT,
    sensitivity_levels={
        SensitivityLevel.PUBLIC,
        SensitivityLevel.INTERNAL
    },
    max_retention_days=365,
    requires_audit=False
)

restricted_policy = ScopePolicy(
    policy_id="restricted_policy",
    name="Restricted Data Policy",
    description="High-security data with audit requirements",
    scope=MemoryScope.PRIVATE,
    sensitivity_levels={SensitivityLevel.RESTRICTED},
    max_retention_days=None,  # Manual deletion only
    requires_audit=True,
    allowed_roles={"admin", "security_admin"}
)

# Register policies with the ACL manager
manager.add_policy(session_policy)
manager.add_policy(tenant_policy)
manager.add_policy(restricted_policy)
```

---

6. Enterprise Operations

6.1 Usage Tracking

Track all billable resources including tokens, API calls, tool executions, and RAG queries.

```
from taskforce.application.reporting import UsageTracker
from taskforce.application.reporting.usage import UsageType
from datetime import datetime, timezone, timedelta

tracker = UsageTracker(max_records_in_memory=100000)

# Record token usage from LLM calls
tracker.record_tokens(
```

```
    tenant_id="acme-corp",
    input_tokens=1500,
    output_tokens=800,
    model="gpt-4o",
    user_id="alice",
    agent_id="customer-support-agent",
    session_id="session-abc123"
)

# Record another session with different model
tracker.record_tokens(
    tenant_id="acme-corp",
    input_tokens=5000,
    output_tokens=2000,
    model="gpt-4o-mini",
    user_id="bob",
    session_id="session-def456"
)

# Record agent execution
tracker.record_agent_execution(
    tenant_id="acme-corp",
    agent_id="customer-support-agent",
    user_id="alice",
    session_id="session-abc123",
    metadata={"mission": "Handle refund request"}
)

# Record tool executions
tracker.record_tool_execution(
    tenant_id="acme-corp",
    tool_name="semantic_search",
    user_id="alice",
    agent_id="customer-support-agent",
    session_id="session-abc123"
)

# Get aggregated usage for billing period
now = datetime.now(timezone.utc)
aggregation = tracker.get_aggregation(
    tenant_id="acme-corp",
    start_date=now - timedelta(days=30),
    end_date=now
)

print("Usage Aggregation:")
print(f"  Period: {aggregation.period_start.date()} to
{aggregation.period_end.date()}")
print(f"  Records: {aggregation.record_count}")
print(f"  Total input tokens: {aggregation.totals.get(UsageType.INPUT_TOKENS,
0):,}")
print(f"  Total output tokens: {aggregation.totals.get(UsageType.OUTPUT_TOKENS,
0):,}")
```

```
print("\nBy Model:")
for model, usage in aggregation.by_model.items():
    inp = usage.get(UsageType.INPUT_TOKENS, 0)
    out = usage.get(UsageType.OUTPUT_TOKENS, 0)
    print(f"  {model}: {inp:,} in / {out:,} out")

print("\nBy Agent:")
for agent_id, usage in aggregation.by_agent.items():
    total = usage.get(UsageType.TOTAL_TOKENS, 0)
    print(f"  {agent_id}: {total:,} tokens")
```

6.2 Cost Management

```
from taskforce.application.reporting import CostCalculator
from taskforce.application.reporting.cost import ModelPricing
from decimal import Decimal

calculator = CostCalculator()

# View default pricing (included for common models)
print("Default Model Pricing (per 1K tokens):")
for model, pricing in calculator.DEFAULT_PRICING.items():
    print(f"  {model}:")
    print(f"    Input: ${pricing.input_token_price}")
    print(f"    Output: ${pricing.output_token_price}")

# Add custom pricing for your organization
calculator.set_pricing("azure-gpt-4", ModelPricing(
    model_name="azure-gpt-4",
    input_token_price=Decimal("0.025"),   # Custom enterprise pricing
    output_token_price=Decimal("0.05"),
    currency="USD",
    notes="Azure EA pricing effective 2026-01"
))

# Generate cost report from usage aggregation
report = calculator.generate_report(aggregation)

print(f"\nCost Report: {report.report_id}")
print(f"Period: {report.period_start.date()} to {report.period_end.date()}")
print(f"Currency: {report.currency}")

print("\nLine Items:")
for item in report.line_items:
    print(f"  {item.description}")
    print(f"    Quantity: {item.quantity:,} {item.unit}")
    print(f"    Unit Price: ${item.unit_price}")
    print(f"    Total: ${item.total:.4f}")

print(f"\nSubtotal: ${report.subtotal:.4f}")
print(f"Adjustments: ${report.adjustments:.4f}")
```

```
print(f"Total: ${report.total:.4f}")

# Apply a discount
report.apply_adjustment(
    amount=Decimal("-5.00"),
    reason="Enterprise volume discount"
)
print(f"Total after discount: ${report.total:.4f}")

# Export for billing integration
import json
print(json.dumps(report.to_dict(), indent=2, default=str))
```

6.3 SLA Metrics

```
from taskforce.infrastructure.metrics import MetricsCollector,
get_metrics_collector

collector = MetricsCollector(prefix="taskforce")

# Record API request metrics
collector.record_request(
    endpoint="/api/v1/agents/execute",
    method="POST",
    status_code=200,
    duration_seconds=1.5,
    tenant_id="acme-corp"
)

# Record agent execution metrics
collector.record_agent_execution(
    agent_id="customer-support-agent",
    success=True,
    duration_seconds=8.5,
    steps=5,
    tokens_used=2300,
    tenant_id="acme-corp"
)

# Record tool execution metrics
collector.record_tool_execution(
    tool_name="semantic_search",
    success=True,
    duration_seconds=0.3,
    tenant_id="acme-corp"
)

# Use timer context manager for automatic timing
with collector.time("custom_operation_duration", labels={"operation":
"data_processing"}):
    # Your operation here
```

```
    import time
    time.sleep(0.1)

# Get SLA summary
sla = collector.get_sla_summary(tenant_id="acme-corp")
print(f"\nSLA Summary for acme-corp:")
print(f"  Total requests: {sla['total_requests']}")
print(f"  Total errors: {sla['total_errors']}")
print(f"  Error rate: {sla['error_rate_percent']:.2f}%")
print(f"  Latency P50: {sla['latency_p50_seconds']:.3f}s")
print(f"  Latency P95: {sla['latency_p95_seconds']:.3f}s")
print(f"  Latency P99: {sla['latency_p99_seconds']:.3f}s")

# Export Prometheus format for Grafana dashboards
prometheus_output = collector.export_prometheus()
print("\nPrometheus Metrics:")
print(prometheus_output)
```

Prometheus Output Format:

```
# TYPE taskforce_http_requests_total counter
taskforce_http_requests_total{endpoint="/api/v1/agents/execute",method="POST",stat
us="200",tenant_id="acme-corp"} 1.0
# TYPE taskforce_http_request_duration_seconds histogram
taskforce_http_request_duration_seconds_bucket{endpoint="/api/v1/agents/execute",m
ethod="POST",status="200",tenant_id="acme-corp",le="0.5"} 0
taskforce_http_request_duration_seconds_bucket{endpoint="/api/v1/agents/execute",m
ethod="POST",status="200",tenant_id="acme-corp",le="1.0"} 0
taskforce_http_request_duration_seconds_bucket{endpoint="/api/v1/agents/execute",m
ethod="POST",status="200",tenant_id="acme-corp",le="2.5"} 1
taskforce_http_request_duration_seconds_sum{endpoint="/api/v1/agents/execute",meth
od="POST",status="200",tenant_id="acme-corp"} 1.5
taskforce_http_request_duration_seconds_count{endpoint="/api/v1/agents/execute",me
thod="POST",status="200",tenant_id="acme-corp"} 1
```

---

7. Compliance & Audit

7.1 Audit Trail

All significant actions in PyTaskforce generate structured audit events.

```
import structlog

logger = structlog.get_logger(__name__)

# Contextual logging with tenant/user context
logger.info(
    "agent.execution.started",
```

```
    tenant_id="acme-corp",
    user_id="alice",
    session_id="session-abc123",
    agent_id="customer-support-agent",
    mission="Handle customer refund request"
)

logger.info(
    "tool.executed",
    tenant_id="acme-corp",
    session_id="session-abc123",
    tool_name="semantic_search",
    duration_ms=285,
    result_count=5
)

logger.info(
    "agent.execution.completed",
    tenant_id="acme-corp",
    session_id="session-abc123",
    agent_id="customer-support-agent",
    success=True,
    total_steps=5,
    tokens_used=2300,
    duration_seconds=8.5
)
```

7.2 Compliance Frameworks

PyTaskforce provides built-in evidence collection for major compliance frameworks.

SOC 2 Type II Mapping

Trust Services Criteria	Control	PyTaskforce Feature
CC1 (Organization)	CC1.1	Tenant isolation, RBAC configuration
CC2 (Communications)	CC2.1	API documentation, error messages
CC3 (Risk Assessment)	CC3.1	Sensitivity levels, scope policies
CC4 (Monitoring)	CC4.1	Structured logging, metrics collection
CC5 (Control Activities)	CC5.1	Approval workflows, version control
CC6 (Access Controls)	CC6.1	RBAC, memory ACLs, authentication
CC7 (Operations)	CC7.1	Evidence chain tracking, audit logs
CC8 (Change Management)	CC8.1	Agent versioning, approval workflows
CC9 (Risk Mitigation)	CC9.1	Encryption, data retention policies

ISO 27001 Annex A Mapping

Control	Description	PyTaskforce Feature
A.5	Information Security Policies	Tenant settings, scope policies
A.6	Organization of Security	RBAC, role hierarchy
A.9	Access Control	JWT/API key auth, permissions
A.10	Cryptography	AES-256 encryption, key rotation
A.12	Operations Security	Audit logging, metrics
A.13	Communications Security	TLS, tenant isolation
A.16	Incident Management	Error tracking, alerting
A.18	Compliance	Evidence export, reporting

GDPR Article 30 Records

PyTaskforce automatically generates GDPR processing records.

7.3 Evidence Export

Complete SOC2 Report Generation

```
from taskforce.application.reporting import (
    ComplianceEvidenceCollector,
    ComplianceReportGenerator,
    ComplianceFramework,
    export_compliance_package
)
from datetime import datetime, timezone, timedelta

# Initialize collector for your tenant
collector = ComplianceEvidenceCollector(tenant_id="acme-corp")

# Collect all evidence
all_evidence = collector.collect_all_evidence()
print(f"Collected {len(all_evidence)} evidence items")

# Generate SOC2 report
generator = ComplianceReportGenerator(collector)
now = datetime.now(timezone.utc)

soc2_report = generator.generate_soc2_report(
    period_start=now - timedelta(days=90),  # Q4 2025
    period_end=now
)

print(f"\nSOC2 Report Generated:")
print(f"  Report ID: {soc2_report.report_id}")
print(f"  Framework: {soc2_report.framework.value}")
```

```
print(f"  Period: {soc2_report.period_start.date()} to
{soc2_report.period_end.date()}")
print(f"  Evidence items: {len(soc2_report.evidence_items)}")

print(f"\nExecutive Summary:")
print(soc2_report.summary)

# Group evidence by control
print("\nEvidence by Control:")
controls = set(e.control_id for e in soc2_report.evidence_items)
for control_id in sorted(controls):
    items = soc2_report.get_evidence_by_control(control_id)
    print(f"  {control_id}: {len(items)} items")
    for item in items:
        print(f"    - {item.title}")

# Export as JSON for auditors
json_package = export_compliance_package(soc2_report, format="json")
with open("soc2_evidence_q4_2025.json", "w") as f:
    f.write(json_package)

# Export as Markdown for internal review
markdown_package = export_compliance_package(soc2_report, format="markdown")
with open("soc2_evidence_q4_2025.md", "w") as f:
    f.write(markdown_package)
```

GDPR Processing Records

```

# Generate GDPR Article 30 records
gdpr_records = generator.generate_gdpr_records()

print(f"\nGDPR Processing Records: {len(gdpr_records)}")
for record in gdpr_records:
    print(f"\nRecord: {record.record_id}")
    print(f"  Purpose: {record.processing_purpose}")
    print(f"  Data Categories: {', '.join(record.data_categories)}")
    print(f"  Data Subjects: {', '.join(record.data_subjects)}")
    print(f"  Recipients: {', '.join(record.recipients)}")
    print(f"  Retention: {record.retention_period}")
    print(f"  Security Measures:")
    for measure in record.security_measures:
        print(f"    - {measure}")
    if record.transfers:
        print(f"  International Transfers:")
        for transfer in record.transfers:
            print(f"    - {transfer['destination']}: {transfer['safeguards']}")
```

ISO 27001 Report

```
iso_report = generator.generate_iso27001_report(
    period_start=now - timedelta(days=365),
    period_end=now
)

print(f"\nISO 27001 Report Generated:")
print(f"  Evidence items: {len(iso_report.evidence_items)}")
print(f"\nControls covered:")
for control_id in sorted(set(e.control_id for e in iso_report.evidence_items)):
    print(f"  - {control_id}")
```

---

8. RAG & Knowledge Management

8.1 Source Citations

PyTaskforce automatically extracts and formats citations from RAG search results, enabling transparent and auditable AI responses.

Citation Extraction

```
from taskforce.infrastructure.tools.rag.citations import (
    RAGCitation,
    RAGCitationExtractor,
    CitationFormatter,
    CitationStyle,
    create_citation_formatter
)

# Citations are automatically extracted from semantic search results
search_result = {
    "success": True,
    "results": [
        {
            "document_id": "doc-policy-001",
            "document_title": "Employee Handbook 2026",
            "content": "Employees are entitled to 25 days of annual leave per
calendar year.",
            "score": 0.95,
            "page_number": 42
        },
        {
            "document_id": "doc-hr-faq",
            "document_title": "HR FAQ Document",
            "content": "Leave requests must be submitted at least 2 weeks in
advance.",
            "score": 0.87,
            "page_number": 5
        }
    ]
```

```
}

# Extract citations
citations = RAGCitationExtractor.extract_from_semantic_search(search_result)

print(f"Extracted {len(citations)} citations:")
for i, citation in enumerate(citations, 1):
    print(f"\n[{i}] {citation.title}")
    print(f"    Document ID: {citation.document_id}")
    print(f"    Score: {citation.score:.2f}")
    print(f"    Page: {citation.page_number}")
    print(f"    Snippet: {citation.snippet[:80]}...")
```

Citation Formatting

```
# Create citations manually for demonstration
citations = [
    RAGCitation(
        document_id="doc-policy-001",
        title="Employee Handbook 2026",
        snippet="Employees are entitled to 25 days of annual leave.",
        score=0.95,
        page_number=42
    ),
    RAGCitation(
        document_id="doc-hr-faq",
        title="HR FAQ Document",
        snippet="Leave requests must be submitted 2 weeks in advance.",
        score=0.87,
        page_number=5
    )
]

# Format with inline style [1], [2]
formatter_inline = CitationFormatter(
    style=CitationStyle.INLINE,
    max_citations=10,
    include_score=False,
    include_page=True
)

agent_response = "According to our policy, you have 25 days of annual leave.
Please submit your requests at least 2 weeks in advance."

result = formatter_inline.format_citations(agent_response, citations)

print("Inline Citation Style:")
print(f"  Text: {result.formatted_text}")
print(f"\n  References:")
for ref in result.references:
    print(f"    {ref}")
```

```
# Format with appendix style
formatter_appendix = CitationFormatter(
    style=CitationStyle.APPENDIX,
    include_score=True
)

result_appendix = formatter_appendix.format_citations(agent_response, citations)

print("\n\nAppendix Citation Style:")
print(result_appendix.formatted_text)
```

Output:

```
Inline Citation Style:
  Text: According to our policy, you have 25 days of annual leave [1]. Please
submit your requests at least 2 weeks in advance [2].

  References:
    [1] Employee Handbook 2026 p. 42
    [2] HR FAQ Document p. 5


Appendix Citation Style:
According to our policy, you have 25 days of annual leave. Please submit your
requests at least 2 weeks in advance.

---
**References:**
[1] Employee Handbook 2026 p. 42 (relevance: 0.95)
[2] HR FAQ Document p. 5 (relevance: 0.87)
```

Configuration-Based Formatting

```
# Load citation settings from profile config
citation_config = {
    "citation_style": "appendix",
    "max_citations": 5,
    "include_score": True,
    "include_page": True
}

formatter = create_citation_formatter(citation_config)
```

8.2 Document Access Control

RAG documents inherit the same ACL system as memory resources.

```
# Apply ACL to a document in the knowledge base
doc_acl = manager.create_acl(
    resource_id="doc-confidential-financials",
    resource_type="document",
    owner_id="finance-team",
    tenant_id="acme-corp",
    sensitivity=SensitivityLevel.CONFIDENTIAL,
    scope=MemoryScope.PROJECT  # Only accessible within finance project
)

# Grant access to the finance role
manager.grant_access(
    resource_id="doc-confidential-financials",
    principal_type="role",
    principal_id="finance_analyst",
    permissions={MemoryPermission.READ, MemoryPermission.REFERENCE},
    granted_by="finance-admin"
)

# RAG searches respect ACLs - unauthorized users won't see confidential docs
def filtered_semantic_search(query: str, user: UserContext, manager:
MemoryACLManager):
    """Search with ACL filtering."""
    raw_results = semantic_search_tool.execute(query=query)

    filtered = []
    for result in raw_results["results"]:
        doc_id = result["document_id"]
        can_read = manager.check_access(
            resource_id=doc_id,
            user_id=user.user_id,
            tenant_id=user.tenant_id,
            roles=user.roles,
            permission=MemoryPermission.READ
        )
        if can_read:
            filtered.append(result)

    return {"success": True, "results": filtered}
```

---

9. Integration & Extensibility

9.1 API Integration

All enterprise features are accessible via REST API.

Agent Execution API

```
# Execute an agent
curl -X POST https://api.taskforce.example.com/v1/agents/execute \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "agent_id": "customer-support-agent",
    "mission": "Help customer with refund request for order #12345",
    "context": {
      "customer_id": "cust-789",
      "order_id": "order-12345"
    }
  }'
```

Admin APIs

```
# List agents in catalog
curl -X GET https://api.taskforce.example.com/v1/admin/agents \
  -H "Authorization: Bearer $TOKEN"

# Get usage summary
curl -X GET "https://api.taskforce.example.com/v1/admin/usage?start=2026-01-
01&end=2026-01-31" \
  -H "Authorization: Bearer $TOKEN"

# Generate compliance report
curl -X POST https://api.taskforce.example.com/v1/admin/compliance/soc2 \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "period_start": "2025-10-01",
    "period_end": "2025-12-31",
    "format": "json"
  }'
```

9.2 Webhook Support

```
# Configure webhooks for workflow events
workflow_config = {
    "webhooks": {
        "on_request_created": {
            "url": "https://your-app.com/webhooks/approval-created",
            "secret": "webhook-secret-key"
        },
        "on_request_approved": {
            "url": "https://your-app.com/webhooks/approval-completed",
            "secret": "webhook-secret-key"
        }
    }
```

```
}

# Webhook payload example
{
    "event": "request_approved",
    "timestamp": "2026-01-15T10:30:00Z",
    "payload": {
        "request_id": "req-abc123",
        "tenant_id": "acme-corp",
        "resource_type": "agent",
        "resource_id": "customer-support-agent",
        "approved_by": ["bob", "carol"]
    },
    "signature": "sha256=..."
}
```

9.3 Extension Points

Custom Approval Workflows

```
from taskforce.application.workflows.approval import ApprovalWorkflow, RequestType

# Create a custom request type
class CustomRequestType(RequestType):
    AGENT_PUBLISH = "agent_publish"
    DATA_EXPORT = "data_export"
    PERMISSION_ESCALATION = "permission_escalation"

# Register custom workflow
data_export_workflow = ApprovalWorkflow(
    workflow_id="data-export-approval",
    tenant_id="acme-corp",
    name="Data Export Approval",
    description="Requires legal and security approval for data exports",
    request_type=RequestType.CUSTOM,
    required_approvers=2,
    allowed_approver_roles={"legal_counsel", "security_admin"},
    auto_expire_hours=24,
    metadata={"custom_type": "data_export"}
)
```

Custom Scope Policies

```
# Create industry-specific retention policies
hipaa_policy = ScopePolicy(
    policy_id="hipaa_retention",
    name="HIPAA Compliant Retention",
    description="6-year retention for healthcare data",
```

```
    scope=MemoryScope.TENANT,
    sensitivity_levels={SensitivityLevel.CONFIDENTIAL,
SensitivityLevel.RESTRICTED},
    max_retention_days=2190,  # 6 years
    requires_audit=True,
    allowed_roles={"hipaa_authorized"}
)
```

---

10. Configuration & Deployment

10.1 Profile-Based Configuration

```yaml
# configs/enterprise.yaml
profile: enterprise

# Agent configuration
agent:
  type: generic
  planning_strategy: native_react
  max_iterations: 15

# Enterprise features
enterprise:
  multi_tenant: true

  # Authentication
  authentication:
    type: jwt
    jwt:
      issuer: "https://auth.yourcompany.com"
      audience: "taskforce-api"
      algorithms: ["RS256"]
      jwks_uri: "https://auth.yourcompany.com/.well-known/jwks.json"
      tenant_claim: "org_id"
      roles_claim: "roles"

  # RBAC
  rbac:
    enabled: true
    default_role: viewer
    custom_roles:
      - name: data_analyst
        permissions:
          - agent:read
          - agent:execute
          - session:create
          - memory:read

  # Encryption
  encryption:
```

```yaml
    enabled: true
    algorithm: fernet
    key_source: env:TASKFORCE_ENCRYPTION_KEY

  # Audit
  audit:
    enabled: true
    retention_days: 365
    include_request_body: false  # For PII protection

  # Compliance
  compliance:
    frameworks:
      - SOC2
      - ISO27001
      - GDPR
    evidence_collection: automatic

# Persistence
persistence:
  type: database
  database_url: ${DATABASE_URL}
  connection_pool_size: 20
  ssl_mode: require

# LLM Provider
llm:
  provider: azure
  deployment_name: ${AZURE_OPENAI_DEPLOYMENT}
  temperature: 0.5
  max_tokens: 4096

# Metrics
metrics:
  enabled: true
  prefix: taskforce
  export:
    prometheus: true
    endpoint: /metrics

# Logging
logging:
  level: INFO
  format: json
  include_context:
    - tenant_id
    - user_id
    - session_id
    - agent_id

# Rate Limiting
rate_limiting:
  enabled: true
```

```
  default_limit: 100  # requests per minute
  by_tenant: true
```

10.2 Environment Variables Reference

Variable	Required	Description	Example
TASKFORCE_PROFILE	No	Configuration profile	enterprise
DATABASE_URL	Yes (prod)	PostgreSQL connection	postgresql://...
TASKFORCE_ENCRYPTION_KEY	Yes (prod)	Base64-encoded 32-byte key	base64...
JWT_SECRET	Conditional	JWT signing secret (HS256)	your-secret
OPENAI_API_KEY	Conditional	OpenAI API key	sk-...
AZURE_OPENAI_API_KEY	Conditional	Azure OpenAI key	...
AZURE_OPENAI_ENDPOINT	Conditional	Azure endpoint	https://....openai.azure.com
AZURE_OPENAI_DEPLOYMENT	Conditional	Azure deployment name	gpt-4o

10.3 Deployment Scenarios

Docker Compose (Development/Staging)

```
# docker-compose.yml
version: '3.8'

services:
  taskforce:
    image: taskforce:latest
    build: .
    ports:
      - "8000:8000"
    environment:
      - TASKFORCE_PROFILE=enterprise
      - DATABASE_URL=postgresql://postgres:password@db:5432/taskforce
      - TASKFORCE_ENCRYPTION_KEY=${ENCRYPTION_KEY}
      - AZURE_OPENAI_API_KEY=${AZURE_OPENAI_API_KEY}
      - AZURE_OPENAI_ENDPOINT=${AZURE_OPENAI_ENDPOINT}
    depends_on:
      - db
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 30s
```

```
      timeout: 10s
      retries: 3

  db:
    image: postgres:15
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=taskforce
    volumes:
      - pgdata:/var/lib/postgresql/data

  prometheus:
    image: prom/prometheus:latest
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml

  grafana:
    image: grafana/grafana:latest
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana-data:/var/lib/grafana

volumes:
  pgdata:
  grafana-data:
```

Kubernetes/Helm (Production)

```
# values.yaml
replicaCount: 3

image:
  repository: your-registry/taskforce
  tag: "1.0.0"
  pullPolicy: IfNotPresent

service:
  type: ClusterIP
  port: 8000

ingress:
  enabled: true
  className: nginx
  hosts:
    - host: taskforce.yourcompany.com
```

```yaml
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: taskforce-tls
      hosts:
        - taskforce.yourcompany.com

resources:
  requests:
    memory: "512Mi"
    cpu: "250m"
  limits:
    memory: "2Gi"
    cpu: "1000m"

autoscaling:
  enabled: true
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 70

env:
  - name: TASKFORCE_PROFILE
    value: enterprise
  - name: DATABASE_URL
    valueFrom:
      secretKeyRef:
        name: taskforce-secrets
        key: database-url
  - name: TASKFORCE_ENCRYPTION_KEY
    valueFrom:
      secretKeyRef:
        name: taskforce-secrets
        key: encryption-key

livenessProbe:
  httpGet:
    path: /health
    port: 8000
  initialDelaySeconds: 10
  periodSeconds: 30

readinessProbe:
  httpGet:
    path: /health
    port: 8000
  initialDelaySeconds: 5
  periodSeconds: 10

serviceMonitor:
  enabled: true
  interval: 30s
  path: /metrics
```

Production Checklist

- ☐ **Security**
  - ☐ TLS certificates configured
  - ☐ Encryption key generated and stored securely
  - ☐ Database credentials in secrets management
  - ☐ Network policies configured
- ☐ **High Availability**
  - ☐ Minimum 3 replicas
  - ☐ Database connection pooling
  - ☐ Load balancer health checks
  - ☐ Pod disruption budgets
- ☐ **Monitoring**
  - ☐ Prometheus scraping configured
  - ☐ Grafana dashboards imported
  - ☐ Alerting rules configured
  - ☐ Log aggregation (ELK/Loki)
- ☐ **Compliance**
  - ☐ Audit log retention configured
  - ☐ Data retention policies set
  - ☐ Encryption at rest verified
  - ☐ Access logging enabled

---

11. Use Case Scenarios

11.1 Customer Service Automation

Scenario: A SaaS company wants to deploy AI agents to handle customer inquiries, with different agent configurations per customer tier.

```
┌─────────────────────────────────────────────────────────────┐
│                    Customer Service Platform                │
├─────────────────────────────────────────────────────────────┤
│  Tenant: Enterprise Customer A                              │
│  ┌──────────────────────────────────────────────────┐       │
│  │ Agent: Premium Support Agent (v2.1.0)            │       │
│  │ - Full RAG access to knowledge base              │       │
│  │ - Create/escalate tickets                        │       │
│  │ - Process refunds up to $500                     │       │
│  │ - SLA: 99.9% uptime, <2s response time           │       │
│  └──────────────────────────────────────────────────┘       │
```

```
┌──────────────────────────────────────────────────────────────┐
│  Tenant: Standard Customer B                                 │
│  ┌──────────────────────────────────────────────────────┐    │
│  │ Agent: Standard Support Agent (v1.5.0)               │    │
│  │ - Limited RAG access (FAQ only)                      │    │
│  │ - Create tickets (no escalation)                     │    │
│  │ - No refund processing                               │    │
│  │ - SLA: 99.5% uptime                                  │    │
│  └──────────────────────────────────────────────────────┘    │
└──────────────────────────────────────────────────────────────┘
```

Implementation Highlights:

```python
# Create tier-specific agent definitions
premium_agent = catalog.create_agent(
    tenant_id="enterprise-customer-a",
    name="Premium Support Agent",
    category="support",
    tags={"tier:premium", "refunds", "escalation"},
    initial_definition={
        "tools": ["semantic_search", "create_ticket", "escalate_ticket",
"process_refund"],
        "rag_indexes": ["full-knowledge-base"],
        "system_prompt": "Premium support with full capabilities...",
        "refund_limit": 500
    }
)

standard_agent = catalog.create_agent(
    tenant_id="standard-customer-b",
    name="Standard Support Agent",
    category="support",
    tags={"tier:standard"},
    initial_definition={
        "tools": ["semantic_search", "create_ticket"],
        "rag_indexes": ["faq-only"],
        "system_prompt": "Standard support for common inquiries..."
    }
)
```

11.2 Internal Knowledge Assistants

Scenario: A large enterprise wants department-specific AI assistants with controlled knowledge sharing.

```
┌──────────────────────────────────────────────────────────────┐
│                    Enterprise Knowledge Platform             │
├──────────────────────────────────────────────────────────────┤
│  Shared Knowledge Base (INTERNAL scope)                      │
│  ├── Company Policies                                        │
```

```
│   ├── General HR Information                                          │
│   └── IT Support Guides                                               │
├──────────────────────────────────────────────────────────────────────┤
│  Engineering Department (PROJECT scope)                               │
│   ┌──────────────────────────────────────┐                            │
│   │ Agent: Engineering Assistant         │                            │
│   │ Memory: Engineering-specific docs    │                            │
│   │ ACL: engineering role required       │                            │
│   └──────────────────────────────────────┘                            │
├──────────────────────────────────────────────────────────────────────┤
│  Finance Department (PROJECT scope: CONFIDENTIAL)                     │
│   ┌──────────────────────────────────────┐                            │
│   │ Agent: Finance Assistant             │                            │
│   │ Memory: Financial reports, forecasts │                            │
│   │ ACL: finance role + audit logging    │                            │
│   └──────────────────────────────────────┘                            │
├──────────────────────────────────────────────────────────────────────┤
│  Executive Team (PRIVATE scope: RESTRICTED)                           │
│   ┌──────────────────────────────────────┐                            │
│   │ Agent: Executive Briefing Assistant  │                            │
│   │ Memory: Board materials, M&A docs    │                            │
│   │ ACL: executive role + encryption     │                            │
│   └──────────────────────────────────────┘                            │
└──────────────────────────────────────────────────────────────────────┘
```

Memory Sharing Pattern:

```python
# Shared knowledge - all authenticated users in tenant can read
shared_acl = manager.create_acl(
    resource_id="shared-policies",
    resource_type="knowledge_base",
    owner_id="system",
    tenant_id="acme-corp",
    sensitivity=SensitivityLevel.INTERNAL,
    scope=MemoryScope.TENANT
)

# Department-specific knowledge
engineering_acl = manager.create_acl(
    resource_id="engineering-docs",
    resource_type="knowledge_base",
    owner_id="engineering-lead",
    tenant_id="acme-corp",
    sensitivity=SensitivityLevel.INTERNAL,
    scope=MemoryScope.PROJECT
)
manager.grant_access(
    resource_id="engineering-docs",
    principal_type="role",
    principal_id="engineering",
    permissions={MemoryPermission.READ, MemoryPermission.REFERENCE},
    granted_by="engineering-lead"
```

```
)

# Executive-only with encryption
executive_acl = manager.create_acl(
    resource_id="executive-materials",
    resource_type="knowledge_base",
    owner_id="ceo",
    tenant_id="acme-corp",
    sensitivity=SensitivityLevel.RESTRICTED,
    scope=MemoryScope.PRIVATE
)
```

11.3 Compliance-Sensitive Industries

Financial Services

```
# SOX-compliant agent deployment
financial_agent_config = {
    "agent": {
        "name": "Financial Analysis Agent",
        "require_approval": True,
        "approval_roles": ["compliance_officer", "cfo"]
    },
    "memory": {
        "encryption": "aes-256-gcm",
        "retention_days": 2555,  # 7 years for SOX
        "sensitivity": "restricted"
    },
    "audit": {
        "log_all_queries": True,
        "log_tool_results": True,
        "compliance_frameworks": ["SOC2", "SOX"]
    },
    "tools": {
        "enabled": ["semantic_search", "calculate"],
        "disabled": ["web_search", "code_execution"]  # No external access
    }
}
```

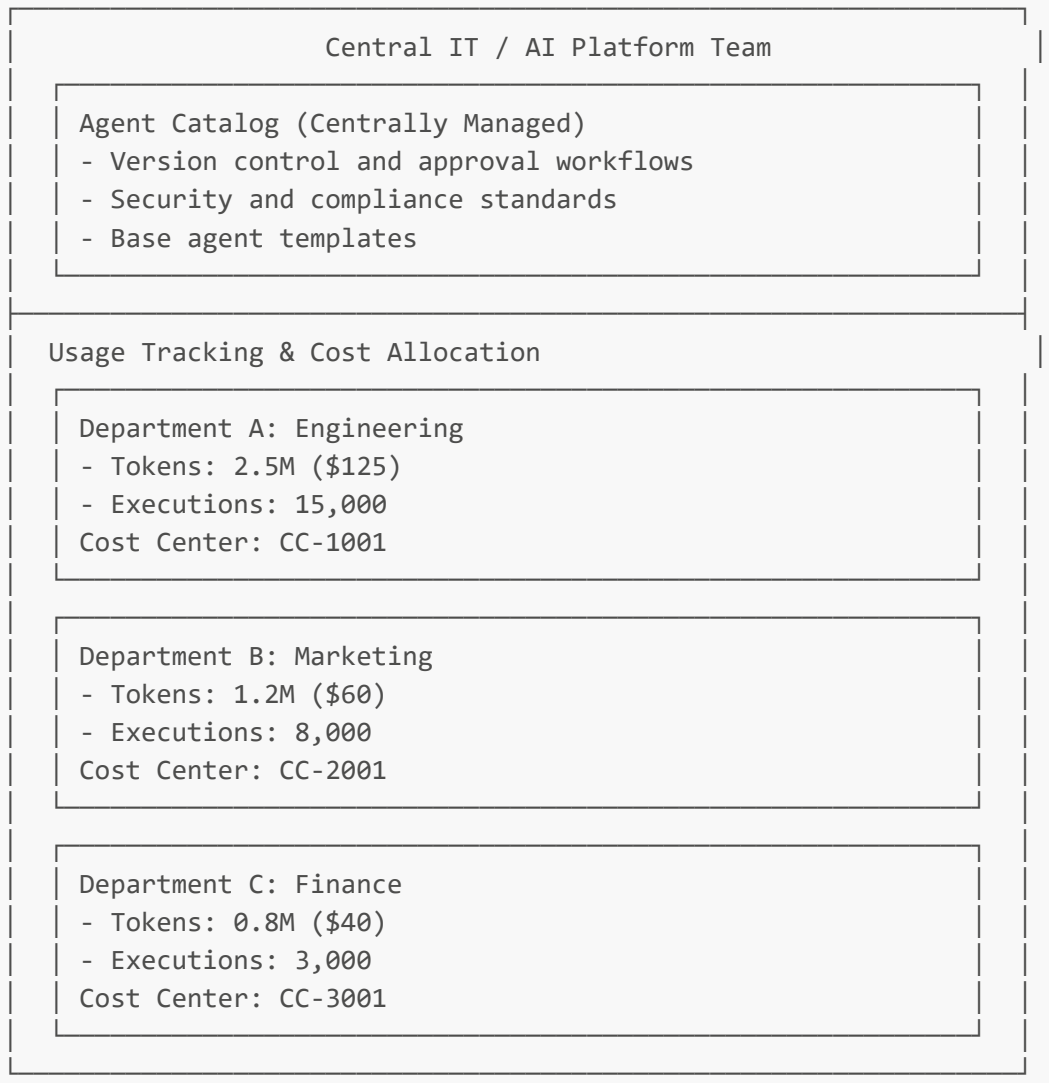
Healthcare (HIPAA Considerations)

```
# HIPAA-aligned configuration
healthcare_config = {
    "agent": {
        "name": "Clinical Documentation Assistant",
        "phi_handling": "enabled",
        "de_identification_required": True
    },
```

```
    "memory": {
        "encryption": "aes-256-gcm",
        "retention_days": 2190,  # 6 years minimum for HIPAA
        "sensitivity": "restricted",
        "baa_required": True  # Business Associate Agreement with LLM provider
    },
    "audit": {
        "log_phi_access": True,
        "access_log_retention_days": 2190,
        "compliance_frameworks": ["HIPAA", "SOC2"]
    },
    "access_control": {
        "minimum_necessary": True,  # HIPAA principle
        "role_based": True,
        "audit_all_access": True
    }
}
```

11.4 Multi-Department Enterprise Deployment

Scenario: Centralized agent catalog with cost allocation by department.



Cost Allocation Implementation:

```
# Track usage with department metadata
tracker.record_tokens(
    tenant_id="acme-corp",
    input_tokens=1500,
    output_tokens=800,
    model="gpt-4o",
    user_id="alice",
    metadata={
        "department": "engineering",
        "cost_center": "CC-1001",
        "project": "code-review-automation"
    }
)

# Generate department-specific cost reports
for department in ["engineering", "marketing", "finance"]:
    dept_records = tracker.get_records(
        tenant_id="acme-corp",
        usage_type=UsageType.TOTAL_TOKENS
    )
    dept_records = [r for r in dept_records if r.metadata.get("department") ==
department]

    # Calculate costs
    total_tokens = sum(r.quantity for r in dept_records)
    # ... generate chargeback report
```

---

12. Decision Guide

12.1 Feature Comparison Matrix

Feature	Community	Enterprise
Agent execution	Yes	Yes
File-based persistence	Yes	Yes
Database persistence	Limited	Full
Multi-tenant isolation	No	Yes
JWT/OAuth2 authentication	No	Yes
API key management	No	Yes
Role-based access control	No	Yes
Agent versioning	No	Yes

Feature	Community	Enterprise
Approval workflows	No	Yes
Memory encryption	No	Yes
Memory ACLs	No	Yes
Usage tracking	Basic	Full
Cost reporting	No	Yes
SLA metrics	No	Yes
Prometheus export	No	Yes
SOC 2 evidence	No	Yes
ISO 27001 evidence	No	Yes
GDPR records	No	Yes
RAG citations	Basic	Full
Custom roles	No	Yes
Webhook integrations	No	Yes

12.2 When is Taskforce Enterprise the Right Choice?

Choose Enterprise when you need:

1. **Multi-tenant deployment**
    - You're building a SaaS product with multiple customers
    - You need complete data isolation between organizations
    - You want per-tenant configuration and usage tracking
2. **Compliance requirements**
    - You need SOC 2, ISO 27001, or GDPR certification
    - You require audit trails for all AI actions
    - You operate in regulated industries (finance, healthcare, legal)
3. **Enterprise security**
    - You require SSO/OAuth2 integration
    - You need role-based access control
    - You want encryption at rest for all data
4. **Operational visibility**
    - You need to track and bill for AI usage
    - You want to monitor SLA compliance
    - You need cost allocation by department/project

5. **Governed AI deployment**
    - You want version control for AI agents
    - You need approval workflows before production deployment
    - You require audit trails for agent changes

12.3 When to Consider Alternatives

Community edition may suffice when:

- Single-tenant, internal tool usage
- Development and experimentation
- No compliance requirements
- Cost sensitivity with low usage

Other solutions may be better when:

- You need real-time streaming responses (consider custom LangChain/LlamaIndex)
- You need tight integration with specific platforms (consider platform-native solutions)
- You have existing agent orchestration (consider incremental enhancement)

12.4 Getting Started Checklist

- ☐ **Evaluate requirements**
    - ☐ Document multi-tenant requirements
    - ☐ List compliance frameworks needed
    - ☐ Define security requirements
    - ☐ Estimate usage volumes
- ☐ **Technical setup**
    - ☐ Provision infrastructure (database, secrets management)
    - ☐ Configure authentication provider (Auth0, Okta, Azure AD)
    - ☐ Set up monitoring stack (Prometheus, Grafana)
    - ☐ Deploy initial instance
- ☐ **Configuration**
    - ☐ Create enterprise profile
    - ☐ Configure RBAC roles and permissions
    - ☐ Set up encryption keys
    - ☐ Configure retention policies
- ☐ **Integration**
    - ☐ Integrate with existing authentication
    - ☐ Connect to knowledge bases
    - ☐ Set up webhook endpoints
    - ☐ Configure alerting
- ☐ **Go-live preparation**

- [ ] Run compliance evidence collection
- [ ] Test failover scenarios
- [ ] Document runbooks
- [ ] Train operations team

Appendix

A. Complete Permission Matrix

Permission	admin	agent_designer	operator	auditor	viewer
`agent:create`	X	X			
`agent:read`	X	X	X	X	X
`agent:update`	X	X			
`agent:delete`	X	X			
`agent:execute`	X		X		
`session:create`	X		X		
`session:read`	X	X	X	X	X
`session:delete`	X		X		
`tool:execute`	X		X		
`tool:read`	X	X	X	X	X
`memory:read`	X		X	X	
`memory:write`	X		X		
`memory:delete`	X				
`tenant:manage`	X				
`user:manage`	X				
`role:manage`	X				
`policy:manage`	X				
`audit:read`	X			X	
`system:config`	X				
`system:metrics`	X				

B. API Reference for Admin Endpoints

Tenant Management

```
GET    /api/v1/admin/tenants
POST   /api/v1/admin/tenants
GET    /api/v1/admin/tenants/{tenant_id}
PUT    /api/v1/admin/tenants/{tenant_id}
DELETE /api/v1/admin/tenants/{tenant_id}
```

User Management

```
GET    /api/v1/admin/users
POST   /api/v1/admin/users
GET    /api/v1/admin/users/{user_id}
PUT    /api/v1/admin/users/{user_id}
DELETE /api/v1/admin/users/{user_id}
```

Agent Catalog

```
GET    /api/v1/admin/agents
POST   /api/v1/admin/agents
GET    /api/v1/admin/agents/{agent_id}
PUT    /api/v1/admin/agents/{agent_id}
DELETE /api/v1/admin/agents/{agent_id}
GET    /api/v1/admin/agents/{agent_id}/versions
POST   /api/v1/admin/agents/{agent_id}/versions
POST   /api/v1/admin/agents/{agent_id}/versions/{version_id}/submit
POST   /api/v1/admin/agents/{agent_id}/versions/{version_id}/approve
POST   /api/v1/admin/agents/{agent_id}/versions/{version_id}/publish
```

Approval Workflows

```
GET    /api/v1/admin/workflows
POST   /api/v1/admin/workflows
GET    /api/v1/admin/requests
GET    /api/v1/admin/requests/pending
POST   /api/v1/admin/requests
POST   /api/v1/admin/requests/{request_id}/approve
POST   /api/v1/admin/requests/{request_id}/reject
```

Reporting

```
GET    /api/v1/admin/usage
GET    /api/v1/admin/usage/aggregation
GET    /api/v1/admin/costs
```

```
POST   /api/v1/admin/costs/report
GET    /api/v1/admin/compliance/evidence
POST   /api/v1/admin/compliance/soc2
POST   /api/v1/admin/compliance/iso27001
POST   /api/v1/admin/compliance/gdpr-records
```

Metrics

```
GET    /metrics              # Prometheus format
GET    /api/v1/admin/sla     # SLA summary
GET    /health               # Health check
GET    /ready                # Readiness check
```

C. Glossary of Terms

Term	Definition
ACL	Access Control List - defines who can access a resource and with what permissions
Agent	An autonomous AI entity that executes tasks using LLM reasoning and tools
Catalog	Central registry for managing agent definitions and versions
Fernet	Symmetric encryption scheme using AES-128-CBC with HMAC for authentication
OIDC	OpenID Connect - authentication layer on top of OAuth 2.0
RBAC	Role-Based Access Control - permission management through role assignments
ReAct	Reason + Act pattern - iterative thinking and action execution loop
Scope	Visibility boundary for data (global, tenant, project, session, private)
Sensitivity	Classification level for data protection (public, internal, confidential, restricted)
Tenant	An organizational unit representing a customer or business entity
Workflow	Governed process requiring approvals before completion

D. Migration Guide from Community Edition

Step 1: Database Migration

```
# Export existing data
taskforce export --output ./backup

# Set up enterprise database
export DATABASE_URL="postgresql://..."

# Run migrations
uv run alembic upgrade head
```

```
# Import data with tenant assignment
taskforce import --input ./backup --tenant-id your-tenant
```

Step 2: Configuration Update

```
# Before (community)
profile: dev
persistence:
  type: file
  work_dir: .taskforce

# After (enterprise)
profile: enterprise
persistence:
  type: database
  database_url: ${DATABASE_URL}

enterprise:
  multi_tenant: true
  encryption:
    enabled: true
```

Step 3: Authentication Setup

```
# Configure your identity provider
# See Section 3.2 for detailed examples
```

Step 4: Role Assignment

```
# Assign existing users to appropriate roles
for user in existing_users:
    await user_manager.update_user(
        user_id=user.id,
        tenant_id="your-tenant",
        roles={"operator"}  # or appropriate role
    )
```

Step 5: Verification

```
# Verify enterprise features
uv run python scripts/test_enterprise_features.py
```

Related Documentation

- Architecture Overview
- API Reference
- CLI Guide
- Configuration Profiles
- ADR-003: Enterprise Transformation

Questions or feedback? Create an issue in the repository.