1 Overview

Neste tutorial, vamos explorar como usar o **OCI GenAI** com as bibliotecas **LLamaindex** e **Langchain** para criarmos um exemplo de um AI Agente que nos auxiliará a fazer contas. Um AI Agent nada mais é do que um sistema automatizado que pode tomar decisões e realizar tarefas de forma independente, interagindo com ferramentas (funções em python, APIs e softwares por exemplo) e o usuário para realizar tarefas específicas. No caso deste nosso exemplo, as ferramentas disponibilizadas ao agente serão funções construídas em **Python**. Você verá que ao inicializar o agente pare responder determinada pergunta, será possível acompanhar o seu fluxo de ações, onde serão mostradas as chamadas das ferramentas em cada momento específico até a obtenção do resultado final. Este tutorial é ideal para quem deseja alavancar a realização de tarefas através da interação usuário e AI ou até mesmo construir automações de tarefas repetivas, aumentando a escalabilidade da conclusão dessas tarefas e trazendo mais eficiência às empresas.

2 Criação do jupyter lab

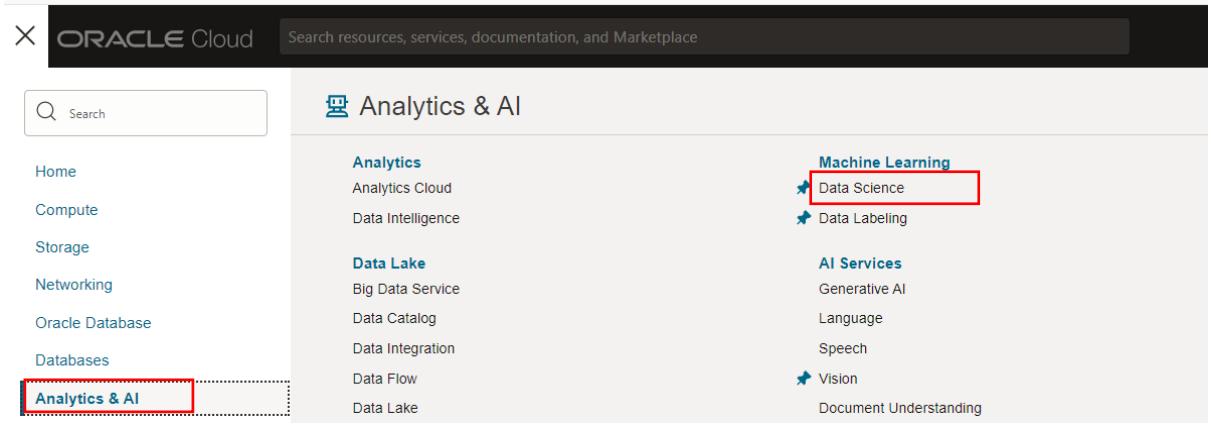
2.1 Selecionando a sua tenant

Entre no site **cloud.oracle.com** . Façam o login com as credenciais fornecidas para acessar a página inicial da Oracle Coud.

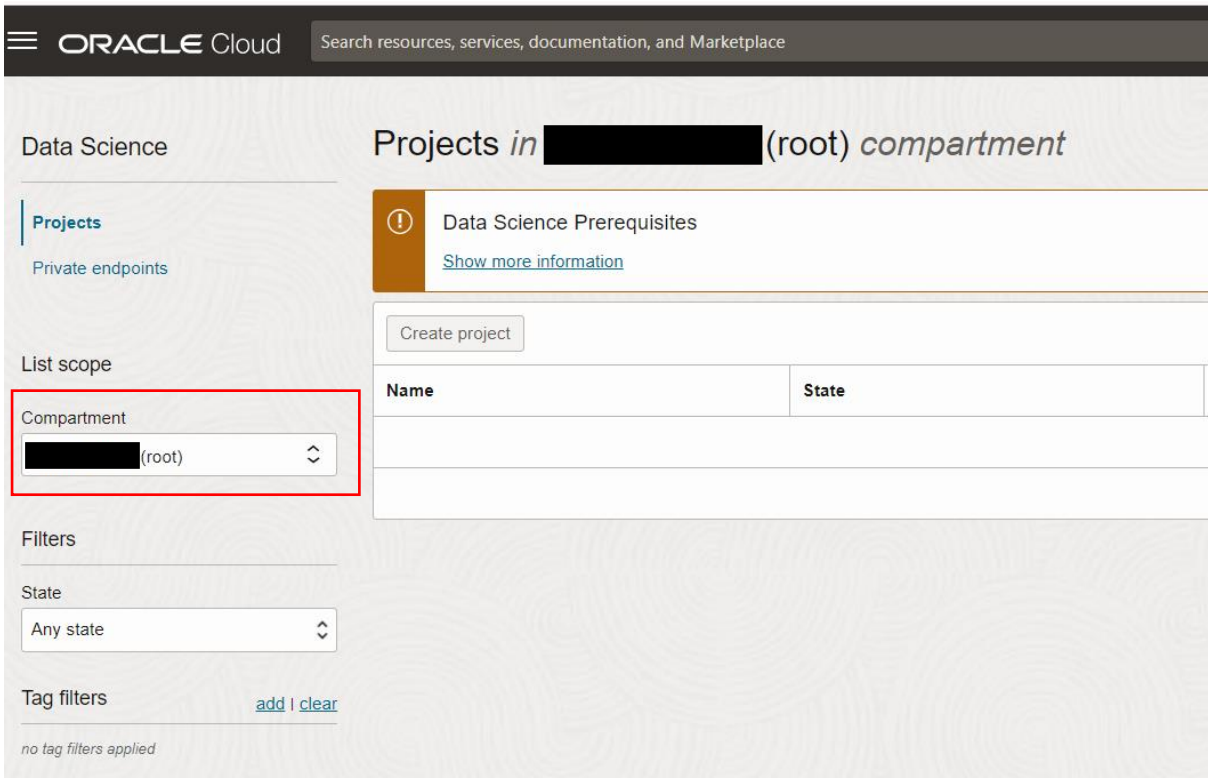
2.2 Criando seu projeto

Agora, vá ao menu (três traços horizontais) no canto superior esquerdo como é mostrado na imagem abaixo:

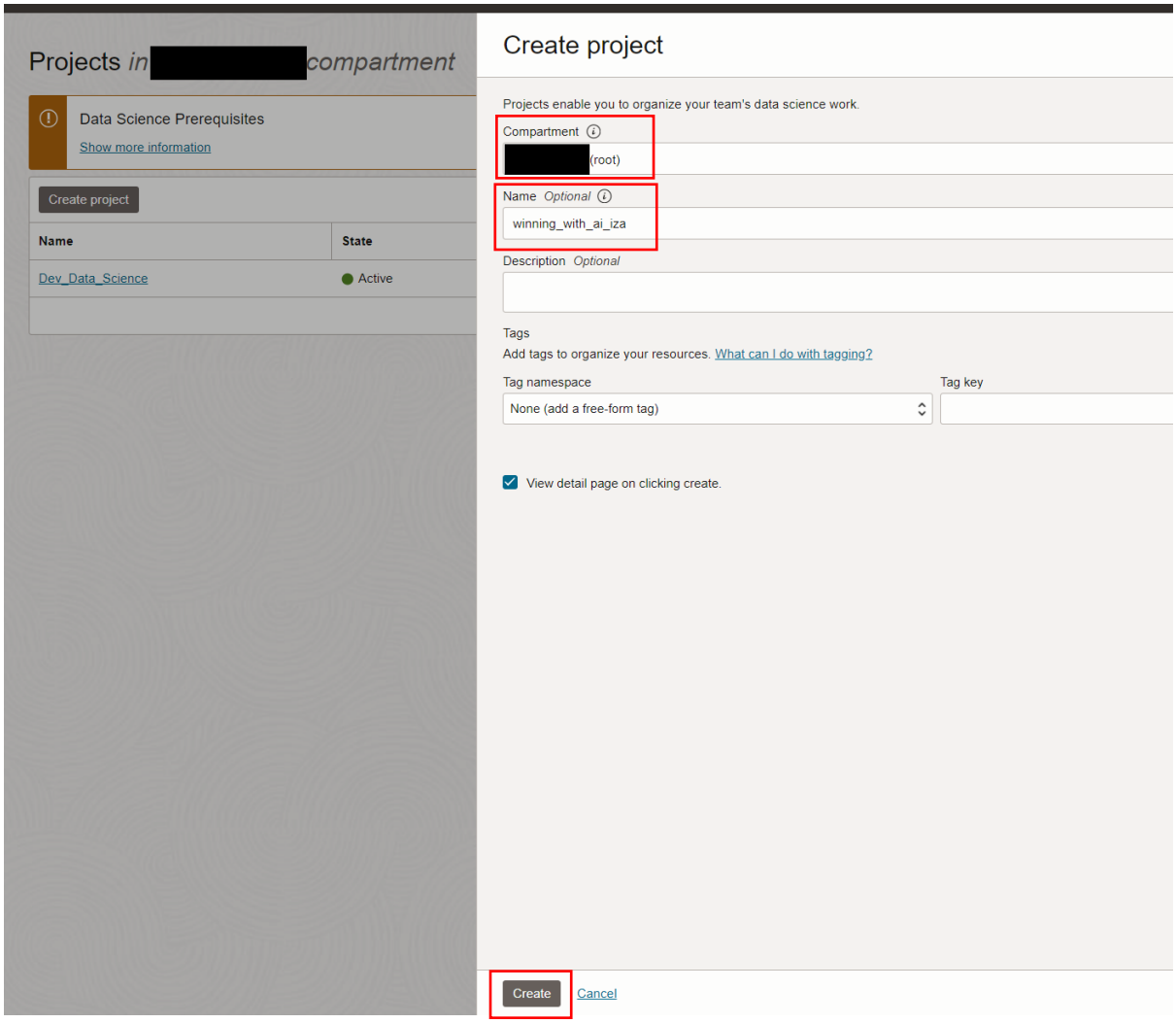
Em seguida, clique em **Analytics & AI** e depois, na seção de **Machine Learning**, clique em **Data Science**. Os campos em que você deve clicar estão em vermelho na imagem abaixo:



Agora, selecione o **Compartment *root*** que estiver aparecendo para vocês, na coluna do lado esquerdo, como demonstrado na imagem abaixo:

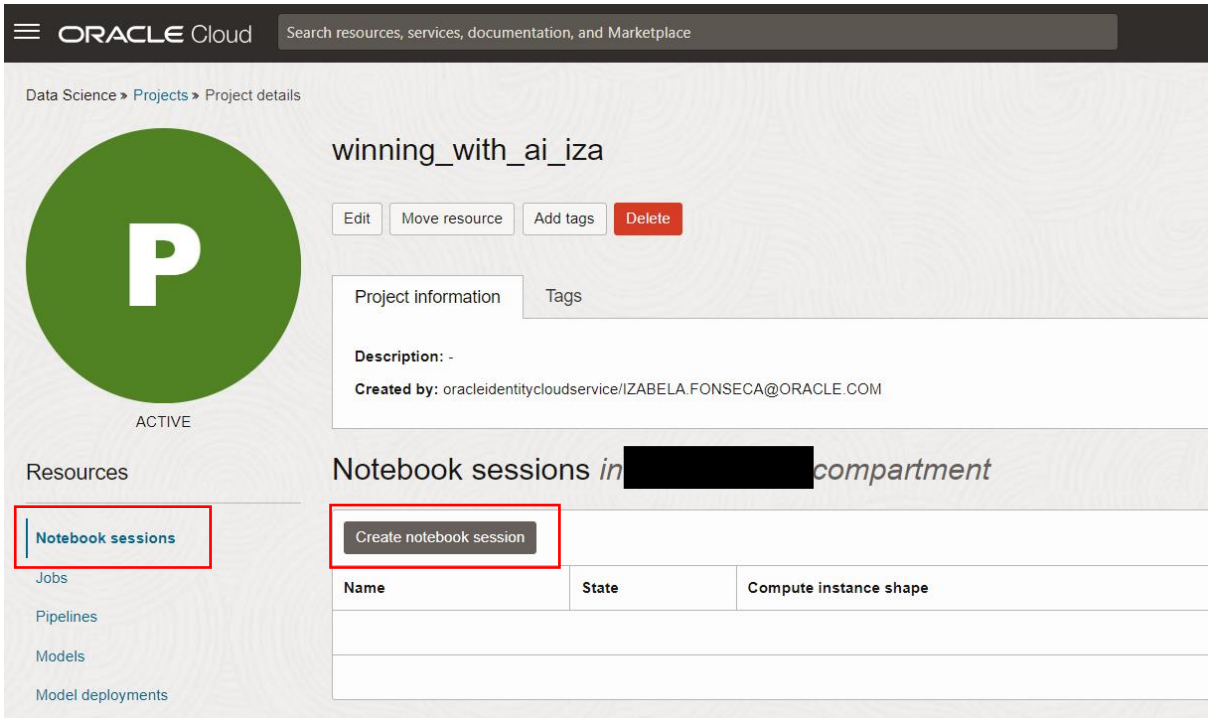


Vamos então criar o projeto, clicando no campo **Create project**, também indicado na imagem abaixo. Dessa forma, uma nova tela será aberta e nela você precisa apenas verificar se o **Compartment** é o ***root*** como você escolheu anteriormente, e então coloque o nome do seu projeto, podendo ser algo como **Dev/Test ou winning_with_ai_iza** (como no exemplo), que também pode ser visto na imagem abaixo, e por fim clique em **Create**.

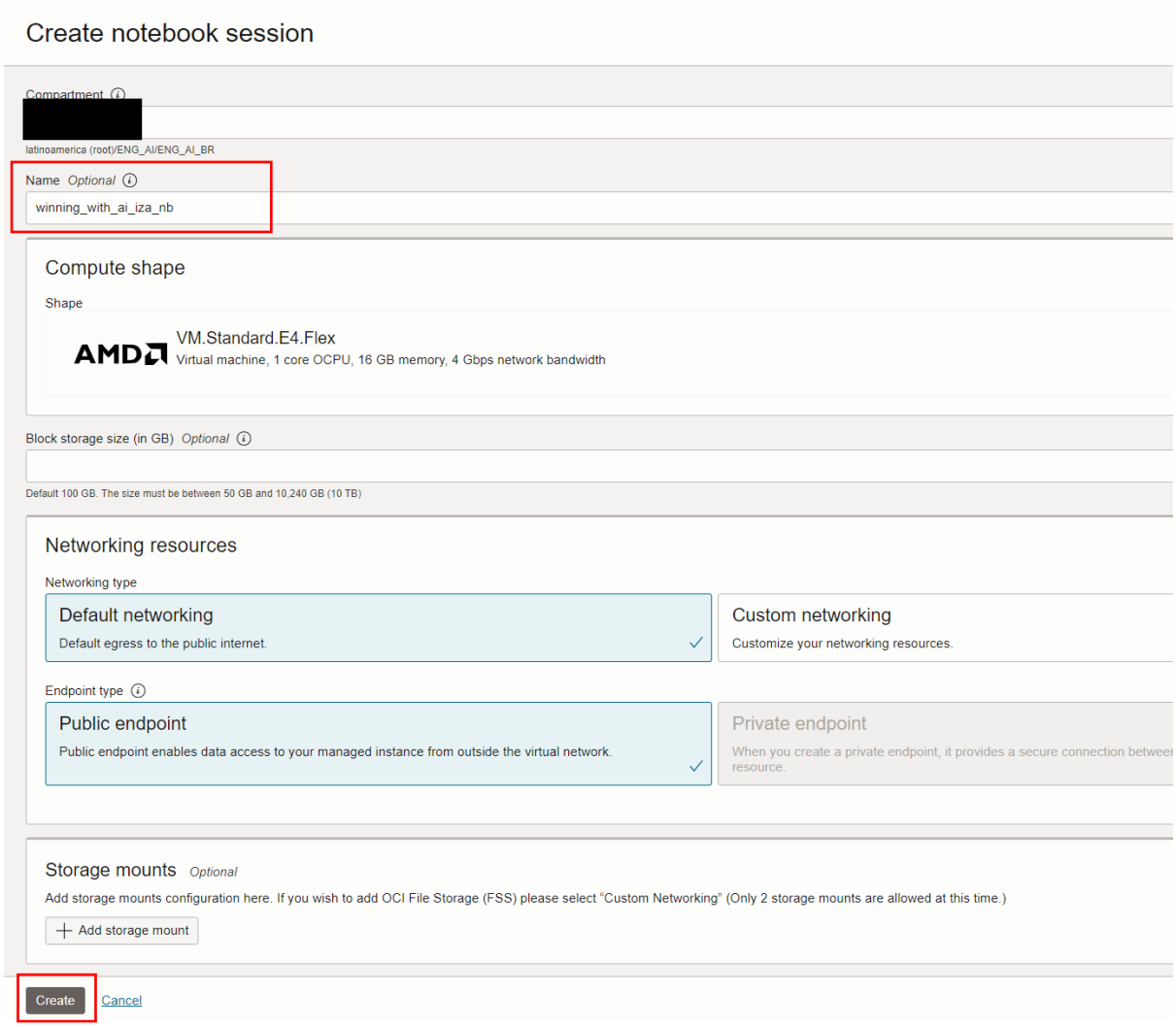


2.3 Criando seu Jupyter Lab

Após a criação do seu projeto, você será redirecionado à tela específica do seu projeto e nela, selecionando **Notebook Sessions** na coluna à esquerda você terá acesso ao botão de **Create notebook session** como mostrado abaixo:

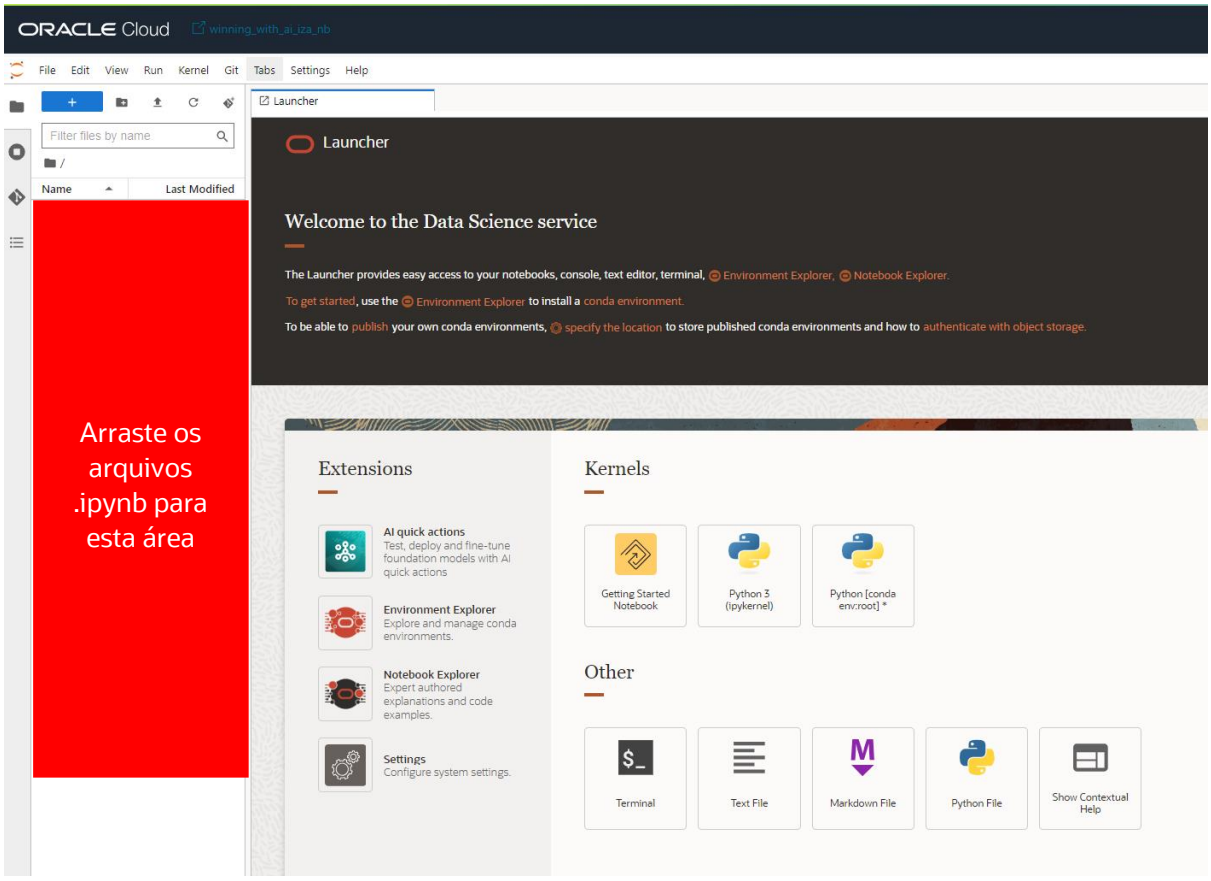


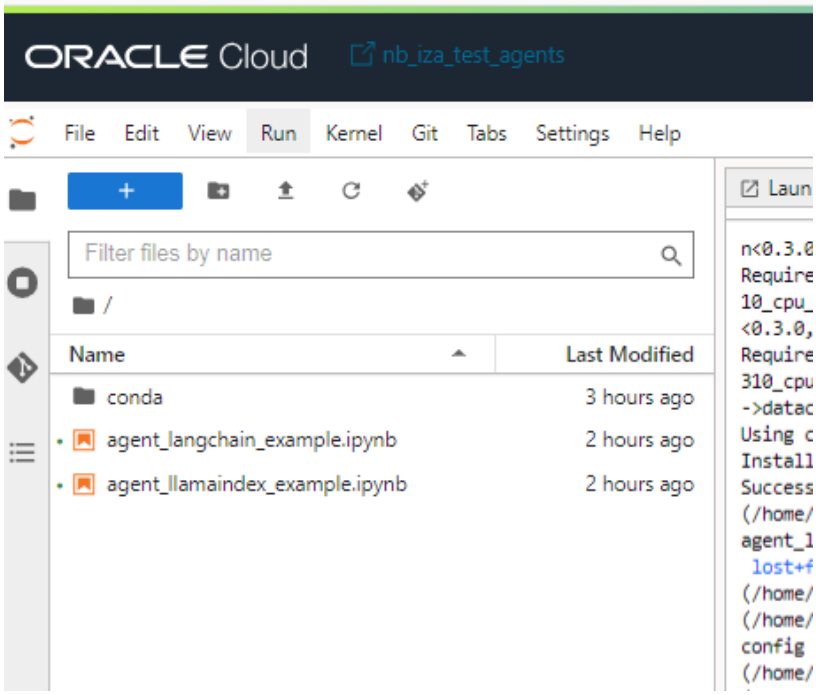
Após clicar em Create notebook session, uma nova tela irá aparecer para que você faça a configuração do seu notebook. Neste caso, vamos apenas colocar o nome do notebook e vamos deixar as demais configurações como estão mesmo. Após colocar o nome clique em **Create**. O campo onde você deve colocar o nome e o botão **Create** estão circulados em vermelho na imagem abaixo:



Após a criação é necessário esperar alguns minutos para que ela seja concluída. Quando estiver concluída, ficando então como Active. Com isso concluído, clique em Open para abrir o seu Jupyter Lab.

Agora então temos acesso ao nosso Jupyter Lab!! Vamos então arrastar os arquivos do nosso tutorial para a parte circulada em vermelho, como é mostrado na imagem abaixo:





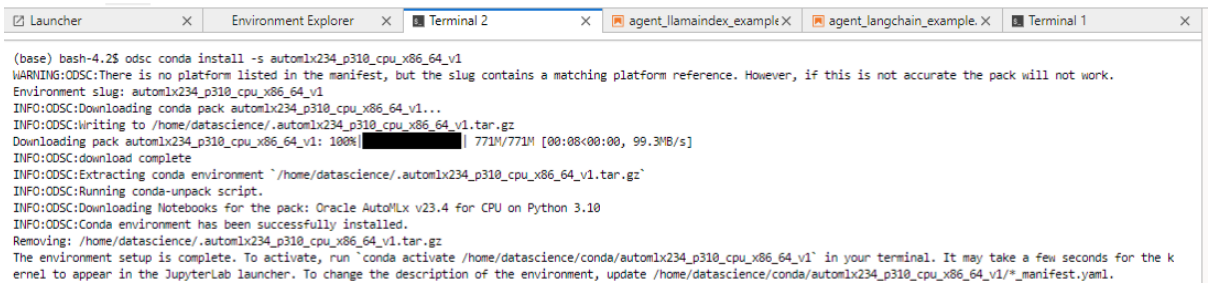
Depois disso, siga os próximos passos para criar o environment e para depois rodar o seus arquivos como orientado!

3 Criação do environment

Após ter o seu ambiente provisionado no OCI Data Science, você irá seguir os seguintes passos para configurar o seu environment (nele estarão instaladas todas as bibliotecas que você precisará para rodar o código que iremos usar):

- Crie seu environment:
  - Vá na aba Launcher e clique em **Environment Explorer**
  - Nele, encontre a opção como na imagem abaixo, depois clique nos **três pontinhos** do lado direito e depois em **Instalar**, como na imagem abaixo:

  - Logo em seguida, um **terminal** irá abrir para que a instalação do environment seja feita. Esse processo leva menos de 2 minutos e ao terminar irá aparecer a mensagem “The environment setup is complete”, como é possível ver na imagem abaixo.

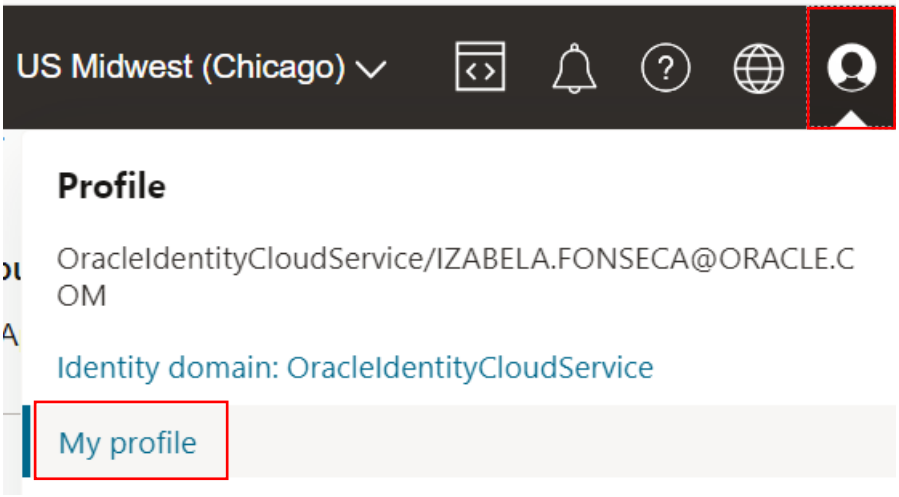


Após finalizar a criação do seu environment, você irá seguir o passo a passo a abaixo para concluir a configuração do environment:

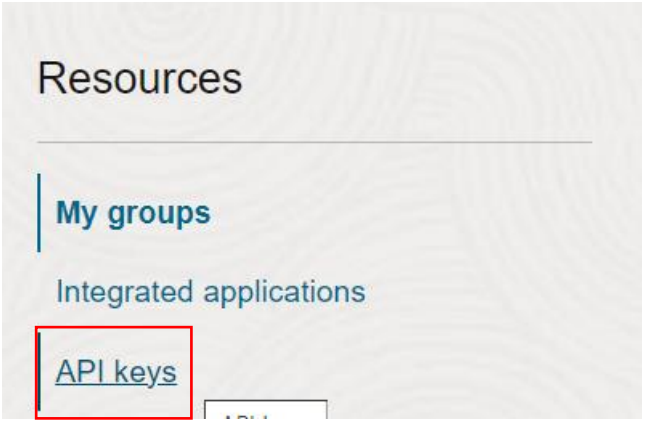
- Ative o environment com o seguinte commando no terminal (copie e cole todo o conteúdo após os dois pontos no terminal): **conda activate /home/datascience/conda/automlx234_p310_cpu_x86_64_v1**
- Depois disso, você irá instalar as seguintes biblioteca abaixo uma a uma:
  - pip install llama-index
  - pip install -U oci
  - pip install llama-index-embeddings-oci-genai
  - pip install llama-index-llms-oci-genai
  - pip install langchain
  - pip install langchain-community

4 Criação do seu arquivo config

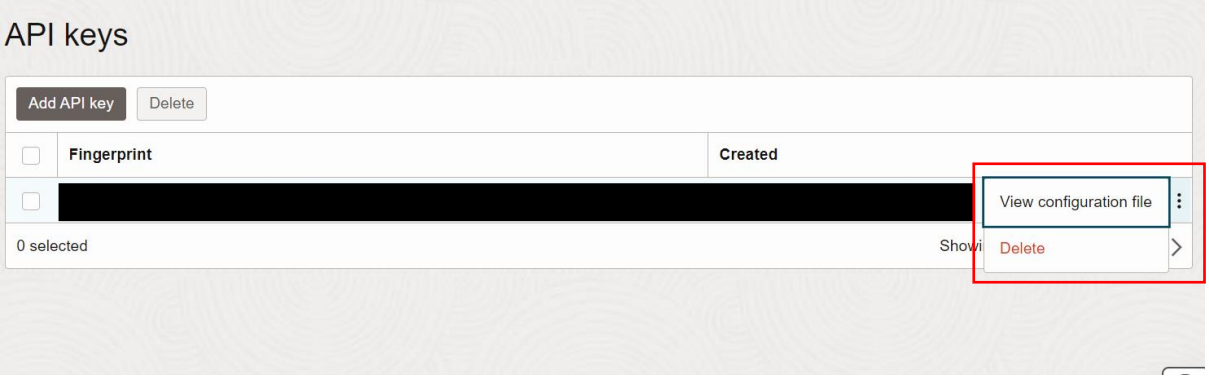
No site **cloud.oracle.com**, após fazer o seu login, clique no ícone do seu usuário no canto superior direito como mostrado na imagem abaixo e então clique em **My profile**.



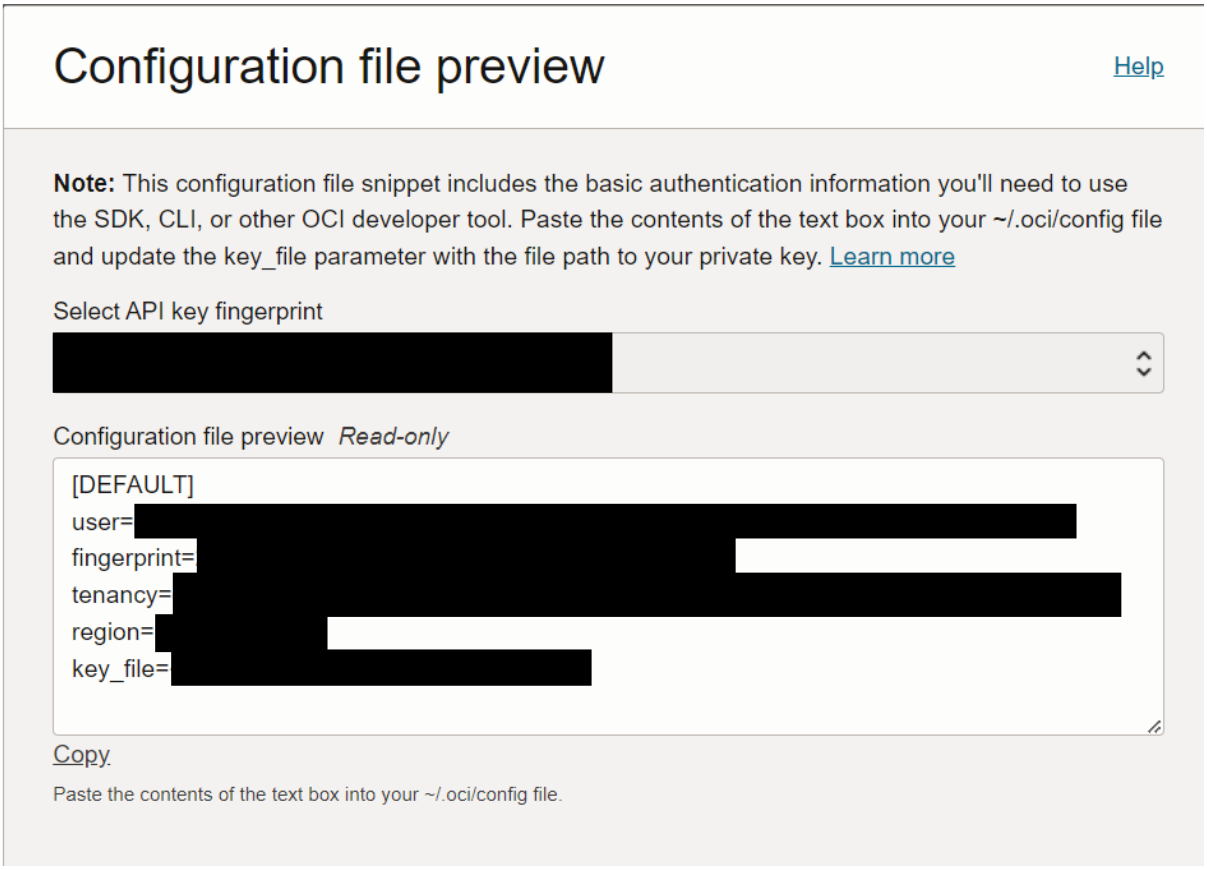
Depois disso, na coluna do lado esquerdo da página na qual você se encontra agora, clique em **API Keys**.



Agora, caso você não tenha, crie uma API Key clicando no botão Add API key. Caso você já tenha uma API key, clique nos três pontinhos no canto direito e clique em **View configuration file**.

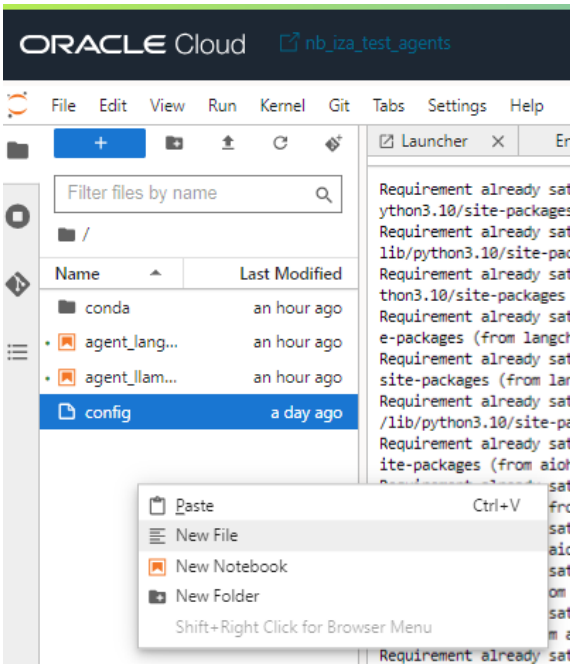


Agora copie o conteúda da janela que abriu, como a janela abaixo.

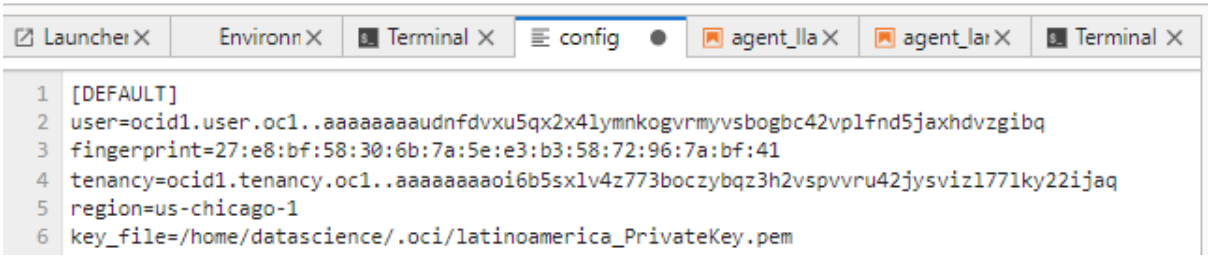


Caso você não tenha uma API KEY criada, basta clicar em "Add API Key", baixar o arquivo da **chave privada (private key)** para a sua máquina local, que será um arquivo com extensão **.pem**. É uma boa prática alterar o nome do arquivo para um nome fácil de ser lembrado.

Volte para o seu Jupyter Lab e, na área para a qual você arrastou os arquivos .ipynb, clique com o botão direito e clique em **New file** e coloque o nome dele como **config**, sem nenhuma extensão. Arraste para esta região também o arquivo da sua **private_key.pem**.



Dentro deste arquivo, coloque o conteúdo obtido a partir da sua API Key neste arquivo, alterando o **key_file** com o caminho completo da sua **private_key.pem** (/home/datascience/.oci/latinoamerica_PrivateKey.pem por exemplo) como é mostrado na imagem abaixo (em seguida vamos mostrar como passar o arquivo config e o .pem para a pasta .oci como é necessário para que o sistema reconheça suas credenciais).



Depois, vá no terminal e dê um comando **ls** para verificar se na pasta onde você está agora contém os arquivos **config** e **private_key.pem**. Estando na pasta correta, dê os seguintes comandos:

- **mkdir ~/.oci**
  - Este comando vai criar a pasta oculta **.oci**.
- **mv config private_key.pem .oci**
  - Este commando irá mover os arquivos **config** e **private_key.pem** para o diretório oculto **.oci**.
- **cd .oci**
  - Este comando irá levar você para a pasta **.oci** através do terminal.
- **pwd**
  - Este comando ira mostrar em qual pasta você está, e serve apenas pra você confrimar que está na pasta **.oci**.
- **ls**
  - Este comando irá listar os arquivos dentro da pasta na qual você se encontra. Neste caso devem ser listados os arquivos **config** e **private_key.pem**.
- **cd ..**

- Por fim, este comando apenas volta para a pasta anterior.

5 Explicação do passo a passo de cada script

5.1 AI Agent usando Llamaindex

5.1.1 Explicação resumida do código

O AI Agent que iremos utilizar criar irá utilizar três ferramentas construídas em Python: uma para somar, uma para multiplicar e uma para elevar um valor ao quadrado. Iremos solicitar que ele faça um cálculo e observaremos o agente chamando cada uma dessas ferramentas para fazer os cálculos. Lembrando que para rodar cada célula dos notebooks, deixa a célula selecionada e clique em *shift + enter* no seu teclado. O primeiro notebook que iremos utilizar é o de nome agente_llamaindex_example.ipynb. O passo a passo básigo que iremos seguir nele ficou como segue:

- Vamos importar bibliotecas necessárias;
- Depois iremos criar a variável **compartment_id** que irá conter as nossas credenciais;
- Criamos então as funções em python referente às operações que queremos habilitar o AI Agent a executar: **add**, **multiply** e **square**;
- Depois disso vamos criar uma variável **command_r_plus** para armazenar a chave de acesso do modelo que queremos usar do OCI GenAI;
- Criamos então um objeto parametrizado do nosso modelo a partir da OCI GenAI;
- Depois disso criamos o objeto do nosso agente a partir do framework ReAct, trazendo para o agente tanto o objeto do modelo quanto a lista das funções criadas;
- Criamos então um prompt simples, passando orientações e também a conta que queremos que a gente faça;
- Por fim, executamos o agente passando o prompt e deopis observamos sua chamada das ferramentas fornecidas para então checarmos o resultado.

5.1.2 Explicação detalhada do código

Vamos então entender mais detalhadamente como o código que utiliza o *framework* do Llamaindex funciona:

- Importando as bibliotecas
  - **import os:** Importa funções para interagir com o sistema operacional;
  - **from llama_index.llms.oci_genai import OCIGenAI**: Importa a classe OCIGenAI para usar IA generativa da OCI;
  - **from llama_index.core.agent import ReActAgent**: Importa ReActAgent, um agente que utiliza prompts intermediários para decisões;
  - **from llama_index.core.tools import FunctionTool**: Importa FunctionTool, uma ferramenta para definir funções que o agente pode usar;
  - **from llama_index.core.llms import ChatMessage**: Importa ChatMessage, que representa mensagens de chat em um modelo de linguagem.
- Carregando suas credenciais
  - **compartment_id = os.environ["NB_SESSION_COMPARTMENT_OCID"]**: Obtém o valor da variável de ambiente NB_SESSION_COMPARTMENT_OCID e o atribui à variável compartment_id.
- Criamos as ferramentas que serão usadas pelo AI Agent (aqui vamos explicar apenas uma ferramenta porque as demais funcionam da mesma forma)

  - def add(inputs: str) -> int

        - Define uma função chamada add que recebe uma string como entrada e retorna um número inteiro;
    - """Add two numbers given as a comma-separated string."""
        - Docstring que descreve a função, explicando que ela soma dois números separados por vírgula;
    - a, b = map(int, inputs.split(","))
        - Divide a string inputs em dois números, converte-os para inteiros e os atribui a a e b;
    - return a + b
        Retorna a soma dos dois números a e b.

- Criando o objeto do modelo parametrizado
    - **command_r_plus = "ocid1.generativeaimodel.XXXX"**: Define o identificador do modelo OCI GenAI a ser usado;
    - **llm = OCIGenAI**: Inicia uma instância do modelo de linguagem OCIGenAI;
        - **model=command_r_plus**:Especifica o modelo a ser usado, referenciado pela variável command_r_plus;
        - **service_endpoint="https://inference.generativeai.us-chicago-1.oci.oraclecloud.com"**: Define o endpoint do serviço OCI Generative AI para realizar inferências;
        - **compartment_id=compartment_id**:Define o compartmento da OCI onde o modelo está localizado, usando o ID do compartimento;
        - **provider="cohere"**: Especifica o provedor do modelo, que é cohere neste caso;
        - **context_size="128000"**: Define o tamanho do contexto de tokens que o modelo pode processar;
        - **temperature=0**: Ajusta o parâmetro de temperatura para controlar a criatividade/aleatoriedade das respostas.
- Criando um objeto do AI Agent
    - **agent = ReActAgent.from_tools(functions, llm=llm, verbose=True)**: Cria um agente ReAct que utiliza as ferramentas definidas (functions) e o modelo de linguagem (llm), com a saída detalhada (verbose=True), que mostrará o "racicínio" do modelo durante a execução e uso das ferramentas;
- Executando o AI Agent
    - **prompt**: Define qual a ação o AI Agent deverá executar bem como define algumas orientações;
    - **result = agent.chat(prompt)**: Executa o agente a partir do prompt definido.

6 Como este resultado pode ser utilizado?

Esse agente simples que realiza cálculos pode servir como base para a compreensão de como agentes de IA podem ser utilizados em contextos mais avançados e complexos. Por exemplo, em vez de apenas executar operações matemáticas, o agente poderia ser expandido para realizar tarefas mais sofisticadas, como processamento de dados financeiros, onde ele pode interpretar grandes volumes de informações e tomar decisões automatizadas, como calcular riscos de investimentos ou otimizar alocações de portfólio. Ele também pode ser integrado a sistemas de análise de dados em tempo real, onde o agente interage com múltiplas fontes de dados, executa cálculos complexos e gera insights acionáveis para os usuários.

Além disso, em um cenário de assistentes virtuais especializados, esse conceito pode ser aplicado para automatizar processos em áreas como engenharia ou ciências, onde o agente poderia não apenas resolver equações, mas também interpretar simulações ou resultados de experimentos. Ele poderia atuar como um colaborador, auxiliando em cálculos complexos e gerando relatórios explicativos. Essa abordagem também pode ser estendida para integração com sistemas de IA multi-agentes, onde vários agentes, cada um com funções específicas, cooperam para resolver problemas de grande escala, como otimização de rotas logísticas ou controle automatizado de redes de energia.