JPMC PROJECT TECH NOTES

**JPMC Project Microservices List**

mss-service-core-engine-dataprovider

mss-service-coreengine

mss-service-whatif

pra-batch-service – for autosys job deployment – python script

mss-service-tradeui – {

hibiscusui – main front face pmt project

Trade

Model

Hypo

PTC Approvers

Pending Trade

Whatif

Hypo Scenario

Portfolio Monitoring

Model History

Hypo History

Vehicle Holding

Trade Flow

PTC Rules

}

mss-service-superpmt

mss-service-newgridapp

mss-service-pmtptc

pra-pmt-gateway

pra-outbound-gateway-service

wm-hibiscus-validation-service

pra-pmt-liquibase-sql

mss-pmt-streamlit - UI hibiscus dashboard UI to different types of status{

stale price-

Audit details - MSTAR -BBG file transfer

Las 5 days approve trade details

PTC check details – fail, pass attributes, isin voleren,figiid, negative price,

CIO

SST,

MFAP

BBG

MSTAR

}

pra-pmt-lambda-app – holding file -BBG , MSTAR

pmt-aws-terraform – deploy lamda s3 infra

wm-s3-browser – s3 files

mss-service-toolboxapp - bautools

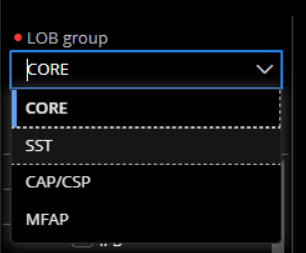
pra-pmt-holding{

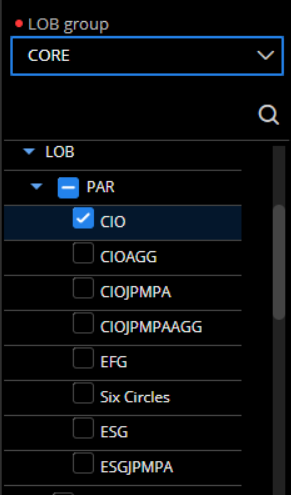
getAccountPositions

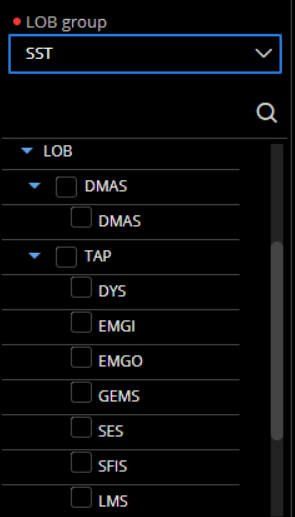
vehicleDetails

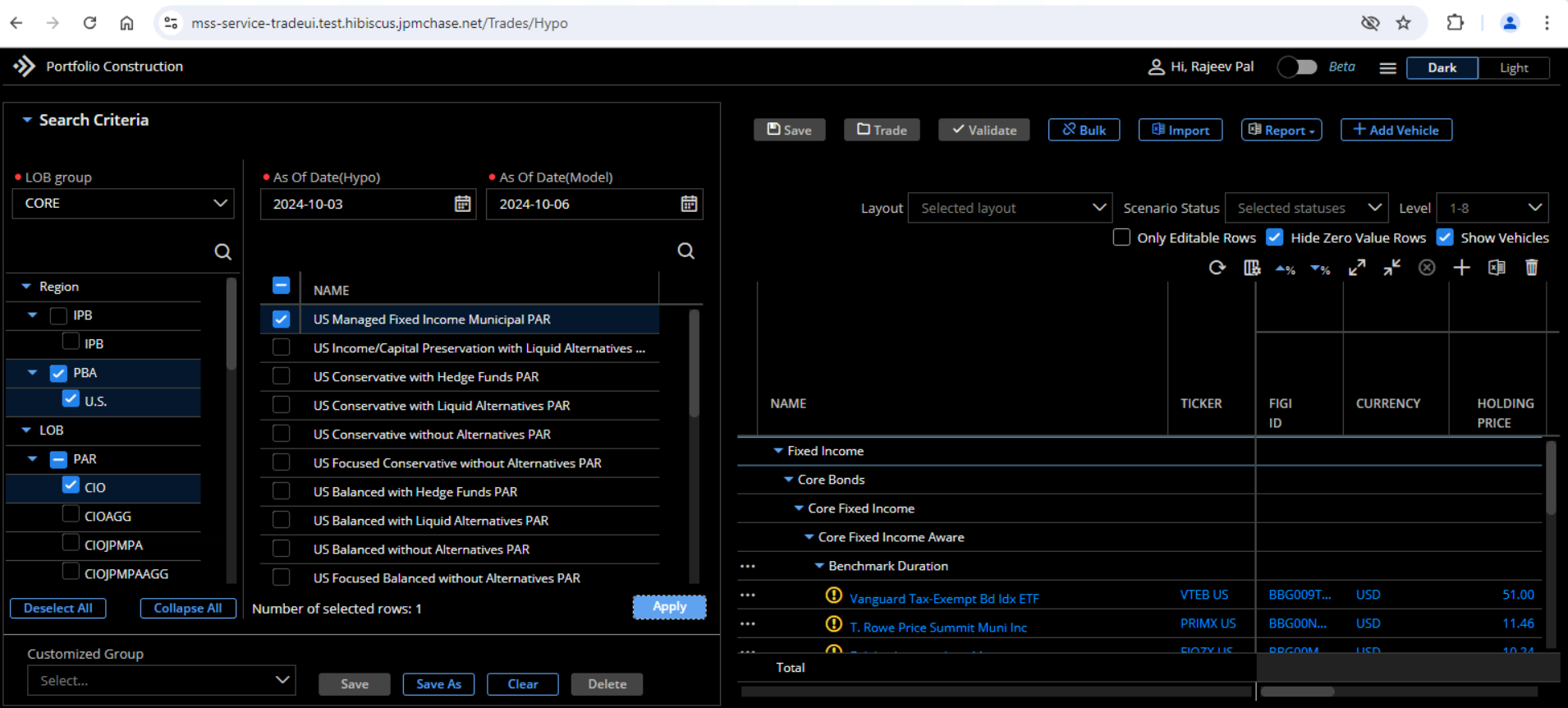
asset class classificationsTree

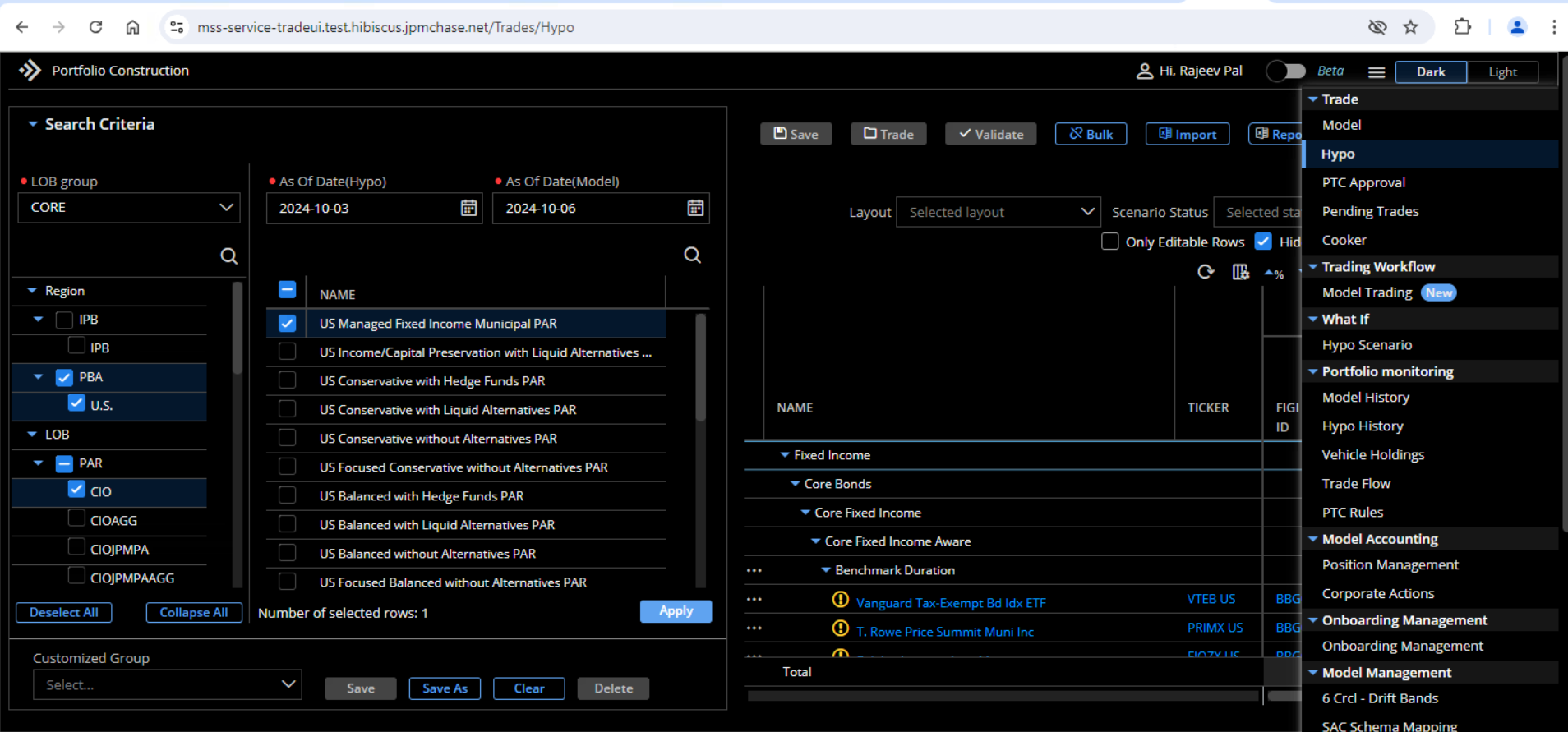
}

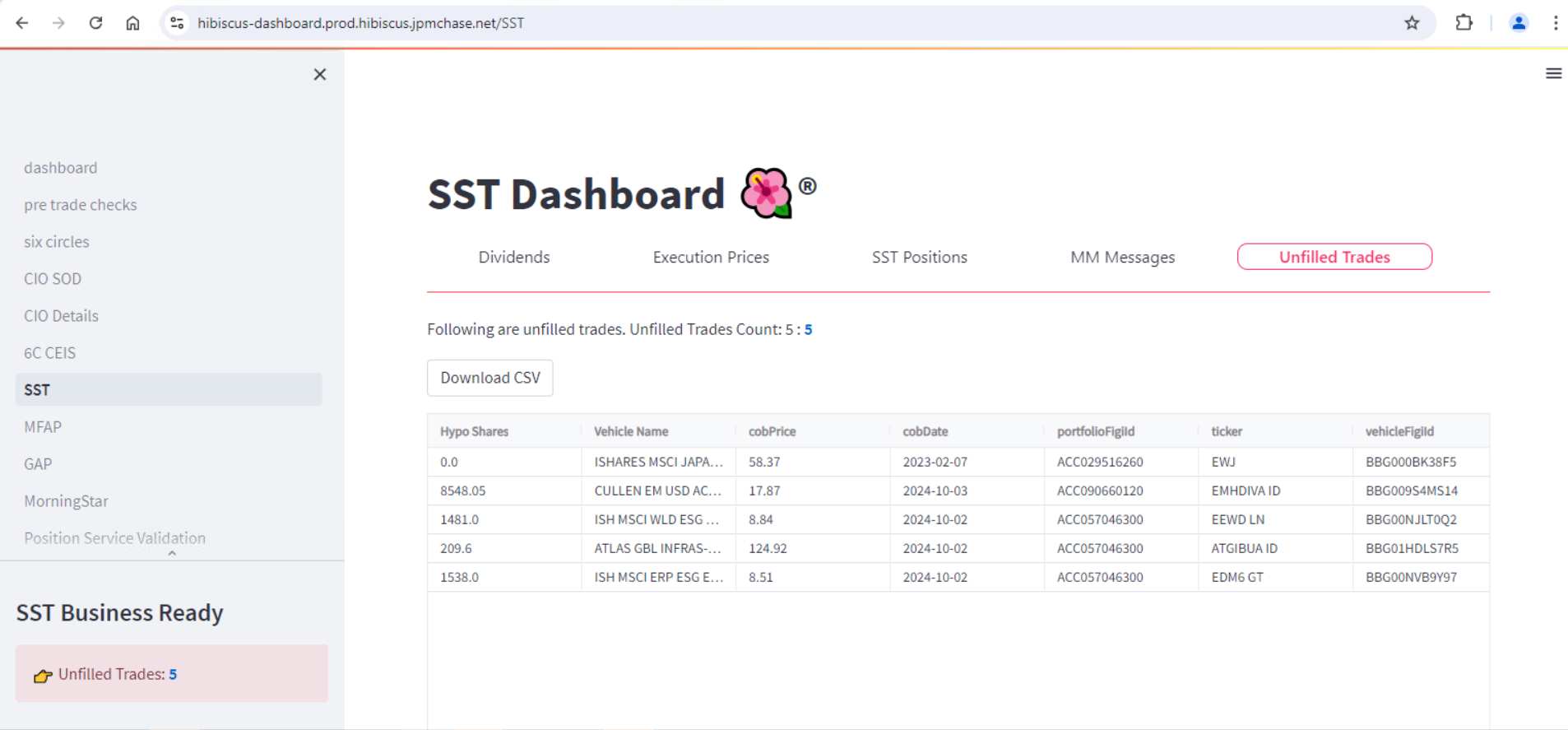


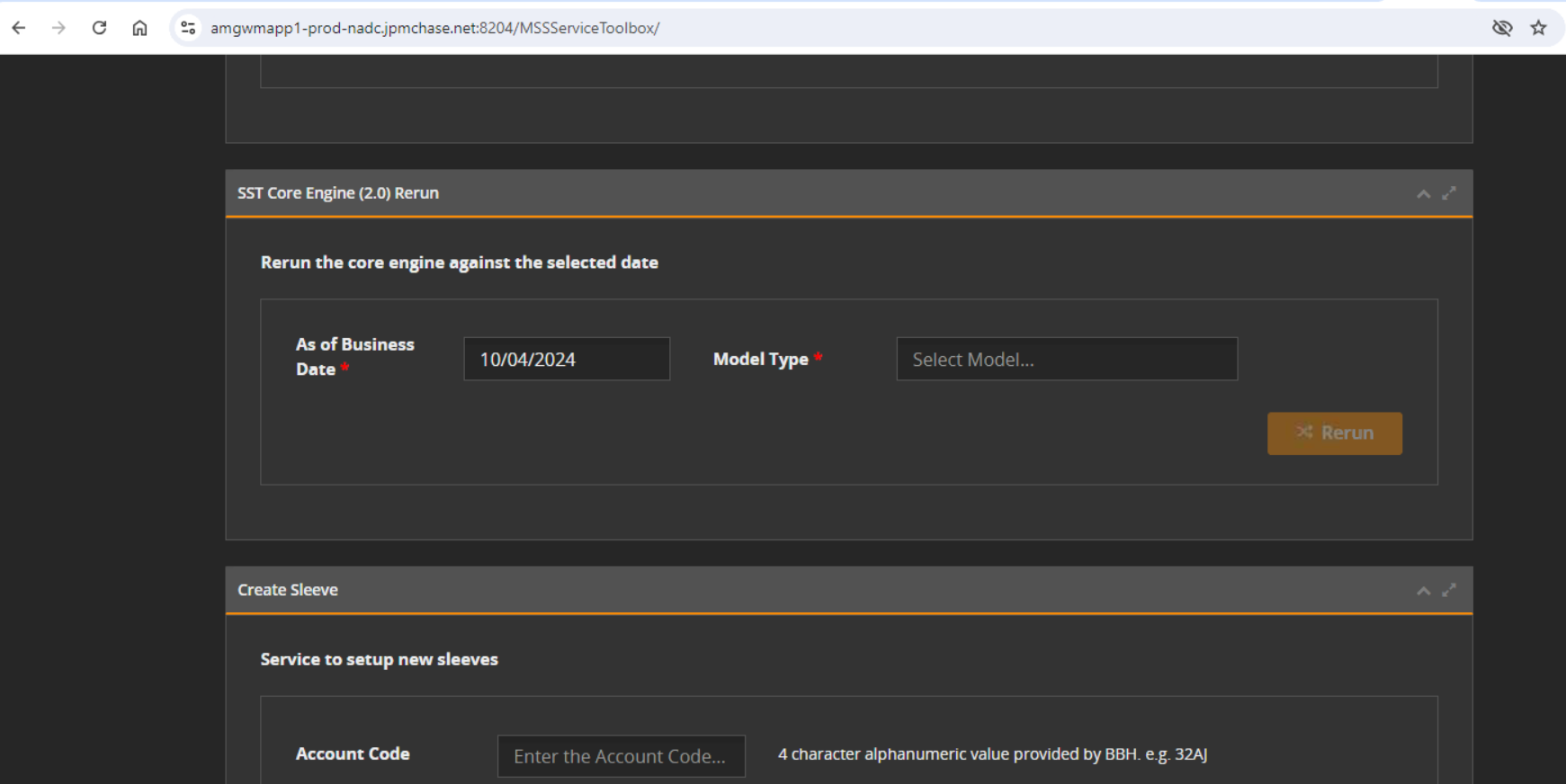


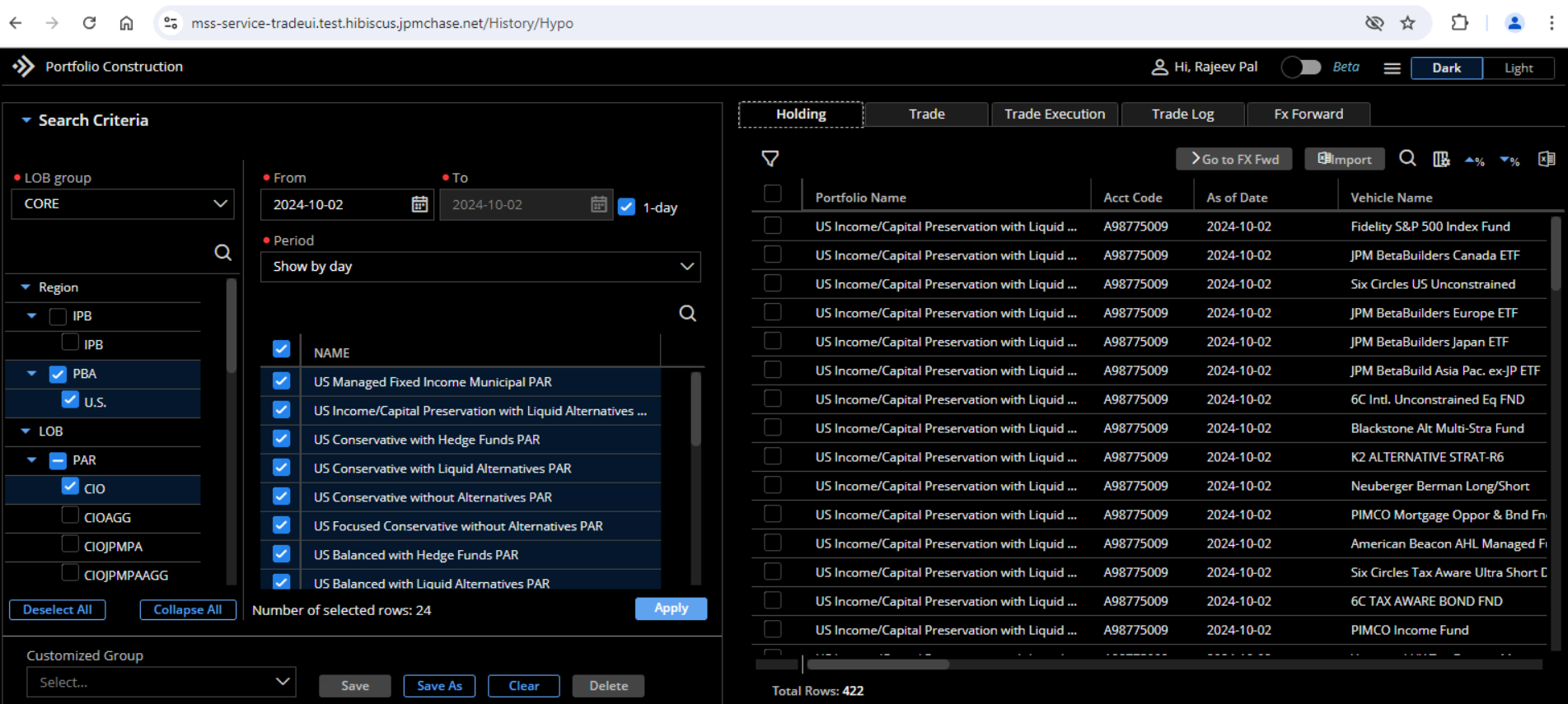


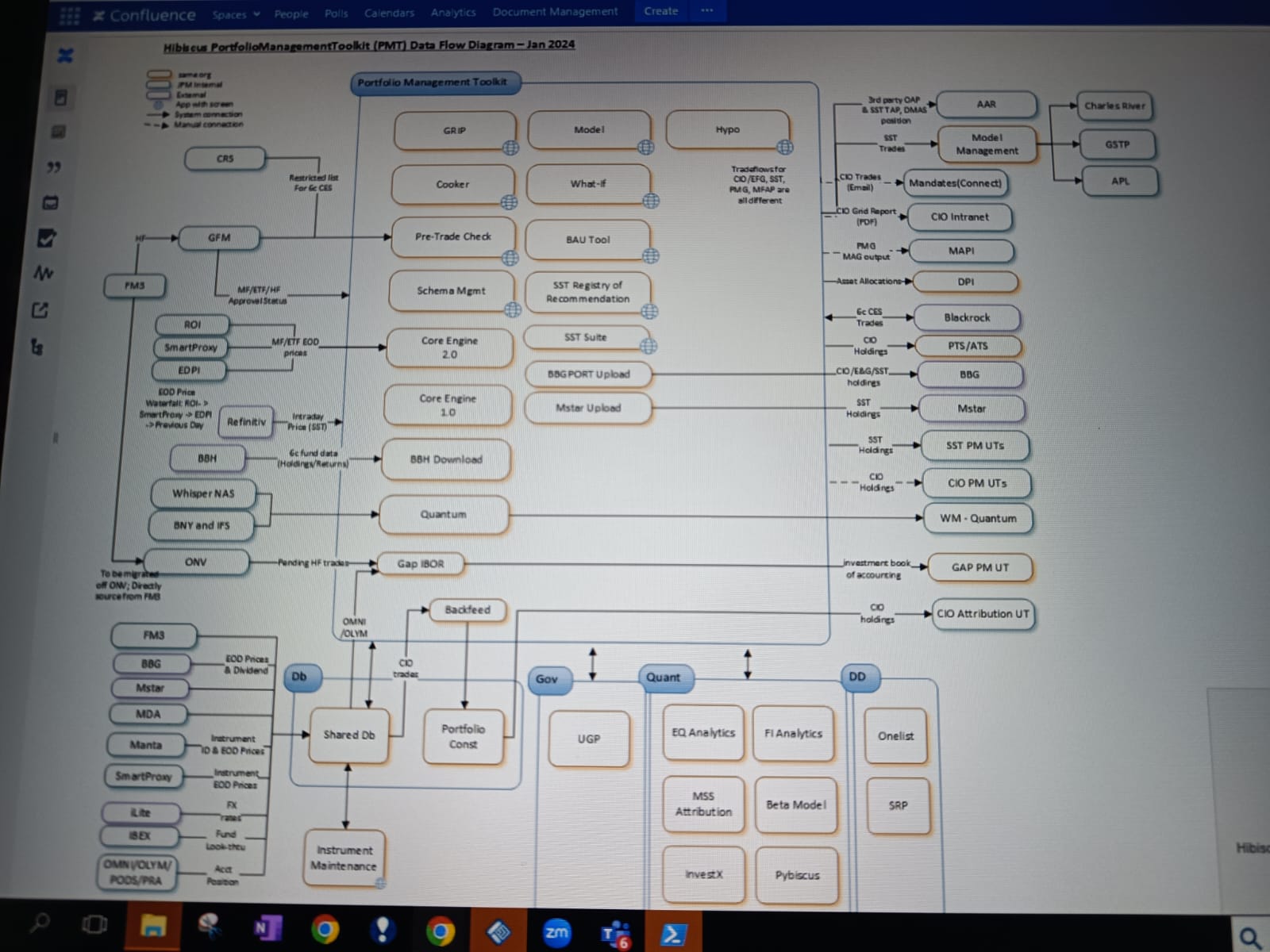












GKP stands for **Google Kubernetes Platform**. It is a managed service provided by **Google Cloud**, known more widely as **Google Kubernetes Engine (GKE)**, which offers a Kubernetes environment for deploying, managing, and scaling containerized applications.

**Key Information:**

* **Full Form:** **Google Kubernetes Platform (GKP)** (commonly referred to as **Google Kubernetes Engine (GKE)**).
* **Owner:** **Google Cloud** (a division of Google).
* **Description:** GKE is a managed Kubernetes service that automates many of the manual tasks involved in managing, deploying, and scaling containerized applications using Kubernetes. It simplifies cluster management and provides additional tools for security, monitoring, and scaling.

**Features:**

1. **Fully Managed:** GKE takes care of tasks like upgrading, monitoring, and securing Kubernetes clusters.
2. **Scalability:** It supports automatic scaling of your applications based on demand.
3. **Integration with Google Cloud Services:** Seamless integration with other Google Cloud services, such as Google Cloud Storage, Compute Engine, and networking tools.
4. **Security:** Built-in features for network security, secret management, and container vulnerability analysis.

**GKE vs GKP:**

While GKP is sometimes referred to in the context of Kubernetes, the more common term is GKE. So, it's possible that "GKP" could be used informally to mean the same thing, but "GKE" is the official product name.

The **Google Kubernetes Engine (GKE) Dashboard** is an interface in Google Cloud that allows you to monitor and manage Kubernetes clusters deployed within GKE. Through the dashboard, you can view detailed information about your clusters, nodes, workloads, services, and resources. GKE manages the underlying Kubernetes infrastructure, handling various aspects of cluster management, such as scaling, updates, and security.

**GKE Dashboard: Cluster Details**

When you access the **GKE dashboard** in the Google Cloud Console, you can view the following cluster-related details:

**1. Cluster Overview**

* **Cluster Name:** The name of the Kubernetes cluster.
* **Cluster Status:** Current status of the cluster (Running, Degraded, or Error).
* **Node Pools:** The set of nodes (VM instances) that comprise your cluster.
* **Location:** Where the cluster is running (regional or zonal).
* **Version:** The Kubernetes version your cluster is running.
* **Master Node:** A managed control plane (GKE handles the master nodes).
* **Network Settings:** Details of VPC (Virtual Private Cloud), IP ranges, and networking configuration.

**2. Node Pools**

* **Number of Nodes:** Total number of nodes in each pool.
* **Node Pool Status:** Status of node pools, such as running or scaling.
* **Node Types:** Machine types for each node (e.g., n1-standard-1, e2-standard-4).
* **Autoscaling:** Whether autoscaling is enabled or disabled for the node pool.
* **Upgrade Status:** Kubernetes version for node pools and upgrade options.

**3. Workloads**

* **Deployments:** Details of the number of Pods, desired replicas, current state, and Pod health.
* **DaemonSets:** Workloads that are running on every node in the cluster.
* **StatefulSets:** Workloads with persistent data that maintains a stable identity.
* **Jobs and CronJobs:** Batch processing workloads, including one-time and scheduled jobs.

**4. Services & Ingress**

* **Services:** Internal and external services that expose your workloads, along with their ClusterIP, ExternalIP, and type (LoadBalancer, ClusterIP, etc.).
* **Ingress:** Details on the Ingress controllers, which manage HTTP/HTTPS routing to services.

**5. Storage**

* **Persistent Volume Claims (PVCs):** Details of storage requests made by the Pods.
* **Persistent Volumes (PVs):** Physical storage associated with PVCs, typically on Google Cloud Storage.

**6. Logs & Monitoring**

* **Cloud Logging:** Integration with Google Cloud Logging to view logs for nodes and workloads.
* **Cloud Monitoring:** Metrics for resource usage, Pod health, and node performance.

**How GKE Manages Clusters in Kubernetes**

GKE abstracts many of the manual tasks involved in managing Kubernetes clusters. Here's how GKE manages clusters:

**1. Cluster Creation & Management**

* **Control Plane Management:** GKE handles the Kubernetes control plane (master nodes), including the Kubernetes API server, scheduler, controller manager, and etcd (Kubernetes’ key-value store).
* **Auto-upgrades:** GKE can automatically upgrade the control plane and node pool Kubernetes versions to ensure your cluster is running the latest features and security patches.
* **Regional and Zonal Clusters:** GKE allows you to run clusters in a single zone or multiple zones/regions for high availability.

**2. Autoscaling**

* **Cluster Autoscaler:** GKE automatically adjusts the size of your node pool based on the resource needs of your workloads.
* **Horizontal Pod Autoscaler:** Automatically scales the number of Pods in a deployment based on CPU utilization or other custom metrics.

**3. Security**

* **Workload Identity:** GKE integrates with Google Cloud Identity to manage authentication for workloads running in the cluster.
* **RBAC (Role-Based Access Control):** GKE enables granular control of cluster access through Kubernetes-native RBAC policies.
* **Node Auto-Upgrades:** GKE ensures nodes are kept up-to-date with security patches and the latest Kubernetes version.

**4. Networking**

* **VPC-Native Clusters:** GKE integrates with Google Cloud's VPC for advanced networking features like IP aliasing, private clusters, and load balancing.
* **Network Policies:** You can enforce network policies to control traffic between Pods or between Pods and external services.
* **Ingress Controllers & Load Balancing:** GKE provides built-in support for HTTP(S) load balancers and Google Cloud's Ingress controllers to manage traffic to your services.

**5. Logging & Monitoring**

* **Cloud Monitoring:** GKE automatically integrates with Google Cloud Monitoring, allowing you to visualize metrics, set up alerts, and gain insights into the performance of your cluster.
* **Cloud Logging:** Logs from your nodes and workloads are automatically sent to Google Cloud Logging, where you can view, filter, and analyze them.

**6. Scaling & Self-Healing**

* **Self-Healing:** GKE continuously monitors the health of the nodes and Pods and will restart or reschedule them if failures occur.
* **Pod Disruption Budgets:** GKE respects Pod disruption budgets when performing maintenance to avoid downtime during upgrades or scaling events.

**7. CI/CD Integration**

* **Google Cloud Build:** GKE can integrate with Cloud Build or Jenkins to automate continuous integration and deployment pipelines.
* **GitOps:** GKE can also support GitOps methodologies using tools like **Flux** or **ArgoCD** to deploy updates directly from your version control system (e.g., GitHub, GitLab).

**Gaia Cloud in Microservice Deployment**

**Gaia Cloud** is part of Oracle's internal cloud platform that focuses on **containerized microservices** and cloud-native applications. It was developed by Oracle to provide an advanced and reliable platform for deploying and managing microservices across different environments.

**Key Features of Gaia Cloud:**

1. **Containerized Applications:**
   * Gaia Cloud supports the deployment of applications that are packaged as containers (usually using Docker), making it easier to manage and scale microservices independently.
2. **Kubernetes Integration:**
   * It leverages Kubernetes (K8s) as the underlying orchestrator for managing these containers. Kubernetes manages the scheduling, scaling, and networking of microservices across the cloud infrastructure.
3. **Cloud-Native Architecture:**
   * Gaia Cloud is built around **cloud-native** principles, allowing the deployment of **distributed** microservices that are **fault-tolerant**, **resilient**, and **scalable**.
4. **Service Mesh Support:**
   * Service meshes like Istio or Oracle's internal service mesh solutions are used to manage microservice-to-microservice communication. This includes handling traffic, retries, circuit-breaking, and security (like mutual TLS).
5. **Oracle Cloud Infrastructure (OCI) Integration:**
   * Gaia Cloud is tightly integrated with **Oracle Cloud Infrastructure (OCI)**, enabling enterprises to deploy microservices on top of Oracle’s infrastructure for greater scalability, security, and performance.
6. **API Management:**
   * Gaia Cloud includes advanced API management features, allowing microservices to expose APIs easily while handling API versioning, access control, and rate limiting.
7. **Monitoring and Observability:**
   * With built-in observability tools, Gaia Cloud allows monitoring of microservices performance, availability, and logs. It integrates with popular tools like Prometheus and Grafana to offer insights into how services are performing.
8. **CI/CD Integration:**
   * Gaia Cloud integrates with continuous integration/continuous deployment (CI/CD) pipelines, making it easier to automate the deployment of new services and updates across different environments (staging, production, etc.).
9. **Security and Compliance:**
   * Oracle’s Gaia Cloud includes robust security mechanisms for microservices, including encryption at rest and in transit, identity management, role-based access control (RBAC), and compliance with industry standards.

**Use Case for Microservice Deployment:**

Gaia Cloud offers several capabilities to ease the process of deploying microservices:

* **Dynamic Scaling:** Microservices can be auto-scaled based on the traffic or resource consumption.
* **Fault Isolation:** In case of failures, only the affected microservices are impacted, without taking down the entire application.
* **Service Discovery:** Gaia Cloud provides built-in service discovery, ensuring that microservices can dynamically find and communicate with each other.
* **Continuous Deployment:** With seamless CI/CD integration, development teams can continuously deploy new features or fixes to specific microservices without redeploying the whole application.

**Who Owns Gaia Cloud?**

**Gaia Cloud** is developed and owned by **Oracle Corporation**. It is part of Oracle's suite of cloud-native tools designed to support microservice architecture in enterprises that require scalable, resilient, and secure cloud solutions.

**GKP VS EKS -**

**Amazon EKS (Elastic Kubernetes Service)** and **Google Kubernetes Engine (GKE)** are both managed Kubernetes services offered by AWS and Google Cloud, respectively. While both aim to simplify the deployment and management of Kubernetes clusters, they have distinct differences in terms of features, integration with their respective ecosystems, pricing, and performance.

**1. Provider Overview**

* **Amazon EKS:** A fully managed Kubernetes service by Amazon Web Services (AWS), designed for running Kubernetes clusters in the AWS cloud or on-premises via **EKS Anywhere**.
* **Google GKE:** A fully managed Kubernetes service by Google Cloud, known for being one of the first managed Kubernetes services and offering deep integration with Google’s cloud infrastructure and tools.

**What is Autosys Jo Scheduler-**

**AutoSys** is an enterprise job scheduling and workload automation tool used to define, schedule, monitor, and manage jobs or scripts on a variety of platforms (Windows, Unix, Linux, etc.). It helps automate the execution of complex workflows in IT environments by enabling users to schedule jobs, manage dependencies between jobs, and monitor their execution.

**Job Configuration in AutoSys**

**Defining a Job in AutoSys (using JIL):**

AutoSys jobs are typically configured using **Job Information Language (JIL)**, which is a command-line interface for defining and managing jobs. Below is a basic example of how a job is configured:

bash

Copy code

insert\_job: sample\_job\_name

job\_type: command

command: /path/to/your/script.sh

machine: target\_machine\_name

owner: user@domain

permission: gx,ge

date\_conditions: 1

days\_of\_week: mo,tu,we,th,fr

start\_times: "12:00"

description: "This is a sample AutoSys job."

std\_out\_file: /path/to/log/output.log

std\_err\_file: /path/to/log/error.log

condition: success(jobA)

In this example:

* **insert\_job**: Defines the job name.
* **job\_type**: Specifies the type of job (e.g., command, box, file watcher).
* **command**: The script or command to execute.
* **machine**: Specifies the target machine where the job will run.
* **owner**: The owner of the job.
* **date\_conditions**: When set to 1, indicates the job will be scheduled based on date or time.
* **days\_of\_week**: Specifies which days the job will run.
* **start\_times**: The time at which the job is set to start.
* **condition**: Specifies job dependencies (in this case, the job runs after jobA succeeds).

**Overview of Your Deployment Environment**

1. **Google Kubernetes Engine (GKE)**:
   * You are deploying your microservices using **GKE**, which is a managed Kubernetes service by Google Cloud that simplifies the deployment, management, and scaling of containerized applications.
2. **Global Load Balancer (GLB)**:
   * You create a **Global Load Balancer (GLB)** configured with **pure geographic DNS**. This setup routes traffic based on the geographic location of users, ensuring they are directed to the nearest available cluster for improved performance and reliability.
3. **MyCompute**:
   * **MyCompute** is a tool utilized in your environment to create and manage the GLB. It likely abstracts the complexities involved in setting up load balancers and configuring DNS, making it easier to handle these tasks within your infrastructure.
4. **Application Traffic Controller (ATC)**:
   * After the GLB is set up, you use the **Application Traffic Controller (ATC)** to manage the IP addresses that are associated with the GLB. This tool enables you to enable or disable specific IPs that are added to the load balancer.
5. **IP Management from GKE Dashboard**:
   * The IP addresses that ATC manages are sourced from the **GKE dashboard**. These IPs represent the cluster nodes or services related to your microservice deployment.

**Summary of Workflow**

* **Creation of GLB**: You use MyCompute to create a GLB that leverages pure geographic DNS for routing traffic effectively.
* **IP Configuration**: The IPs added to the GLB come from the GKE dashboard, which displays the relevant information about your deployed microservices.
* **Traffic Management**: The ATC tool allows you to dynamically manage the enabled or disabled status of these IPs in the GLB, ensuring efficient handling of application traffic.

**Conclusion**

This setup indicates a tailored approach to managing microservices deployment, load balancing, and traffic routing within your organization's cloud infrastructure. If you need further clarification on any of these components or additional assistance with your configuration, feel free to ask!

**How to connect to AWS s3:**

s3.endpoint-url=https://object-uat-na.jpmchase.net:8443

$3.region= NA

s3.bucket-name-8mno08el-bbg-up-test

s3.auth.type=KERBEROS

s3.key.preffix=sstBBG/

s3.key.preffix.mstar-sstMstar/

s3.auth.keys-api-url=https://osaf-oss-keys.gaiacloud.jpmchase.net/oss/api/v2/keys/

s3.auth.account-id=b9e43587-d6f9-44ac-882c-b02c0218cc9e

s3.auth.adfs-resource-id=JPMC:URI:RS-102548-11066-ObjectStorageService-PROD

#

adfs-client-id: PC-86626-K026882-209894-PROD

s3.auth.adfs-client-id=CC-86626-sixc\_uat-223661-PROD

s3.auth.adfs-provider-url=https://idag2.jpmorganchase.com/adfs/oauth2/token

s3.auth.proxyHost-object-uat-na.jpmchase.net

application:

s3:

auth:

keys-api-url: https://osaf-oss-keys.gaiacloud.jpmchase.net/oss/api/v2/keys/ account-id: 4b61777a-fb5e-4e6f-824e-b39dbc9f3451

adfs-resource-id: JPMC:URI:RS-102548-11066-ObjectStorageService-PROD

adfs-client-id: CC-86626-sixc\_prod-223662-PROD

adfs-provider-url: https://idag2.jpmorganchase.com/adfs/oauth2/token proxyHost: s3-prd1.object.nane.jpmchase.net

s3uat:

enabled: false

endpoint-url: https://object-uat-na.jpmchase.net:8443

bucket-name: 8mno08el-bbh-s3-test

auth:

keys-api-url: https://osaf-oss-keys.gaiacloud.jpmchase.net/oss/api/v2/keys/ username: AD\K026882

password: MercuryS3BBH!

account-id: b9e43587-d6f9-44ac-882c-b02c0218cc9e

adfs-resource-id: JPMC:URI:RS-102548-11066-ObjectStorageService-PROD

#

adfs-client-id: PC-86626-K026882-209894-PROD

adfs-client-id: CC-86626-sixc\_uat-223661-PROD

adfs-provider-url: <https://idag2.jpmorganchase.com/adfs/oauth2/token>

@Bean

@Profile("local")

public AdfsClientTokenProvider mercuryAdfsClientTokenProvider() {

return

AdfsClientTokenProvider.builder().id("adfs-s3-credential-provider").resourceUri(adfsResourceId)

.clientId(adfsClientId).providerTokenEndpointUrl(adfsProviderUrl)

.enableUsername PasswordAuthentication (username, password) //for running locally

.fetchTokenOnBuild(true).build();

}

@Bean

@Profile("!local")

public AdfsClientTokenProvider mercuryAdfsClientTokenProvider() {

return AdfsClientTokenProvider.builder().id("adfs-s3-credential-provider").resourceUri(adfsResourceId)

.clientId(adfsClientId).providerTokenEndpointUrl(adfsProviderUrl) .enableKerberosAuthentication().fetchTokenOnBuild(true).build();

}

@Bean("adfsS3ClientKeysProvider")

public AdfsS3ClientKeysProvider adfsS3ClientKeysProvider(

@Value("${application.s3.endpoint-url}") final String endpointUrl, @Value("${application.s3.bucket-name}") final String bucketName,

final AdfsClientTokenProvider mercuryAdfsClientTokenProvider) {

return AdfsS3ClientKeysProvider.builder().id("mercury-keys-provider")

.adfsTokenProvider (mercuryAdfsClientTokenProvider).accountId(accountId).buckets (bucketName) .keysEndpoint(keysApiUrl).keysSyncWaitPeriodInSeconds(40).readTimeout(Duration.of Seconds(90))

.dataplaneEndpoint(endpointUrl).build();

}

@Bean ("S3MercuryClient")

@Primary

public S3Client s3Client (@Value("${application.s3.endpoint-url}") final String endpointUrl,

final AdfsS3ClientKeysProvider adfsS3ClientKeysProvider) throws URISyntaxException {

final S3ClientBuilder s3ClientBuilder = S3Client.builder().endpointOverride (new URI (endpointUrl)) .region (Region.of( value: "ignored")).overrideConfiguration(ClientOverrideConfiguration.builder().putAdvancedOption(SdkAdvancedClientOption.SIGNER, AwsS3V4Signer.create()).

executionAttributes(ExecutionAttributes.builder().put(S3SignerExecutionAttribute.ENABLE\_PAYLOAD\_SIGNING, true).build()).build());

return AdfsS3Client.builder().adfsS3ClientKeysProvider(adfsS3ClientKeysProvider).s3ClientBuilder(s3ClientBuilder).build();

GET PUT DELETE

public class S3Service {

private final S3Client client;

private final MercuryS3Properties mercuryS3Properties = null;

private final AdfsClientTokenProvider tokenProvider;

public $3Service (S3Client client, AdfsClientTokenProvider adfsClientTokenProvider) {

this.client= client;

// this.mercuryS3Properties = properties;

this.tokenProvider = adfsClientTokenProvider;

}

public void deleteObject(String bucket, String key) {

final DeleteObjectRequest request = DeleteObjectRequest.builder().bucket(bucket).key(key).build();

client.deleteObject(request);

}

public Set<String> getBucketList(){

ListBucketsResponse response = client.listBuckets();

return response.buckets().stream().map(Bucket::name).collect (Collectors.toSet());

}

public void getObject(final String bucket, final String key, final ObjectStreamHandler objectStreamHandler) throws IOException {

final GetObjectRequest request = GetObjectRequest.builder().bucket(bucket).key(key).build();

try (final InputStream objectStream = client.getObject(request)) {

objectStreamHandler.handle(objectStream);

}

}

public List<S3File> getS3Files(String bucket) {

final ListObjectsV2Request request = ListObjectsV2Request.builder().bucket(bucket).build();

final ListObjectsV2Iterable response = client.listObjectsV2Paginator (request);

return response.contents().stream() Stream<S3Object>.map(S3File::of) Stream<S3File>.collect(Collectors.toList());

}

public void putObject(final String bucket, final MultipartFile file, final String namespace) throws IOException {

final PutObjectRequest request = PutObjectRequest.builder().bucket (bucket).key(!namespace.isEmpty()

? namespace +"/" +file.getOriginalFilename() : file.getOriginalFilename()).build();

final RequestBody requestBody = RequestBody.fromBytes (new ByteArrayInputStream(file.getBytes()).readAllBytes());

client.putObject(request, requestBody);

}

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>s3</artifactId>

<version>2.20.0</version> <!-- Use the latest version -->

</dependency>

import software.amazon.awssdk.core.sync.RequestBody;

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.s3.S3Client;

import software.amazon.awssdk.services.s3.model.PutObjectRequest;

import java.nio.file.Paths;

import java.io.FileWriter;

import java.io.IOException;

public class S3UploadCSV {

public static void main(String[] args) {

// Specify the region your S3 bucket is located in

Region region = Region.US\_EAST\_1; // Example region

S3Client s3Client = S3Client.builder().region(region).build();

// Bucket name and file details

String bucketName = "your-s3-bucket-name";

String key = "folder/csvfile.csv"; // S3 object key (including path)

String filePath = "localfile.csv"; // Path to your local file

// Create a CSV file (if you need to generate it)

createSampleCSV(filePath);

// Upload the file to S3

uploadFileToS3(s3Client, bucketName, key, filePath);

// Close the S3 client

s3Client.close();

}

/\*\*

\* Method to upload a file to an S3 bucket.

\*/

public static void uploadFileToS3(S3Client s3Client, String bucketName, String key, String filePath) {

try {

PutObjectRequest putObjectRequest = PutObjectRequest.builder()

.bucket(bucketName)

.key(key)

.build();

s3Client.putObject(putObjectRequest, RequestBody.fromFile(Paths.get(filePath)));

System.out.println("File uploaded successfully to S3: " + key);

} catch (Exception e) {

System.err.println("Error uploading file to S3: " + e.getMessage());

}

}

/\*\*

\* Method to create a sample CSV file locally.

\*/

public static void createSampleCSV(String filePath) {

try (FileWriter writer = new FileWriter(filePath)) {

writer.append("ID,Name,Email\n");

writer.append("1,John Doe,john.doe@example.com\n");

writer.append("2,Jane Doe,jane.doe@example.com\n");

System.out.println("CSV file created: " + filePath);

} catch (IOException e) {

System.err.println("Error creating CSV file: " + e.getMessage());

}

}

}

**Explanation:**

1. **Maven Dependency**:  
   Ensure that the AWS SDK for S3 (software.amazon.awssdk:s3) is added to your project dependencies.
2. **S3 Client**:  
   The S3Client object is initialized for a specific AWS region, which allows you to interact with S3.
3. **Create CSV File**:  
   The createSampleCSV() method creates a simple CSV file locally. If you already have the CSV file, you can skip this part and provide the file path directly.
4. **Upload to S3**:  
   The uploadFileToS3() method creates a PutObjectRequest to upload the CSV file to the specified S3 bucket. The file is uploaded using RequestBody.fromFile() which points to the local file path.
5. **Key**:  
   The key represents the path (including file name) where the object will be stored in S3.

**Prerequisites:**

* **AWS Credentials**:  
  Ensure that your AWS credentials are set up properly via the AWS CLI or through environment variables, credentials file, or IAM roles if running in an EC2 instance.
* **S3 Bucket**:  
  You need to have an existing S3 bucket where you can upload the file.

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.s3.S3Client;

import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;

import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;

import software.amazon.awssdk.services.s3.model.S3Object;

public class S3ListFiles {

public static void main(String[] args) {

// Specify the AWS region and create an S3 client

Region region = Region.US\_EAST\_1; // Example region

S3Client s3Client = S3Client.builder().region(region).build();

// Bucket name and optional prefix (folder path in S3)

String bucketName = "your-s3-bucket-name";

String prefix = "folder/"; // Optional: list files from a specific folder

// Call the method to list files

listFilesFromS3Bucket(s3Client, bucketName, prefix);

// Close the S3 client

s3Client.close();

}

/\*\*

\* Method to list files from an S3 bucket.

\*/

public static void listFilesFromS3Bucket(S3Client s3Client, String bucketName, String prefix) {

try {

// Build the ListObjectsV2Request

ListObjectsV2Request listObjectsRequest = ListObjectsV2Request.builder()

.bucket(bucketName)

.prefix(prefix) // Optional: specify folder prefix to list

.build();

// List the objects in the specified bucket

ListObjectsV2Response listObjectsResponse = s3Client.listObjectsV2(listObjectsRequest);

// Iterate through the list of S3 objects and print details

for (S3Object s3Object : listObjectsResponse.contents()) {

System.out.println("File: " + s3Object.key() + " (Size: " + s3Object.size() + " bytes)");

}

} catch (Exception e) {

System.err.println("Error listing files from S3: " + e.getMessage());

}

}

}

**Explanation:**

1. **AWS SDK Setup**:  
   The S3Client object is created using the specified AWS region. The region should match the one where your S3 bucket is hosted.
2. **ListObjectsV2Request**:  
   This request is used to list objects (files) from a specific S3 bucket. The prefix parameter is optional and is used to filter results by a folder path (for example, if you want to list files inside a specific folder).
3. **ListObjectsV2Response**:  
   The listObjectsV2 method returns a response containing the list of objects in the bucket. Each object is represented by an S3Object that contains details such as the file key (name) and size.
4. **Iterating over Files**:  
   The contents() method returns a list of S3Object instances, which represent individual files in the bucket. The key (file path) and size are printed for each file.

**Parameters:**

* **bucketName**: The name of your S3 bucket.
* **prefix** (optional): If you want to list files from a specific folder, you can provide a folder path as the prefix (e.g., "folder/"). If you don't want to use this, simply pass an empty string.

**Example Output:**

arduino

Copy code

File: folder/file1.csv (Size: 12345 bytes)

File: folder/file2.csv (Size: 67890 bytes)

File: folder/file3.txt (Size: 23456 bytes)

**Error Handling:**

This example includes basic error handling by catching any Exception and printing the error message. You can extend this with more robust error handling and logging as needed.

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.s3.S3Client;

import software.amazon.awssdk.services.s3.model.Bucket;

import software.amazon.awssdk.services.s3.model.ListBucketsResponse;

public class S3ListBuckets {

public static void main(String[] args) {

// Specify the AWS region and create an S3 client

Region region = Region.US\_EAST\_1; // Example region (optional)

S3Client s3Client = S3Client.builder().region(region).build();

// Call the method to list buckets

listS3Buckets(s3Client);

// Close the S3 client

s3Client.close();

}

/\*\*

\* Method to list all S3 buckets.

\*/

public static void listS3Buckets(S3Client s3Client) {

try {

// Fetch the list of buckets

ListBucketsResponse listBucketsResponse = s3Client.listBuckets();

// Iterate through the list of buckets and print their names

for (Bucket bucket : listBucketsResponse.buckets()) {

System.out.println("Bucket Name: " + bucket.name());

}

} catch (Exception e) {

System.err.println("Error listing S3 buckets: " + e.getMessage());

}

}

}

**Explanation:**

**Explanation:**

1. **S3Client Creation**:  
   The S3Client object is instantiated to interact with the S3 service. The Region is optional for listing buckets, but you can specify a region if desired. You can also omit the Region if you want it to automatically determine the appropriate region from your environment.
2. **Listing Buckets**:  
   The listBuckets() method of the S3Client is called to get the list of all buckets in the account. This returns a ListBucketsResponse object.
3. **Iterating Over Buckets**:  
   The buckets() method from ListBucketsResponse returns a list of Bucket objects. Each Bucket object contains details like the bucket name and creation date, though in this example, only the name is printed.
4. **Error Handling**:  
   A basic try-catch block is used to handle any exceptions that might occur during the operation.

**Example Output:**

yaml

Copy code

Bucket Name: my-first-bucket

Bucket Name: project-backups

Bucket Name: logs-bucket

**AWS Credentials:**

Ensure that your AWS credentials (access key and secret key) are configured properly using any of the standard methods like environment variables, AWS credentials file, or an IAM role if you're running this code on an AWS EC2 instance.

**Additional Features:**

* If you want to retrieve additional information about each bucket, like the creation date, you can access the creationDate() method of the Bucket object.
* The S3 buckets are global, so you don't need to specify a region to list them, but the region might be required for subsequent S3 operations like listing objects in a bucket.