

warning "Mise à jour du backend (octobre 2026)" L'inférence GitHub Models a été retirée le 30 juillet 2026. Les références ci-dessous sont conservées uniquement comme contexte historique pendant la migration de ce lab. Utilisez le Lab 013 ? Backends LLM gratuits pour configurer Foundry Local ou Ollama avant de continuer.

Lab 046 : Microsoft Agent 365 ? Gouvernance des agents en entreprise

Niveau : L300 Parcours : ?? Pro Code Durée : ~75 min ? Coût : Payant ? Licence Microsoft 365 Copilot + abonnement Azure + programme Frontier

Ce que vous apprendrez

Ce qu'est Microsoft Agent 365 et en quoi il diffère des **frameworks** d'agents Comment Agent 365 attribue à chaque agent IA sa propre Entra Agent Identity Comment installer et configurer le CLI Agent 365 Comment créer un Agent Blueprint (le modèle de gouvernance d'entreprise) Comment ajouter l'observabilité (OpenTelemetry) à un agent Python existant avec le SDK Agent 365 Comment publier un agent dans le Centre d'administration Microsoft 365 Comment créer une instance d'agent qui apparaît dans l'organigramme dans Teams

Introduction

La plupart des frameworks d'agents IA se concentrent sur la **construction** d'agents ? comment ils raisonnent, appellent des outils et mémorisent le contexte. Microsoft Agent 365 résout un problème différent : comment les entreprises gouvernent, sécurisent et gèrent les agents à grande échelle ?

Considérez Agent 365 comme le **plan de contrôle** des agents IA dans votre tenant Microsoft 365 :

Plan de contrôle Agent 365

Un agent amélioré avec Agent 365 obtient :

Capacité | Ce que cela signifie | Entra Agent ID | Sa propre identité dans Azure AD ? comme un compte utilisateur, mais pour un agent | Blueprint | Modèle approuvé par l'IT définissant les capacités, les permissions MCP et les politiques de gouvernance | Notifications | Peut recevoir et répondre aux @mentions dans Teams, aux e-mails, aux commentaires Word | Outils MCP gouvernés | Accès à Mail, Calendrier, Teams, SharePoint sous contrôle administrateur | Observabilité | Traces OpenTelemetry complètes de chaque appel d'outil et inférence LLM |

warning "Programme Preview requis" Microsoft Agent 365 est en preview Frontier. Vous devez rejoindre le Microsoft Copilot Frontier Program et disposer d'au moins une licence Microsoft 365 Copilot dans votre tenant.

Architecture : couches d'Agent 365

Architecture en couches d'Agent 365

Le SDK Agent 365 se situe **au-dessus** de votre framework d'agent. Il ne le remplace pas ? il l'enveloppe et l'améliore.

Prérequis

[] Licence Microsoft 365 Copilot (au moins 1 dans votre tenant) [] Accès au programme

Frontier ? inscrivez-vous ici [] Abonnement Azure ? droits de création de ressources []

Permissions Entra ID ? Rôle Administrateur global, Administrateur Agent ID ou Développeur

Agent ID [] .NET 8.0 ou ultérieur ? pour le CLI Agent 365 [] Python 3.11+ et pip []

Jeton GitHub Models (gratuit) ? pour l'agent OutdoorGear que nous améliorerons

tip "Pas d'accès Frontier ? Suivez quand même" Si vous n'avez pas encore accès à Frontier, vous pouvez toujours suivre les étapes 1?4 localement avec des outils simulés. Le fichier de démarrage inclut un mode simulé. Les étapes 5?7 nécessitent un vrai tenant.

tip "Démarrage rapide avec GitHub Codespaces" Ouvrir dans GitHub Codespaces

Toutes les dépendances sont préinstallées dans le devcontainer.

? Fichiers d'accompagnement

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-046/ dans votre répertoire de travail.

Fichier | Description | Télécharger | a365.config.sample.json | Fichier de configuration / données | ? Télécharger | broken_observability.py | Exercice de correction de bugs (3 bugs + auto-tests) | ? Télécharger | outdoorgear_a365_starter.py | Script de démarrage avec TODOs | ? Télécharger |

Étape 1 : Installer le CLI Agent 365

Le CLI Agent 365 (a365) est l'épine dorsale en ligne de commande pour l'ensemble du cycle de vie de développement des agents.

[diagram or code sample]

Sortie attendue : [diagram or code sample]

note "Utilisez toujours --prerelease" Tant qu'Agent 365 n'est pas en GA, incluez toujours --prerelease dans les commandes d'installation/mise à jour. Sans cela, le package ne sera pas trouvé dans les flux NuGet.

Étape 2 : Enregistrer une application cliente personnalisée dans Entra ID

Le CLI a besoin de son propre enregistrement d'application Entra pour s'authentifier auprès de votre tenant.

Dans le portail Azure ? Entra ID ? Inscriptions d'applications :

1. Cliquez sur Nouvelle inscription 2. Nom : Agent365-CLI-App 3. Types de comptes pris en charge : Comptes dans cet annuaire organisationnel uniquement 4. Cliquez sur Inscrire 5. Copiez l'ID d'application (client) ? vous en aurez besoin à l'étape 3 6. Allez dans Permissions d'API ? Ajouter une permission ? Microsoft Graph 7. Ajoutez ces Permissions d'application : - AgentLifecycle.ReadWrite.All - Application.ReadWrite.All 8. Cliquez sur Accorder le consentement administrateur

[diagram or code sample]

Étape 3 : Initialiser la configuration Agent 365

[diagram or code sample]

Le CLI vous demandera : L'ID du tenant L'ID de l'abonnement Azure Le nom du groupe de ressources L'ID de l'application cliente de l'étape 2 L'URL du point de terminaison de messagerie de votre agent

Cela crée un fichier a365.config.json dans le répertoire de votre projet :

[diagram or code sample]

tip "Utilisez l'exemple de configuration" Le lab inclut lab-046/a365.config.sample.json comme référence. Copiez-le, remplissez vos valeurs, renommez-le en a365.config.json.

Étape 4 : Ajouter le SDK Agent 365 à votre agent Python

Installez les packages du SDK Agent 365 :

[diagram or code sample]

4a. Ajouter l'observabilité (OpenTelemetry)

Le SDK instrumente automatiquement votre agent pour le SDK OpenAI Agents :

[diagram or code sample]

4b. Ajouter les outils MCP gouvernés

Connectez-vous aux serveurs MCP Microsoft 365 (Mail, Calendrier, Teams) sous contrôle administrateur :

[diagram or code sample]

4c. Ajouter les notifications (Teams / Outlook)

Faites en sorte que votre agent réponde aux @mentions et aux e-mails :

[diagram or code sample]

Étape 5 : Créer l'Agent Blueprint

Le blueprint est le modèle d'entreprise approuvé par l'IT pour votre agent. Créez-le avec une seule commande CLI :

[diagram or code sample]

Cette commande : 1. Crée un Azure Entra Agent ID (principal de service pour votre agent) 2. Enregistre les permissions d'outils MCP de l'agent telles que définies dans a365.config.json 3. Crée l'Agent Blueprint dans Azure 4. Génère un blueprintId dont vous aurez besoin pour la publication

[diagram or code sample]

Étape 6 : Déployer le code de l'agent sur Azure

Si vous n'avez pas de déploiement Azure existant :

[diagram or code sample]

Votre agent doit être accessible à l'URL du point de terminaison de messagerie que vous avez définie dans a365.config.json.

Étape 7 : Publier dans le Centre d'administration Microsoft 365

[diagram or code sample]

Après la publication :

1. Allez dans le Centre d'administration Microsoft 365 ? Agents 2. Trouvez OutdoorGearAgent dans le registre 3. Cliquez sur Créer une instance pour instancier l'agent pour votre organisation 4. L'agent obtient : - Sa propre entrée dans votre organigramme - Une adresse e-mail (outdoorgear-agent@yourorg.com) - La possibilité d'être @mentionné dans Teams - Une visibilité dans la carte des agents (qui l'utilise, quelles données il accède) info "L'agent apparaît dans Teams en quelques minutes" Après la création d'une instance, votre agent apparaît dans la recherche Teams. Les utilisateurs peuvent le @mentionner dans les conversations et les canaux. Il apparaît également dans les contacts Outlook.

? Exercice de correction de bugs : Corriger la configuration d'observabilité cassée

Le lab inclut une configuration d'observabilité cassée. Trouvez et corrigez 3 bugs !

[diagram or code sample]

Installation : [diagram or code sample]

Les 3 bugs :

| Composant | Symptôme | Type | 1 | A365ObservabilityProvider | TypeError: missing required argument 'service_name' | Paramètre requis manquant | 2 |

OpenAIAgentInstrumentation | Les traces affichent service_name: unknown au lieu de OutdoorGearAgent | Provider non passé à l'instrumentation | 3 | Point de terminaison de l'exporteur | ConnectionRefusedError: localhost:4317 | Mauvais point de terminaison ? devrait utiliser un collecteur HTTPS |

Vérifiez vos corrections : Après avoir corrigé les 3 bugs, exécutez : [diagram or code sample]

? Quiz de connaissances

question "Q1 (Choix multiple) : Un agent construit avec LangChain souhaite utiliser Microsoft Agent 365. Qu'est-ce qu'Agent 365 fournit que LangChain ne fournit PAS ?"

- A) La capacité d'appeler des API et des outils externes - B) Le raisonnement et la planification en plusieurs étapes - C) Une identité Entra, des outils MCP gouvernés, l'observabilité et la gouvernance/conformité d'entreprise - D) Un meilleur modèle LLM pour le raisonnement

??? success "? Révéler la réponse" Correct : C

Agent 365 n'est PAS un framework d'agent ? il n'aide pas à construire la logique de raisonnement. LangChain gère déjà l'appel d'outils, le raisonnement en plusieurs étapes et

la planification. Ce qu'Agent 365 ajoute, c'est la couche entreprise : une identité Entra unique pour l'agent, un accès MCP gouverné approuvé par l'IT aux charges de travail M365, l'observabilité OpenTelemetry, la gouvernance basée sur les blueprints, et la capacité pour les administrateurs IT de voir, surveiller et contrôler l'agent depuis le Centre d'administration M365.

question "Q2 (Choix multiple) : Qu'est-ce qu'un Agent Blueprint dans Microsoft Agent 365 ?"

- A) Un modèle Bicep/ARM pour déployer des ressources Azure - B) Un modèle préconfiguré approuvé par l'IT définissant les capacités d'un agent, les permissions MCP, les politiques de gouvernance et les contraintes de conformité - C) Une classe Python dont tous les agents Agent 365 doivent hériter - D) Un diagramme du flux d'appels d'outils de l'agent

??? success "?" Révéler la réponse" Correct : B

Un blueprint provient de Microsoft Entra et constitue le *modèle d'entreprise* à partir duquel des instances d'agents conformes sont créées. Il définit ce que l'agent peut faire (capacités), quelles données M365 il peut accéder (permissions MCP), comment il est gouverné (politiques DLP, restrictions d'accès externe, règles de journalisation) et les métadonnées de cycle de vie. Chaque instance d'agent créée à partir du blueprint hérite de toutes ces règles ? garantissant qu'aucun « agent fantôme » n'a un accès non contrôlé.

question "Q3 (Exécutez le lab) : Ouvrez lab-046/outdoorgear_a365_starter.py. Combien de commentaires TODO y a-t-il dans le fichier ?"

Ouvrez le fichier de démarrage et comptez les marqueurs # TODO.

??? success "?" Révéler la réponse" 5 TODOs

Le fichier de démarrage comporte 5 points d'intégration à compléter : 1. TODO 1 :

Initialiser A365ObservabilityProvider avec le nom et la version du service 2. TODO 2 :

Appliquer OpenAIAgentInstrumentation pour auto-instrumenter les traces 3. TODO 3 :

Implémenter le gestionnaire on_teams_mention 4. TODO 4 : Se connecter aux serveurs d'outils MCP gouvernés 5. TODO 5 : Enregistrer l'Entra Agent ID de l'agent

question "Q4 (Choix multiple) : Un utilisateur @mentionne votre agent OutdoorGear dans un canal Teams. Quel composant du SDK Agent 365 reçoit et achemine cette notification vers votre code d'agent ?"

- A) A365ObservabilityProvider - B) A365ToolingClient - C) A365NotificationHandler - D) OpenAIMcpRegistrationService

??? success "?" Révéler la réponse" Correct : C ? A365NotificationHandler

Le A365NotificationHandler reçoit les événements des applications Microsoft 365 ? @mentions Teams, e-mails entrants dans la boîte aux lettres de l'agent, notifications de commentaires Word, et plus encore. Vous le sous-classez et redéfinissez des méthodes comme on_teams_mention() et on_email_received(). Le A365ObservabilityProvider gère la télémétrie, A365ToolingClient gère l'accès aux outils MCP, et OpenAIMcpRegistrationService enregistre les serveurs MCP avec le SDK OpenAI Agents.

Résumé

Concept | Point clé | Agent 365 ? framework d'agent | Il ajoute des *capacités d'entreprise* par-dessus votre agent existant ? ne remplace pas SK, LangChain, etc. | Entra Agent ID | Chaque agent obtient sa propre identité ? comme un compte utilisateur mais pour un agent | Blueprint | Modèle approuvé par l'IT ; toutes les instances héritent de ses règles de gouvernance | Observabilité | Auto-instrumentation OpenTelemetry ? chaque appel d'outil et inférence LLM est tracé | MCP gouverné | Outils M365 (Mail, Calendrier, Teams, SharePoint) accessibles sous contrôle IT | Notifications | Les agents peuvent être @mentionnés dans Teams, recevoir des e-mails, répondre aux commentaires Word | Frontier requis | Encore en preview ? nécessite une licence M365 Copilot + inscription au programme

Frontier |

Prochaines étapes

Construisez d'abord l'agent sous-jacent : ? Lab 016 ? OpenAI Agents SDK Ajoutez des outils MCP à votre agent : ? Lab 020 ? Serveur MCP en Python Approfondissement de l'observabilité : ? Lab 033 ? Observabilité des agents avec App Insights Pipeline RAG d'entreprise : ? Lab 042 ? RAG d'entreprise avec évaluations *(bientôt disponible)*

Modernisation de Phase 3

warning "Verified 2026-10-27" Utilisez Entra Agent ID et Agent 365 GA à la frontière identité-gouvernance. Séparez identité agent, délégation, permissions, inventaire, politique et audit.

[diagram or code sample]

Microsoft Entra Agent ID GA Microsoft Agent 365 GA

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré

Supporting file: a365.config.sample.json

```
{ "_comment": "Copy this file to a365.config.json and fill in your values.", "tenantId":  
"YOUR_ENTRA_TENANT_ID", "subscriptionId": "YOUR_AZURE_SUBSCRIPTION_ID", "resourceGroup":  
"rg-outdoorgear-agent", "clientAppId": "YOUR_CLI_CLIENT_APP_ID", "agentMessagingEndpoint":  
"https://outdoorgear-agent.azurewebsites.net/api/messages", "agentBlueprintName":  
"OutdoorGearAgent", "agentDisplayName": "OutdoorGear Assistant", "agentDescription": "AI  
agent that helps employees find outdoor gear products, check inventory, and process  
orders.", "mcpPermissions": [ "mail.read", "calendar.readwrite", "teams.message.send" ],  
"observability": { "enabled": true, "exporterEndpoint":  
"https://your-otel-collector.endpoint/v1/traces", "serviceName": "OutdoorGearAgent",  
"serviceVersion": "1.0.0" } }
```

Supporting file: broken_observability.py

==== Broken Observability Setup ? Bug-Fix Exercise

===== Lab 046: Microsoft Agent 365 ? Enterprise

Agent Governance

This file has 3 intentional bugs in the Agent 365 observability setup. Find and fix them all, then run the self-test at the bottom.

Run with: python lab-046/broken_observability.py

?? Mock Agent 365 SDK classes (simulates real SDK for the exercise) ?????????? In the real lab you'd import from microsoft.agents.a365.*

```
class A365ObservabilityProvider: """Simulated Agent 365 observability provider."""  
def __init__(self, service_name: str, service_version: str, exporter_endpoint: str):  
    if not service_name: raise ValueError("service_name is required and cannot be empty")  
    if not exporter_endpoint.startswith("https://"): raise ConnectionRefusedError( f"Invalid exporter  
endpoint: '{exporter_endpoint}'. " f"Must start with https:// (got: {exporter_endpoint})" )  
    self.service_name = service_name  
    self.service_version = service_version  
    self.exporter_endpoint = exporter_endpoint  
    self._initialized = True  
    @property  
    def is_initialized(self): return self._initialized  
class OpenAIAgentInstrumentation: """Simulated OpenAI Agents SDK instrumentation for Agent  
365."""
```

```

def __init__(self, provider: A365ObservabilityProvider): if not isinstance(provider,
A365ObservabilityProvider): raise TypeError( f"provider must be an
A365ObservabilityProvider instance, " f"got {type(provider).__name__}" ) if not
provider.is_initialized: raise RuntimeError("ObservabilityProvider must be initialized
before instrumenting") self._provider = provider self._instrumented = False
def instrument(self): self._instrumented = True
@property def service_name(self): return self._provider.service_name if self._instrumented
else "unknown"
@property def is_active(self): return self._instrumented
?? BUGGY setup code ? find and fix 3 bugs! ????????????????????????????????? The
developer wrote this setup code with 3 mistakes. The bugs are marked with comments but NOT
all in obvious places. Read carefully!
def setup_observability(): """ Initialize Agent 365 observability for the OutdoorGear Agent.
BUG HUNT: - Bug #1: A365ObservabilityProvider is initialized incorrectly ? missing a
required argument - Bug #2: OpenAIInstrumentation is not receiving the provider it
needs - Bug #3: The exporter endpoint is wrong ? it uses localhost instead of HTTPS """
# BUG #1 is here ? look carefully at the arguments observability =
A365ObservabilityProvider( service_version="1.0.0",
exporter_endpoint="https://my-otel-collector.azure.com/v1/traces" # Missing:
service_name="OutdoorGearAgent" )
# BUG #2 is here ? the instrumentation is not connected to the provider instrumentation =
OpenAIInstrumentation(provider=None) # should be: provider=observability
instrumentation.instrument()
# BUG #3 is here ? wrong endpoint scheme bad_endpoint_observability =
A365ObservabilityProvider( service_name="OutdoorGearAgent", service_version="1.0.0",
exporter_endpoint="localhost:4317" # should start with https:// )
return observability, instrumentation
?? Self-test runner ?????????????????????????????????????????????????????????
def run_tests(): """Run all tests. After fixing the 3 bugs, all tests should pass."""
passed = 0 failed = 0
print("=" * 55) print(" Agent 365 Observability ? Bug-Fix Self-Test") print("=" * 55)
# Test 1: Provider initializes with service_name print("\nTest 1: A365ObservabilityProvider
has correct service_name...") try: provider = A365ObservabilityProvider(
service_name="OutdoorGearAgent", service_version="1.0.0",
exporter_endpoint="https://my-collector.azure.com/v1/traces" ) assert provider.service_name
== "OutdoorGearAgent", \ f"Expected 'OutdoorGearAgent', got '{provider.service_name}'"
print(" ? Passed ? service_name = OutdoorGearAgent") passed += 1 except Exception as e:
print(f" ? Failed ? {e}") failed += 1
# Test 2: Instrumentation correctly reflects service_name print("\nTest 2: Instrumentation
reports service_name: OutdoorGearAgent...") try: provider = A365ObservabilityProvider(
service_name="OutdoorGearAgent", service_version="1.0.0",
exporter_endpoint="https://my-collector.azure.com/v1/traces" ) instr =
OpenAIInstrumentation(provider=provider) instr.instrument() assert instr.service_name
== "OutdoorGearAgent", \ f"Expected 'OutdoorGearAgent', got '{instr.service_name}'" assert
instr.is_active, "Instrumentation should be active after .instrument()" print(" ? Passed ?
service_name correctly propagated from provider") passed += 1 except Exception as e:
print(f" ? Failed ? {e}") failed += 1
# Test 3: Endpoint validation print("\nTest 3: Exporter endpoint must start with

```

```

https://...) try: # This should FAIL (raise ConnectionRefusedError) bad =
A365ObservabilityProvider( service_name="test", service_version="1.0.0",
exporter_endpoint="localhost:4317" ) print(" ? Failed ? should have raised
ConnectionRefusedError for localhost endpoint") failed += 1 except ConnectionRefusedError:
print(" ? Passed ? correctly rejected localhost endpoint") passed += 1 except Exception as
e: print(f" ? Failed ? unexpected error: {e}") failed += 1
# Test 4: Full happy-path setup print("\nTest 4: Full observability setup (no bugs)...")
try: provider = A365ObservabilityProvider( service_name="OutdoorGearAgent",
service_version="1.0.0", exporter_endpoint="https://otel-collector.azure.com/v1/traces" )
instr = OpenAIInstrumentation(provider=provider) instr.instrument() assert
provider.is_initialized assert instr.is_active assert instr.service_name ==
"OutdoorGearAgent" print(" ? Passed ? full setup works correctly") passed += 1 except
Exception as e: print(f" ? Failed ? {e}") failed += 1
# Summary print("\n" + "=" * 55) print(f" Results: {passed} passed, {failed} failed")
print("=" * 55)
if failed == 0: print("\n? All tests passed! Your observability setup is bug-free.")
print("\nWhat you fixed:") print(" Bug #1: Added missing service_name='OutdoorGearAgent' to
A365ObservabilityProvider") print(" Bug #2: Changed provider=None to provider=observability
in OpenAIInstrumentation") print(" Bug #3: Changed 'localhost:4317' to a valid
https:// endpoint") else: print(f"\n?? {failed} test(s) still failing. Keep debugging!")
print("\nHints:") print(" Bug #1: Look at what arguments A365ObservabilityProvider __init__
requires") print(" Bug #2: What should be passed to
OpenAIInstrumentation(provider=...)?") print(" Bug #3: What scheme does a valid OTEL
exporter endpoint require?")
return failed == 0
if __name__ == "__main__": print("Running tests on BUGGY setup_observability()...")
print("(The tests test the CORRECTED behavior ? your job is to fix setup_observability)\n")
# First, demonstrate the bugs print("Current buggy setup_observability() output:") try:
obs, instr = setup_observability() print(f" observability.service_name = '{getattr(obs,
'service_name', 'MISSING')}'") print(f" instrumentation.service_name =
'{instr.service_name}'") except Exception as e: print(f" ? Error in setup_observability():
{e}")
print("\n" + "-" * 55) print("Running correctness tests (these define the target
behavior):") print("-" * 55)
run_tests()

```

Supporting file: outdoorgear_a365_starter.py

""" OutdoorGear Agent ? Agent 365 Starter File =====

Lab 046: Microsoft Agent 365 ? Enterprise Agent Governance

This file gives you a working OutdoorGear agent skeleton with 5 TODOs for integrating

Microsoft Agent 365 SDK capabilities: - TODO 1: Initialize A365ObservabilityProvider - TODO

2: Apply OpenAI auto-instrumentation - TODO 3: Implement on_teams_mention handler - TODO 4:

Connect to governed MCP tooling servers - TODO 5: Register Entra Agent ID

Run in mock mode (no Frontier access required): AGENT_365 MOCK=true python

outdoorgear_a365_starter.py

Run with real Agent 365: AGENT_365 MOCK=false python outdoorgear_a365_starter.py """

import os import asyncio from typing import Optional

?? Agent 365 SDK imports (install: pip install microsoft-agents-a365-*) ?????? try: from

```

microsoft.agents.a365.observability.core import A365ObservabilityProvider from
microsoft.agents.a365.observability.extensions.openai import OpenAIAgentInstrumentation
from microsoft.agents.a365.notifications import A365NotificationHandler from
microsoft.agents.a365.tooling import A365ToolingClient from
microsoft.agents.a365.tooling.extensions.openai import OpenAIMcpRegistrationService
AGENT_365_AVAILABLE = True except ImportError: AGENT_365_AVAILABLE = False print("?? Agent
365 SDK not installed. Running in mock mode.") print(" Install: pip install
microsoft-agents-a365-observability-core "
"microsoft-agents-a365-observability-extensions-openai "
"microsoft-agents-a365-notifications microsoft-agents-a365-tooling\n")
?? OpenAI Agents SDK ????????????????????????????????????????????????????????? try: from
openai import AsyncOpenAI from agents import Agent, Runner, function_tool
OPENAI_AGENTS_AVAILABLE = True except ImportError: OPENAI_AGENTS_AVAILABLE = False
print("?? OpenAI Agents SDK not installed.") print(" Install: pip install openai
openai-agents\n")
?? Configuration ????????????????????????????????????????????????????????? MOCK_MODE =
os.environ.get("AGENT_365_MOCK", "true").lower() == "true" GITHUB_TOKEN =
os.environ.get("GITHUB_TOKEN", "") TENANT_ID = os.environ.get("AZURE_TENANT_ID",
"mock-tenant-id") ENTRA_AGENT_ID = os.environ.get("ENTRA_AGENT_ID", "mock-agent-id")
OTEL_ENDPOINT = os.environ.get("OTEL_ENDPOINT", "https://mock-collector.endpoint/v1/traces")
?? OutdoorGear product catalog ????????????????????????????????????????????????????????? PRODUCTS =
[ {"id": "P001", "name": "TrailBlazer Tent 2P", "category": "tent", "price": 249.99,
"in_stock": True}, {"id": "P002", "name": "Summit Dome 4P", "category": "tent", "price":
549.99, "in_stock": True}, {"id": "P003", "name": "TrailBlazer Solo", "category": "tent",
"price": 299.99, "in_stock": False}, {"id": "P004", "name": "ArcticDown -20°C", "category":
"sleeping bag", "price": 389.99, "in_stock": True}, {"id": "P005", "name": "SummerLight
+5°C", "category": "sleeping bag", "price": 149.99, "in_stock": True}, {"id": "P006",
"name": "Osprey Atmos 65L", "category": "backpack", "price": 289.99, "in_stock": True},
{"id": "P007", "name": "DayHiker 22L", "category": "backpack", "price": 89.99, "in_stock":
True}, ]
?? Agent tools ????????????????????????????????????????????????????????? def
search_products(query: str, category: Optional[str] = None) -> list: """Search OutdoorGear
products by keyword and optional category.""" results = PRODUCTS if category: results = [p
for p in results if category.lower() in p["category"].lower()] if query: results = [p for p
in results if query.lower() in p["name"].lower() or query.lower() in p["category"].lower()]
return results
def get_product_details(product_id: str) -> dict: """Get full details for a specific
product by ID.""" for p in PRODUCTS: if p["id"].upper() == product_id.upper(): return p
return {"error": f"Product {product_id} not found"}
def check_inventory(category: str) -> dict: """Get inventory summary for a product
category.""" items = [p for p in PRODUCTS if category.lower() in p["category"].lower()]
in_stock = [p for p in items if p["in_stock"]] out_of_stock = [p for p in items if not
p["in_stock"]] return { "category": category, "total": len(items), "in_stock":
len(in_stock), "out_of_stock": len(out_of_stock), "items": items }
???????????????????????????????????????????????????????????????????????????????? TODO 1:
Initialize A365ObservabilityProvider
???????????????????????????????????????????????????????????????????????????????? Replace this
mock with real initialization: observability = A365ObservabilityProvider(

```



```
policies defined in your Blueprint. tooling_client = A365ToolingClient(
agent_id=ENTRA_AGENT_ID, tenant_id=TENANT_ID ) mcp_servers = await
tooling_client.get_mcp_servers() registration_service = OpenAIMcpRegistrationService(agent)
for server in mcp_servers: await registration_service.register(server)
async def connect_m365_tools(agent): """TODO 4: Connect governed MCP tools to agent.""" if
MOCK_MODE or not AGENT_365_AVAILABLE: print("[MOCK] M365 MCP tools connected (mock):
mail.read, calendar.readwrite, teams.message.send") return agent
# TODO 4: Uncomment and fill in: # tooling_client = A365ToolingClient( #
agent_id=ENTRA_AGENT_ID, # tenant_id=TENANT_ID # ) # mcp_servers = await
tooling_client.get_mcp_servers() # registration_service =
OpenAIMcpRegistrationService(agent) # for server in mcp_servers: # await
registration_service.register(server) raise NotImplementedError("TODO 4: Connect M365 MCP
tools")
????????????????????????????????????????????????????????????????????????????????????? TODO 5:
Register Entra Agent ID ??????????????????????????????????????????????????????????????????????????????????????
??????????? Tell the Agent 365 runtime which Entra Agent ID this agent uses. This ties the
agent's identity to the blueprint and observability traces. In real code, typically done
via environment variable ENTRA_AGENT_ID or by reading from a365.config.json at startup.
def register_agent_identity(agent_id: str = ENTRA_AGENT_ID): """TODO 5: Register the Entra
Agent ID for this agent instance.""" if MOCK_MODE: print(f"[MOCK] Entra Agent ID
registered: {agent_id}") return agent_id
# TODO 5: In production, validate the agent_id against your tenant: # from
microsoft.agents.a365.runtime import A365Runtime # runtime =
A365Runtime(tenant_id=TENANT_ID) # runtime.set_agent_id(agent_id) # return
runtime.validate_agent_id()
print(f"[INFO] Entra Agent ID: {agent_id}") return agent_id
????????????????????????????????????????????????????????????????????????????????????? Main agent
setup & demo ??????????????????????????????????????????????????????????????????????????????????????
async def build_agent(): """Build and configure the OutdoorGear Agent with Agent 365
capabilities."""
print("=" * 60) print(" OutdoorGear Agent ? Microsoft Agent 365 Starter") print(f" Mode:
{'? MOCK' if MOCK_MODE else '? PRODUCTION'}") print("=" * 60)
# Step 1 & 2: Observability observability = init_observability()
apply_instrumentation(observability)
# Step 5: Register identity agent_id = register_agent_identity()
if not OPENAI_AGENTS_AVAILABLE: print("\n?? OpenAI Agents SDK not available ? skipping
agent creation") print(" Install: pip install openai openai-agents") return None, None
# Build the agent with OutdoorGear tools client = AsyncOpenAI(
base_url="https://models.inference.ai.azure.com", api_key=GITHUB_TOKEN or "mock-key" ) if
GITHUB_TOKEN else None
# Define tools for the agent tools = [ function_tool(search_products),
function_tool(get_product_details), function_tool(check_inventory), ]
agent = Agent( name="OutdoorGearAgent", instructions="""You are the OutdoorGear Assistant,
an enterprise AI agent for the OutdoorGear retail chain. Help employees find products,
check inventory, and answer product questions. You are governed by Microsoft Agent 365,
which means your actions are logged, auditable, and controlled by IT policies.
Always be concise and factual. Only answer questions about OutdoorGear products.""",
tools=tools, model="gpt-4o-mini" )
```

```

# Step 4: Connect M365 tools (mock in demo mode) if MOCK_MODE: print("[MOCK] M365 MCP tools
connected (mock): mail.read, calendar.readwrite, teams.message.send") else: agent = await
connect_m365_tools(agent)
# Create notification handler handler = OutdoorGearNotificationHandler(agent)
print(f"\n? Agent built: {agent.name}") print(f" Entra Agent ID: {agent_id}") print(f"
Tools: {[t.name for t in tools]}") print(f" Governed MCP: mail.read, calendar.readwrite,
teams.message.send")
return agent, handler
async def run_demo(agent, handler): """Run a mock demo simulating Teams @mentions and
direct queries."""
if not agent: return
print("\n" + "=" * 60) print(" Demo: Simulated Agent 365 Interactions") print("=" * 60)
# Simulate a Teams @mention print("\n? Simulating Teams @mention...") await
handler.on_teams_mention( message="What tents do you have in stock under $300?" )
# Simulate a direct query (would come through Runner in production) print("\n? Direct agent
query (local test)...") if GITHUB_TOKEN: result = await Runner.run( agent, input="How many
backpacks are in stock?" ) print(f"Agent response: {result.final_output}") else:
print("[SKIPPED] No GITHUB_TOKEN ? set GITHUB_TOKEN to test real queries") # Manually call
the tool to demonstrate inventory = check_inventory("backpack") print(f"[MOCK ANSWER]
Backpack inventory: {inventory['in_stock']} in stock, " f"{inventory['out_of_stock']} out

```