Big Anchoring Machine Learning Module Documentation

Versão alpha_01

ICA

03 set., 2024

Contents:

CAPÍTULO 1

BigAnchoringMachieLearningModule

1.1 O projeto Big Anchoring

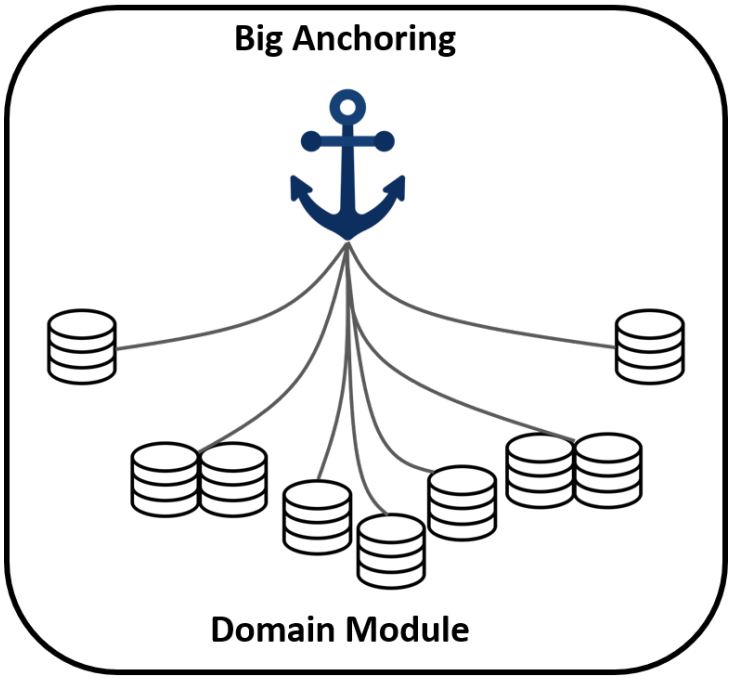
O Big Anchoring é um projeto inovador, projetado para revolucionar o processo de ancoragem de FPSOs (Floating Production Storage and Offloading) em múltiplos aspectos essenciais. O projeto se destaca por sua contribuição no gerenciamento e tratamento eficiente dos dados obtidos através do sensoriamento das plataformas, garantindo que as informações críticas sejam processadas com precisão e rapidez.

Além disso, o Big Anchoring permite a flexibilidade de utilizar diferentes infraestruturas, sejam elas em nuvem (cloud) ou locais (on-premises), para otimizar o processamento de tarefas, oferecendo soluções adaptáveis às necessidades específicas de cada operação. Outro ponto forte do projeto é sua interface gráfica, que combina robustez com uma usabilidade intuitiva. Essa interface simplifica a visualização e o monitoramento dos dados disponíveis, proporcionando uma experiência de usuário mais fluida, clara e eficiente.

Para atuar de forma eficiente nos diferentes aspectos do processo de ancoragem de FPSOs, o projeto Big Anchoring é dividido em três módulos principais: **Gerenciamento de Dados**, **Processamento** e **Visualização**. Esses módulos trabalham em conjunto para garantir o tratamento adequado dos dados, a otimização do processamento em diferentes infraestruturas (cloud e on-premises), e a visualização intuitiva das informações disponíveis.

1.1.1 Big Anchoring Domain Module

Este módulo apresenta uma plataforma avançada de Big Data e sistemas distribuídos, projetada para o tratamento eficiente de grandes volumes de dados em ambientes críticos, como a ancoragem de FPSOs, utilizando datalakes e o ecossistema Kafka.

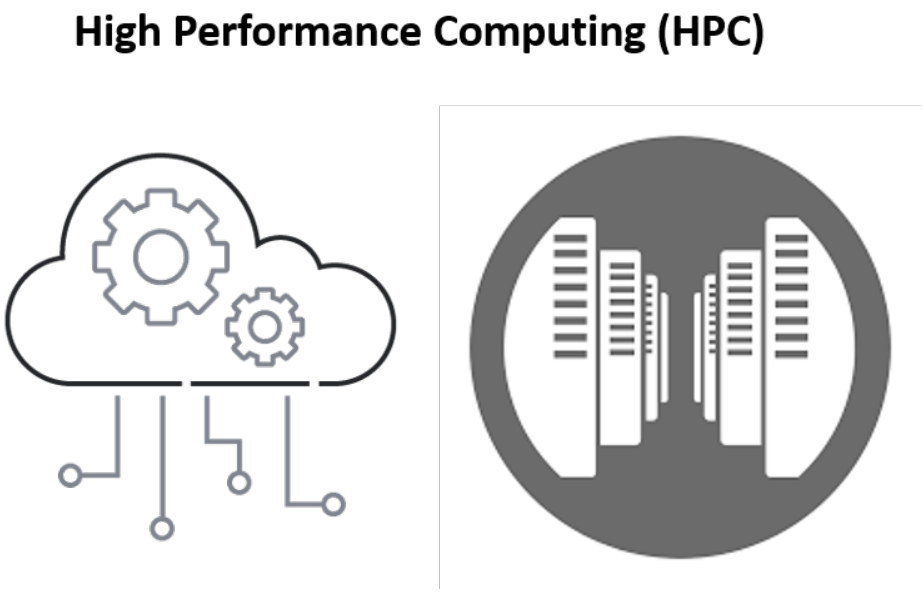


Eficiência no Gerenciamento de Dados Críticos

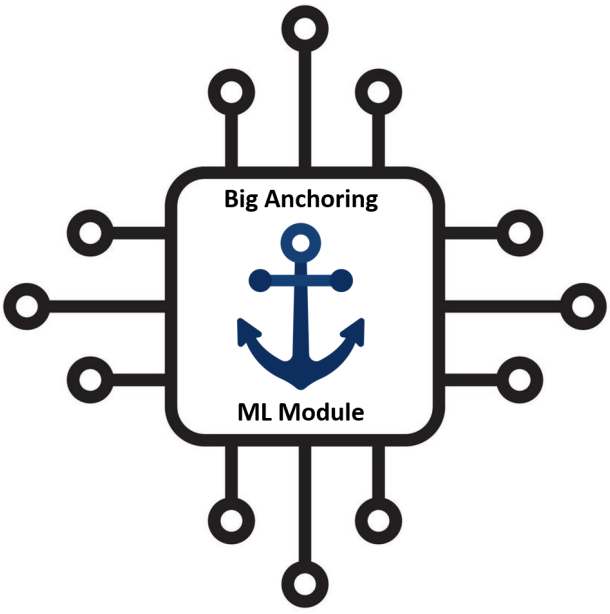
O Big Anchoring Domain Module é uma plataforma inovadora baseada em Big Data, criada para gerenciar de forma eficiente grandes volumes de dados em sistemas críticos, como a ancoragem de FPSOs. Com uma arquitetura escalável, a solução organiza datalakes para armazenamento e manipulação de dados estruturados e não estruturados, essencial para o controle de informações vitais em operações offshore de óleo e gás.

1.1.2 Big Anchoring Machine learning Module

Com o aumento da complexidade dos modelos de inteligência artificial para o monitoramento de FPSOs e o consequente crescimento do volume de dados processados, a necessidade de um aumento significativo na capacidade computacional torna-se evidente. Esse incremento é crucial para garantir a execução eficiente das tarefas, o que torna indispensável o uso de infraestruturas de High-Performance Computing (HPC). Essas infraestruturas podem ser implementadas como clusters on-premises ou por meio de recursos em nuvem (cloud), oferecendo a flexibilidade e escalabilidade necessárias para atender às demandas do projeto.



O módulo **Big Anchoring Machine Learning** oferece um serviço que facilita a utilização de diferentes infraestruturas, sejam elas clusters on-premises ou estruturas em nuvem. Com ele, os usuários podem submeter tarefas, quando autorizados, tanto para a estrutura HPC da Petrobras (atena02) quanto para as infraestruturas da Amazon (AWS) e da Microsoft (Azure), proporcionando escalabilidade e acesso a grande poder computacional.

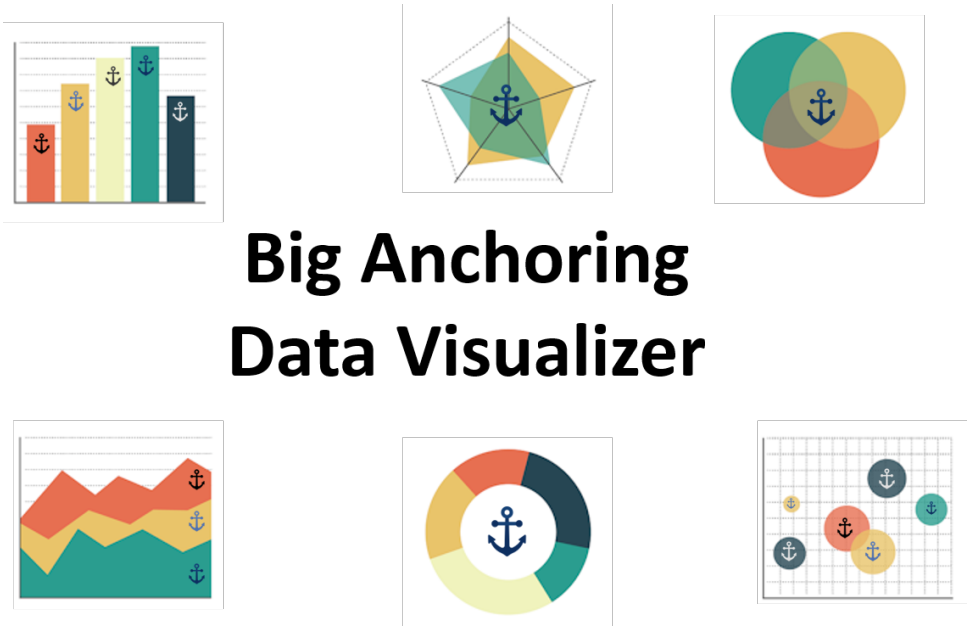


A forma escolhida para permitir que o usuário acesse o serviço do **Big Anchoring Machine Learning Module** é por meio de uma biblioteca Python. Essa biblioteca facilita a utilização dos serviços, permitindo que sejam

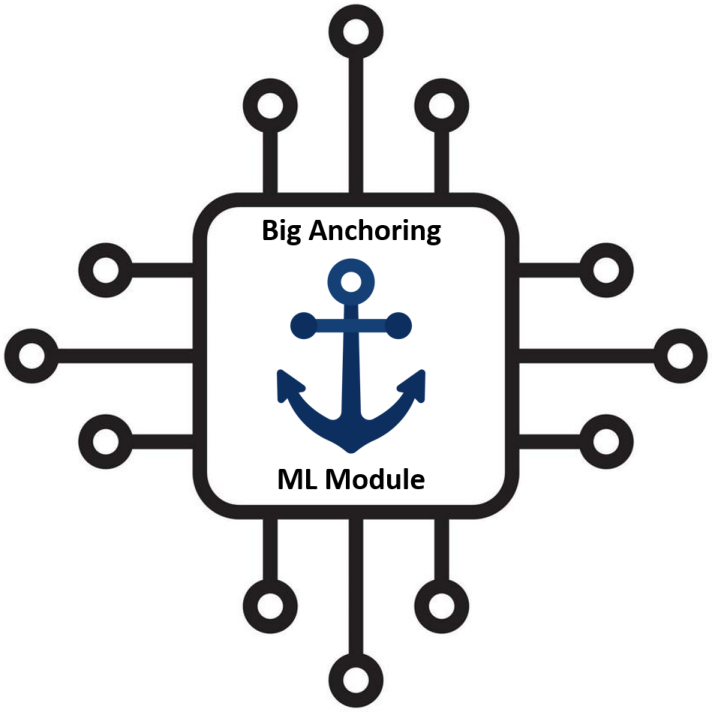
acessados de maneira semelhante à chamada de funções em Python, sem a necessidade de se preocupar com requisições diretas ao serviço.

1.1.3 Big Anchoring Data Visualizer Module

Com o intuito de fornecer uma ferramenta que facilite a visualização dos diferentes monitoramentos necessários ao processo de ancoragem de uma FPSO, foi desenvolvido o **Big Anchoring Data Visualizer**, que oferece uma interface gráfica amigável e intuitiva.



1.2 Big Anchoring ML Module - Documentação



Esta biblioteca Python foi desenvolvida para facilitar a submissão de tarefas em diferentes infraestruturas, incluindo a da Petrobras (atena02) e infraestruturas de nuvem, como Amazon Web Services (AWS) e Microsoft Azure. Com funções parametrizadas, a biblioteca elimina a necessidade de criar scripts separados para fazer requisições ao servidor, simplificando o processo de execução de tarefas no Big Anchoring ML Module.

1.2.1 biganchoring_ml_module.task_management

class biganchoring_ml_module.task_management.**TaskHandler**(*task_id*)

Bases: object

__init__(*task_id*)

cancel_task()

Cancela a tarefa com o task_id fornecido.

get_task_run_id()

Resumo:

Recupera o ID de execução de uma tarefa (task) da API usando o task ID. A função conecta-se ao servidor utilizando credenciais de autenticação armazenadas em variáveis de ambiente, envia uma solicitação GET para recuperar o ID de execução a partir do MLFLOW e retorna a resposta da API.

Exceções:

ValueError: Se a resposta da API indicar uma falha na recuperação do ID de execução da tarefa, como um código de status inesperado.

Retorna:
requests.models.Response: O objeto de resposta HTTP contendo as informações do ID de execução da tarefa.

`get_task_status()`

Resumo:
Recupera o status de um task da API usando o ID da task. A função conecta-se ao servidor utilizando credenciais de autenticação armazenadas em variáveis de ambiente, envia uma solicitação GET para recuperar o status da task e retorna a resposta da API.

Exceções:
ValueError: Se a resposta da API indicar uma falha na recuperação do status da task, como um código de status inesperado.

Retorna:
requests.models.Response: O objeto de resposta HTTP contendo as informações do status da task.

1.2.2 biganchoring_ml_module.submitter

`class biganchoring_ml_module.submitter.Server_connection(node_name='atn1b01n28', username=None, password=None, port=None)`

Bases: `object`

Server_connection foi desenvolvido para facilitar os processos de conexões.

`__init__(node_name='atn1b01n28', username=None, password=None, port=None)`

Resumo:
Inicializa a classe Server_connection com os argumentos necessários.

Parâmetros:
node_name (str, obrigatório): O nó onde o servidor está rodando em atena02.

username (str, obrigatório): O nome de usuário necessário para fazer login no servidor.

password (str, obrigatório): A senha necessária para fazer login no servidor.

port (str, obrigatório): A porta necessária para fazer login no servidor.

`login()`

Resumo:
Realiza a solicitação *POST* para efetuar o login.

Parâmetros:
Nenhum

Retorna:
Resposta da solicitação de login.

`class biganchoring_ml_module.submitter.Submitter(dataset_name=None, dataset_file=None, processor_file=None, runner_location='atena02', execution_mode='mlflow', experiment_name='default_experiment', execution_command='mlflow run project', tracking_uri=None, instance_type='gpu', account='default_account', **model_params)`

Bases: `object`

__init__(*dataset_name=None*, *dataset_file=None*, *processor_file=None*, *runner_location='atena02'*, *execution_mode='mlflow'*, *experiment_name='default_experiment'*, *execution_command='mlflow run project'*, *tracking_uri=None*, *instance_type='gpu'*, *account='default_account'*, ***model_params*)

Resumo:
Inicializa a classe Submiter com vários parâmetros e parâmetros opcionais de modelo.

Parâmetros:
dataset_name (str, obrigatório): O nome do dataset. Padrão é None.

dataset_file (str, obrigatório): O nome do arquivo do dataset. Padrão é None.

processor_file (str, obrigatório): O nome do arquivo do processador. Padrão é None.

runner_location (str, obrigatório): A localização do runner. Padrão é «atena02».

execution_mode (str, obrigatório): O modo de execução («mlflow» ou «no-mlflow»). Padrão é «mlflow».

experiment_name (str, obrigatório): O nome do experimento. Padrão é «default_experiment».

execution_command (str, obrigatório): O comando que o usuário deseja executar. Padrão é «mlflow run project».

tracking_uri (str, obrigatório): O URI onde vai ocorrer o tracking do MLFlow. Padrão é None.

instance_type (str, obrigatório): O tipo de instância a ser utilizada. Padrão é «gpu».

account (str, obrigatório): A conta a ser utilizada. Padrão é «default_account».

**model_params: Parâmetros adicionais do modelo como argumentos de palavra-chave. Estes serão armazenados como model_params.

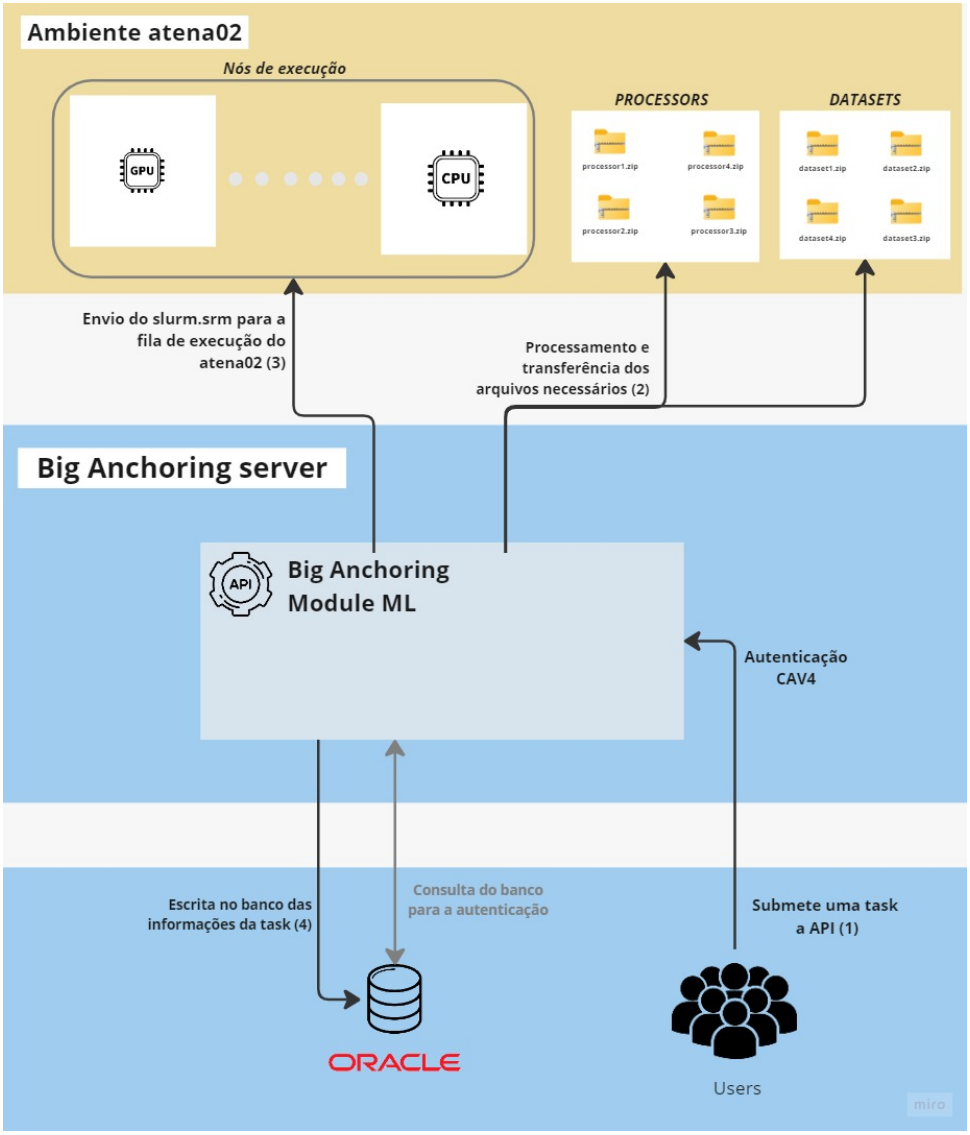
submit_task()

Resumo:
Realiza os passos necessários para fazer a submissão de uma tarefa.

Retorna:
Resposta da solicitação de submissão.

1.3 Exemplos de Uso

Estes exemplos ilustram as principais funcionalidades dessa biblioteca em desenvolvimento. O objetivo dessa biblioteca é simplificar o acesso aos serviços de submissão de tarefas em diferentes infraestruturas, oferecidas pela API do Big Anchoring Machine Learning Module. A figura abaixo ilustra de maneira simplificada como o serviço está atuando em relação à infraestrutura do atena02.



Nos exemplos a seguir, vamos explorar os principais procedimentos para a gestão de tarefas dentro do cluster. Primeiramente, veremos como realizar a submissão de uma tarefa, seguido pelo processo de obtenção do status da task que foi previamente submetida. Em seguida, aprenderemos a recuperar o *run_id* do MLflow utilizado na task, se aplicável. Por fim, demonstraremos como cancelar uma tarefa que já foi submetida.

1.3.1 Submissão de uma task ao atena02

```
from biganchoring_ml_module import Submitter
import time
import json
# Create a Submitter object
sb = Submitter( dataset_name="Dataset_name",
                dataset_file="test_data.zip",
                processor_file="test_processor.zip",
                runner_location="atena02", execution_mode="mlflow", experiment_name=
↪"test_lib",
                execution_command="mlflow run measurements_regression_training_right",
                instance_type="gpu", account="twinscie", n_estimators=2, random_
↪state=42)

submission_response = sb.submit_task()
response_data = json.loads(submission_response.text)

if response_data:
   job_id = response_data.get("job_id")
   task_id = response_data.get("id")
   experiment_name = response_data.get("experiment_name")
   instance_type = response_data.get("instance_type")

   print(f"Job ID:{job_id}")
   print(f"ID: {task_id}")
   print(f"experiment_name: {experiment_name}")
   print(f"instance_type: {instance_type}")
```

1.3.2 Obtendo Satus de uma task

```
from biganchoring_ml_module import TaskHandler
import json

task_id = input('Entre com o seu task_id:  ')

task = TaskHandler(task_id)
status_response =  task.get_job_status()

status_data = json.loads(status_response.text)
print(status_data)
```

1.3.3 Obtendo o MLFlow run_id de uma task

```
from biganchoring_ml_module import TaskHandler
import json

task_id = input('Entre com o seu task_id:  ')

task = TaskHandler(task_id)
status_response =  task.get_job_run_id()

status_data = json.loads(status_response.text)
print(status_data)
```

1.3.4 Cancelando uma task

```
from biganchoring_ml_module import TaskHandler
import json

task_id = input('Entre com o seu task_id:  ')

task = TaskHandler(task_id)
status_response =  task.cancel_task()

status_data = json.loads(status_response.text)
print(status_data)
```

1.4 Links Uteis

ICA Home Page

API GitHub Project

GitHub Project

Índice de Módulos do Python

b