

warning "Atualização de backend (outubro de 2026)" A inferência do GitHub Models foi descontinuada em 30 de julho de 2026. As referências abaixo são mantidas apenas como contexto histórico enquanto este lab é migrado. Use o Lab 013 ? Backends de LLM gratuitos para configurar o Foundry Local ou o Ollama antes de continuar.

Lab 046: Microsoft Agent 365 ? Governança de Agentes Empresariais

Nível: L300 Trilha: ?? Pro Code Tempo: ~75 min ? Custo: Pago ? Licença Microsoft 365

Copilot + assinatura Azure + programa Frontier

O que Você Vai Aprender

O que é o Microsoft Agent 365 e como ele difere de *frameworks* de agentes Como o Agent 365 dá a cada agente de IA sua própria Identidade de Agente Entra Como instalar e configurar a CLI do Agent 365 Como criar um Agent Blueprint (o modelo de governança empresarial) Como adicionar observabilidade (OpenTelemetry) a um agente Python existente usando o SDK do Agent 365 Como publicar um agente no Centro de Administração do Microsoft 365 Como criar uma instância de agente que aparece no organograma no Teams

Introdução

A maioria dos frameworks de agentes de IA foca em *construir* agentes ? como eles raciocinam, chamam ferramentas e lembram o contexto. O Microsoft Agent 365 resolve um problema diferente: como as empresas governam, protegem e gerenciam agentes em escala? Pense no Agent 365 como o *plano de controle* para agentes de IA no seu tenant do Microsoft 365:

Agent 365 Control Plane

Um agente aprimorado com o Agent 365 obtém:

Capacidade | O que significa | Entra Agent ID | Sua própria identidade no Azure AD ? como uma conta de usuário, mas para um agente | Blueprint | Modelo aprovado pelo TI definindo capacidades, permissões MCP e políticas de governança | Notificações | Pode receber e responder a @menções no Teams, e-mails, comentários no Word | Ferramentas MCP Governadas | Acesso a Mail, Calendar, Teams, SharePoint sob controle administrativo | Observabilidade | Rastreamentos completos via OpenTelemetry de cada chamada de ferramenta e inferência LLM | warning "Programa preview obrigatório" O Microsoft Agent 365 está em preview Frontier. Você precisa participar do Microsoft Copilot Frontier Program e ter pelo menos uma licença Microsoft 365 Copilot no seu tenant.

Arquitetura: Camadas do Agent 365

Agent 365 Layered Architecture

O SDK do Agent 365 fica *acima* do seu framework de agentes. Ele não o substitui ? ele envolve e aprimora.

Pré-requisitos

[] Licença Microsoft 365 Copilot (pelo menos 1 no seu tenant) [] Acesso ao programa

Frontier ? inscreva-se aqui [] Assinatura Azure ? direitos de criação de recursos []

Permissões Entra ID ? Administrador Global, Administrador de ID de Agente ou função de Desenvolvedor de ID de Agente [] .NET 8.0 ou posterior ? para a CLI do Agent 365 []

Python 3.11+ e pip [] Token GitHub Models (gratuito) ? para o agente OutdoorGear que vamos aprimorar

tip "Sem acesso ao Frontier? Acompanhe mesmo assim" Se você ainda não tem acesso ao Frontier, pode seguir os Passos 1?4 localmente com ferramentas simuladas. O arquivo inicial inclui um modo simulado. Os Passos 5?7 requerem um tenant real.

tip "Início Rápido com GitHub Codespaces" Open in GitHub Codespaces

Todas as dependências estão pré-instaladas no devcontainer.

? Arquivos de Suporte

note "Baixe estes arquivos antes de iniciar o lab" Salve todos os arquivos em uma pasta lab-046/ no seu diretório de trabalho.

Arquivo | Descrição | Download | a365.config.sample.json | Arquivo de configuração / dados | ? Download | broken_observability.py | Exercício de correção de bugs (3 bugs + auto-testes) | ? Download | outdoorgear_a365_starter.py | Script inicial com TODOs | ? Download |

Passo 1: Instalar a CLI do Agent 365

A CLI do Agent 365 (a365) é a espinha dorsal de linha de comando para todo o ciclo de vida de desenvolvimento de agentes.

[diagram or code sample]

Saída esperada: [diagram or code sample]

note "Sempre use --prerelease" Até que o Agent 365 seja GA, sempre inclua --prerelease nos comandos de instalação/atualização. Sem isso, o pacote não será encontrado nos feeds do NuGet.

Passo 2: Registrar um Aplicativo Cliente Personalizado no Entra ID

A CLI precisa de seu próprio registro de aplicativo Entra para autenticar contra o seu tenant.

No Portal Azure ? Entra ID ? Registros de aplicativo:

1. Clique em Novo registro 2. Nome: Agent365-CLI-App 3. Tipos de conta suportados: Contas apenas neste diretório organizacional 4. Clique em Registrar 5. Copie o ID do Aplicativo (cliente) ? você precisará dele no Passo 3 6. Vá para Permissões de API ? Adicionar uma permissão ? Microsoft Graph 7. Adicione estas Permissões de aplicativo: - AgentLifecycle.ReadWrite.All - Application.ReadWrite.All 8. Clique em Conceder consentimento do administrador

[diagram or code sample]

Passo 3: Inicializar a Configuração do Agent 365

[diagram or code sample]

A CLI solicitará: ID do Tenant ID da assinatura Azure Nome do grupo de recursos O ID do aplicativo cliente do Passo 2 A URL do endpoint de mensagens do seu agente

Isso cria um arquivo a365.config.json no diretório do seu projeto:

[diagram or code sample]

tip "Use o config de exemplo" O lab inclui lab-046/a365.config.sample.json como referência.

Copie-o, preencha seus valores e renomeie para a365.config.json.

Passo 4: Adicionar o SDK do Agent 365 ao Seu Agente Python

Instale os pacotes do SDK do Agent 365:

[diagram or code sample]

4a. Adicionar Observabilidade (OpenTelemetry)

O SDK instrumenta seu agente automaticamente para o OpenAI Agents SDK:

[diagram or code sample]

4b. Adicionar Ferramentas MCP Governadas

Conecte-se aos servidores MCP do Microsoft 365 (Mail, Calendar, Teams) sob controle administrativo:

[diagram or code sample]

4c. Adicionar Notificações (Teams / Outlook)

Faça seu agente responder a @menções e e-mails:

[diagram or code sample]

Passo 5: Criar o Agent Blueprint

O blueprint é o modelo empresarial aprovado pelo TI para o seu agente. Crie-o com um único

comando CLI:

[diagram or code sample]

Este comando: 1. Cria um Azure Entra Agent ID (service principal para o seu agente) 2.

Registra as permissões de ferramentas MCP do agente conforme definido em a365.config.json

3. Cria o Agent Blueprint no Azure 4. Retorna um blueprintId que você precisará para publicação

[diagram or code sample]

Passo 6: Implantar o Código do Agente no Azure

Se você não tem uma implantação Azure existente:

[diagram or code sample]

Seu agente deve estar acessível na URL do endpoint de mensagens que você definiu em a365.config.json.

Passo 7: Publicar no Centro de Administração do Microsoft 365

[diagram or code sample]

Após a publicação:

1. Vá para o Centro de Administração do Microsoft 365 ? Agentes 2. Encontre o

OutdoorGearAgent no registro 3. Clique em Criar instância para instanciar o agente para sua

organização 4. O agente recebe: - Sua própria entrada no organograma - Um endereço de

e-mail (outdoorgear-agent@yourorg.com) - Capacidade de ser @mencionado no Teams -

Visibilidade no Mapa de Agentes (quem o usa, quais dados ele acessa)

info "O agente aparece no Teams em minutos" Após criar uma instância, seu agente aparece na

busca do Teams. Os usuários podem @mencioná-lo em chats e canais. Ele também aparece nos contatos do Outlook.

? Exercício de Correção de Bugs: Corrigir a Configuração de Observabilidade Quebrada

O lab inclui uma configuração de observabilidade quebrada. Encontre e corrija 3 bugs!

[diagram or code sample]

Configuração: [diagram or code sample]

Os 3 bugs:

| Componente | Sintoma | Tipo | 1 | A365ObservabilityProvider | TypeError: missing

required argument 'service_name' | Parâmetro obrigatório ausente | 2 |

OpenAIAgentInstrumentation | Traces mostram service_name: unknown em vez de

OutdoorGearAgent | Provider não passado para a instrumentação | 3 | Endpoint do exportador

| ConnectionRefusedError: localhost:4317 | Endpoint errado ? deve usar coletor HTTPS |

Verifique suas correções: Após corrigir todos os 3 bugs, execute: [diagram or code sample]

? Verificação de Conhecimento

question "Q1 (Múltipla Escolha): Um agente construído com LangChain quer usar o Microsoft Agent 365. O que o Agent 365 fornece que o LangChain NÃO fornece?"

- A) A capacidade de chamar APIs e ferramentas externas - B) Raciocínio e planejamento em múltiplos passos - C) Identidade respaldada pelo Entra, ferramentas MCP governadas, observabilidade e governança/conformidade empresarial - D) Um modelo LLM melhor para raciocínio

??? success "? Revelar Resposta" Correto: C

O Agent 365 NÃO é um framework de agentes ? ele não ajuda a construir lógica de raciocínio.

O LangChain já lida com chamadas de ferramentas, raciocínio em múltiplos passos e

planejamento. O que o Agent 365 adiciona é a camada empresarial: uma identidade Entra única

para o agente, acesso MCP governado e aprovado pelo TI para cargas de trabalho do M365,

observabilidade OpenTelemetry, governança baseada em blueprints e a capacidade dos

administradores de TI verem, monitorarem e controlarem o agente a partir do Centro de

Administração do M365.

question "Q2 (Múltipla Escolha): O que é um Agent Blueprint no Microsoft Agent 365?"

- A) Um modelo Bicep/ARM para implantar recursos Azure - B) Um modelo pré-configurado e aprovado pelo TI que define as capacidades do agente, permissões MCP, políticas de governança e restrições de conformidade - C) Uma classe Python que todos os agentes do Agent 365 devem herdar - D) Um diagrama do fluxo de chamadas de ferramentas do agente
??? success "?" Revelar Resposta" Correto: B

Um blueprint vem do Microsoft Entra e é o *modelo empresarial* a partir do qual instâncias de agentes conformes são criadas. Ele define o que o agente pode fazer (capacidades), quais dados do M365 ele pode acessar (permissões MCP), como ele é governado (políticas DLP, restrições de acesso externo, regras de log) e metadados de ciclo de vida. Cada instância de agente criada a partir do blueprint herda todas essas regras ? garantindo que não haja "agentes sombra" com acesso descontrolado.

question "Q3 (Execute o Lab): Abra lab-046/outdoorgear_a365_starter.py. Quantos comentários TODO existem no arquivo?"

Abra o arquivo inicial e conte os marcadores # TODO.

??? success "?" Revelar Resposta" 5 TODOS

O arquivo inicial tem 5 pontos de integração para você completar: 1. TODO 1: Inicializar A365ObservabilityProvider com nome e versão do serviço 2. TODO 2: Aplicar OpenAI Agent Instrumentation para auto-instrumentar traces 3. TODO 3: Implementar o handler on_teams_mention 4. TODO 4: Conectar aos servidores de ferramentas MCP governadas 5. TODO 5: Registrar o Entra Agent ID do agente

question "Q4 (Múltipla Escolha): Um usuário @menciona seu Agente OutdoorGear em um canal do Teams. Qual componente do SDK do Agent 365 recebe e roteia essa notificação para o código do seu agente?"

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

OpenAIMcpRegistrationService

??? success "?" Revelar Resposta" Correto: C ? A365NotificationHandler

O A365NotificationHandler recebe eventos de aplicativos do Microsoft 365 ? @menções no Teams, e-mails recebidos na caixa de correio do agente, notificações de comentários no Word e muito mais. Você cria uma subclasse dele e sobrescreve métodos como on_teams_mention() e on_email_received(). O A365ObservabilityProvider lida com telemetria, o A365ToolingClient gerencia acesso a ferramentas MCP e o OpenAIMcpRegistrationService registra servidores MCP com o OpenAI Agents SDK.

Resumo

Conceito | Principal aprendizado | Agent 365 ? framework de agentes | Ele adiciona *capacidades empresariais* sobre o seu agente existente ? não substitui SK, LangChain, etc.
| Entra Agent ID | Cada agente recebe sua própria identidade ? como uma conta de usuário, mas para um agente | Blueprint | Modelo aprovado pelo TI; todas as instâncias herdam suas regras de governança | Observabilidade | Auto-instrumentação OpenTelemetry ? cada chamada de ferramenta e inferência LLM é rastreada | MCP Governado | Ferramentas do M365 (Mail, Calendar, Teams, SharePoint) acessíveis sob controle do TI | Notificações | Agentes podem ser @mencionados no Teams, receber e-mails, responder a comentários no Word | Frontier obrigatório | Ainda em preview ? requer licença M365 Copilot + inscrição no programa Frontier |

Próximos Passos

Construa o agente subjacente primeiro: ? Lab 016 ? OpenAI Agents SDK Adicione ferramentas MCP ao seu agente: ? Lab 020 ? MCP Server em Python Aprofundamento em observabilidade: ?

Lab 033 ? Observabilidade de Agentes com App Insights Pipeline RAG empresarial: ? Lab 042 ?

RAG Empresarial com Avaliações *(em breve)*

Modernização da Fase 3

warning "Verified 2026-10-27" Use Entra Agent ID e Agent 365 GA na fronteira de identidade e governança. Separe identidade do agente, delegação do usuário, permissões, inventário, política e auditoria.

[diagram or code sample]

Microsoft Entra Agent ID GA Microsoft Agent 365 GA

Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado

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: OpenAIAgentInstrumentation 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 =
OpenAIAgentInstrumentation(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 =
OpenAIAgentInstrumentation(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 = OpenAIAgentInstrumentation(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 OpenAIAgentInstrumentation") 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
OpenAIAgentInstrumentation(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

```

import logging
logging.basicConfig(level=logging.DEBUG)
from typing import Optional, List, Dict
from openai import AsyncOpenAI
from agents import Agent, Runner, function_tool
from microsoft_agents_a365_observability_core import A365ObservabilityProvider
from microsoft_agents_a365_notifications import MicrosoftAgentsA365Notifications
from microsoft_agents_a365_tooling import MicrosoftAgentsA365Tooling

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 ")

if AGENT_365_AVAILABLE:
    from microsoft_agents_a365_extensions import OpenAIRegistrationService
else:
    from microsoft_agents_a365_extensions import MockRegistrationService

try:
    from openai import AsyncOpenAI
except ImportError:
    print("?? OpenAI SDK not installed. Running in mock mode.")
    print(" Install: pip install openai ")

try:
    from agents import Agent, Runner, function_tool
except ImportError:
    print("?? Agents SDK not installed. Running in mock mode.")
    print(" Install: pip install agents ")

try:
    from microsoft_agents_a365_observability_core import A365ObservabilityProvider
except ImportError:
    print("?? A365 Observability SDK not installed. Running in mock mode.")
    print(" Install: pip install microsoft-agents-a365-observability-core ")

try:
    from microsoft_agents_a365_notifications import MicrosoftAgentsA365Notifications
except ImportError:
    print("?? A365 Notifications SDK not installed. Running in mock mode.")
    print(" Install: pip install microsoft-agents-a365-notifications ")

try:
    from microsoft_agents_a365_tooling import MicrosoftAgentsA365Tooling
except ImportError:
    print("?? A365 Tooling SDK not installed. Running in mock mode.")
    print(" Install: pip install microsoft-agents-a365-tooling ")

# 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}
]

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(
service_name="OutdoorGearAgent", service_version="1.0.0", exporter_endpoint=OTEL_ENDPOINT )

In mock mode, we use a simple print-based tracer:
class MockObservabilityProvider:
    def __init__(self, service_name: str):
        self.service_name = service_name
        print(f"[MOCK] ObservabilityProvider initialized: {service_name}")

```

[illegible]

```

async def connect_mcp365_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
of stock")
# Simulate receiving an email print("\n? Simulating email to agent mailbox...") await
handler.on_email_received( subject="Inquiry: Camping Tents for Team Retreat", body="Hi, we
need 5 tents for a company retreat. What do you recommend?" )

```