Using Models in Production (Managed Cloud)

ESCORER

H2O.ai

Summary

The eScorer is a model inference server, that can support scoring H2O-3, Driverless-AI Mojo's and Python models concurrently.

The code base has been used by customers in production for over 3 years, one customer has deployed in ten countries which represented a x6.5 saving in runtime costs and reduced the response time latency by x5.5

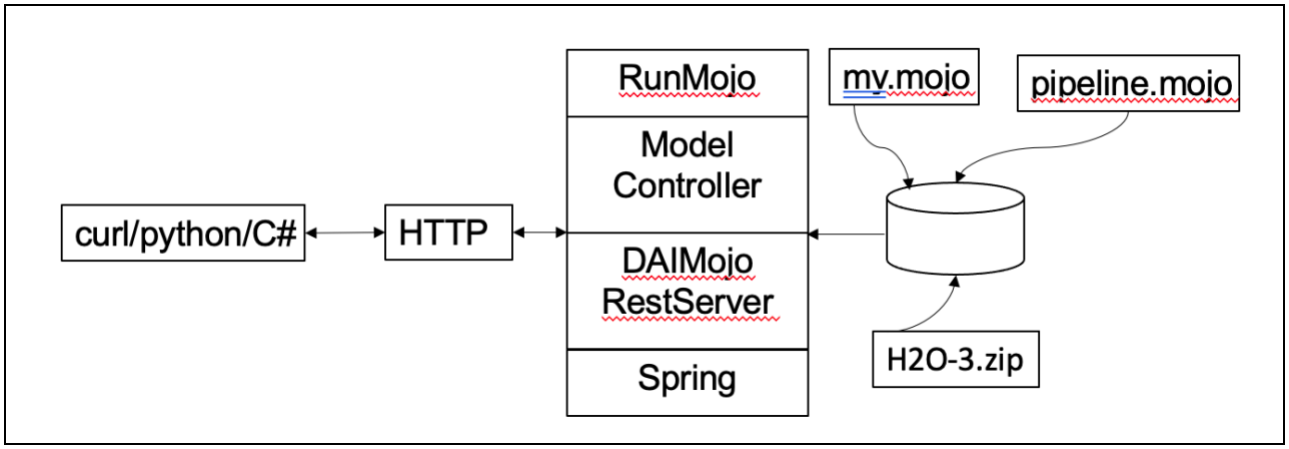
It is a base component for integration with Snowflake, AWS, GBQ, Qlik, Tableau, Pega and UiPath to name a few.

It has several features:

- Supports keycloak authentication and user roles, enabling segregation of duties
- Telemetry on scoring and performance per request/row
- Dynamically scalable and fully redundant in the Managed Cloud
- Several different calling formats are supported (uri, json, file)
- Calls can return predictions and or explainability in a single call
- Database scoring is supported directly from the eScorer against a customer's database
- A GUI is available that provides a way to access all the eScorer functions
- Autogen enables deployment artifacts to be generated for a model so that other applications or user personas can use the models.
- Support for structured and unstructured models (i.e tabular and images)

Rest Endpoint

The Rest Endpoint Scorer enables models from both Driverless AI and H2O-3 to be served from one common endpoint.



Distribution

The application is provided as a single jar file a properties file is generated so that environment specific options can be used.

Requirement

- Java JDK 1.8
- Driverless AI model (pipeline.mojo) or H2O-3 (mojo.zip) model
- Driverless AI license if using Driverless AI models
- Rest Server distribution

Hardware Sizing

The minimum CPU requirement is 4 CPU's either physical or vcores, as a rule of thumb a single CPU should be able to perform 1,000 predictions a second for a simple model.

Memory is used to cache the models that are being used for scoring, the total size of the model on disk can be used to estimate memory requirements.

- Sum in MB the size of all models that will be used for scoring
- Multiple that by 2x the number of CPU's
- Add 1024MB for the system for IO/OS pages

Setting the parameter unloadmodels to true (default false) will cause the model with the lowest invocation count to be flushed from memory.

For example: 1 x Mojo = 4MB on a 4 x CPU system

4 * (2x4) + 1024 = 1056MB
Round this to the common system sizes for example (8, 16, 32GB etc)
Disk is used to store the runtime currently ~170MB however optional logs can be enabled for Scoring, Audit and Monitoring see the specific parameters in this document in the Configuration sections.

Additionally collecting Standard out and Error logs is a good practice for the runtime, these can be saved locally or exported with standard SysLogNG configurations.

Load Balancing

Models and Rest Server instances are stateless, it's a best practice that production level instances have the appropriate number of servers / instances to gracefully handle a failure (hardware or software)

Most loan balancers perform a ‘health’ check on the instances to ensure the instance is available, the Rest Server has a ‘ping’ URL that can be called by the load balancer to verify availability.

Additionally the Monitoring functions in the Rest Server can be used to check operation of the instance, see the Monitoring section in this document.

Configuration

The configuration parameters can be specified either on the command line or in a properties file. The properties file makes it simpler to maintain and share settings across instances for example if running in a cluster.

Create Properties

A properties file should be created for each model, as each model may have different SQL to run, a different database to connect too or a different model name.

When the scorer runs to make predictions the properties file is passed on the command line, but first we must create the file.

```
java -Dcreateproperties=true -jar ai.h2o.mojos.jar > myproduction.properties
```

The above command will generate a properties file with the defaults that can then be changed for the environment. To use this file at runtime, use the following command line argument.

```
-Dpropertiesfilename=myproduction.properties
```

If no propertiesfilename is set the default properties file H2OaiRestServer.properties will be used, is no properties file exists then the program defaults will be used.

Create Autogen

The server can be executed to generate notebooks and terminate, rather than execute as an endpoint, this is used as easy way to create templates.

Parameter	Default	Description
createautogen		Name of model to use for notebook generation
createnotebook	SQL	Type of notebook
notebookfilename	Modelname+type	Provides a way to generate the output filename

The current directory is used as the location of the model, unless the command line parameter ModelDirectory is included on the command line.

Parameters

The properties file enables specific settings to be set for the environment.

ModelDirectory

This is the location of the Driverless AI models (*.mojo) and the H2O-3 models (*.zip). In Windows environments use a double backslash (\\) for the path.

```
ModelDirectory = /H2O.ai/Models/
```

Modelname

The default model name to use, if not model name is passed with a scoring request.

```
modelname = pipeline.mojo
```

SecureEndPoints

Defines the start of a uri for which authentication will be required.

```
SecureEndPoints = /modelsecure**
```

SecureEndPointsAllowedIP

This is an IP address for which requests are always allowed regardless of the SecureEndPoints setting. This is useful for specific management requests from a defined IP.

The default is for any request to be accepted.

```
SecureEndPointsAllowedIP = 0.0.0.0/0
```

Restuser

This is the default username for operations that require standard user access.

```
restuser = h2o
```

Restpass

This is the encrypted password for the restuser

```
restpass = aDJvMTIz
```

Adminuser

The username required for administrative requests to the server

```
adminuser = h2oadmin
```

adminpass

This is the encrypted password for the adminuser

```
adminpass = aDJvMTIz
```

ScorerHostIP

If this instance should forward recipes to a specific host this is the forwarding destination.

```
ScorerHostIP = 127.0.0.1
```

ScorerHostPort

Used in conjunction with the ScorerHostIP for supporting recipes.

```
ScorerHostPort = 9191
```

ScorerRecycle

If ScorerPool is set to zero (default) and ScorerRecycle is true (default) then a single Python model is executed and dynamically started and stopped as needed.

```
ScorerRecycle = true
```

ScorerPool

This is the number of Python HTTP Servers that are running in this instance. This also is the port index off the ScorerHostPort that is used for connections.

This allows multiple models to be started at the same time. Zero is the default, usually using a single instance with ScorerRecycle set to true is more maintainable except in in concurrent request to different models.

```
ScorerPool = 0
```

Scorelog

If enabled (true) then when a prediction is made the model name, UUID, input variables and predictions are written to standard out as part of an audit trail.

```
scorelog = true
```

Modelmonitor

If enabled (true) then when a prediction is made, a specially formatted line is written to the standard out file, this is then used by the Model Monitoring component.

```
modelmonitor = true
```

Extendedoutput

If enabled (true) then all the predictions are returned, otherwise only the first result is returned.

```
extendedoutput = true
```

Explainability

If enabled then the prediction will also return klime reason codes, if the model supports returning them.

```
explainability = false
```

ExplainabilityShapley

If set to true and if the model supports this function, then the shapley reason codes are returned with the prediction.

```
explainabilityShapley = false
```

ExplainabilityShapContrib

If set to true and if the model supports this function, then the shapley reason codes are returned with the prediction for Driverless AI models.

```
explainabilityShapContrib = false
```

ExplainabilityUsesFloats

Some older models use floats for numeric features.

```
explainabilityUsesFloats = true
```

DetectShift

This enables checking if the input row is different from the training, this requires the Experiment summary to be available on the Rest Endpoint.

```
DetectShift = false
```

DetectShiftTolerance

Acceptable difference as percentage, only valid if DetectShift enabled

```
DetectShiftTolerance = 0.0f
```

DetectShiftEnforce

If DetectShift is enabled and the input row is equal or larger than the DetectShiftTolerance value, then a blank response is sent and not the prediction.

```
DetectShiftEnforce = false
```

Cachefeatures

If the model is already loaded, we can use the models features, this saves processing time at the expense of memory to hold the list of features.

```
cachefeatures = true
```

UnloadModels

If memory is required to load a new model, but memory is limited, if the parameter unload is set to true, then the model with the lowest use count will be unloaded.

```
UnloadModels = true
```

RetryOnUnload

If enabled this option will cause the request that initialed the unloading of models to be retried, three attempts will be made, after that an exception (HTTP 500) error will be returned to the caller.

```
RetryOnUnload = true
```

PreloadModels

This is a list of model's comma separated that will be loaded at start up, otherwise models are loaded on demand.

```
PreloadModels=pipeline.mojo,mymodel.mojo
```

InputSeperator

If specified the feature separator this is usually not set.

```
InputSeperator = ,
```

SecureModel

The Rest Server can monitor for attacks on the model for example probing.

```
SecureModel = false
```

SecureModelLogging

If set to true, the request will be logged to standard out.

```
SecureModelLogging = false
```

SecureModelFeatureChangeDistance

Minimum value distance between feature request inputs.

```
SecureModelFeatureChangeDistance = 1000
```

SecureModelFeatureChangeNumber

Minimum number of features that should change per request.

```
SecureModelFeatureChangeNumber = 2
```

SecureModelFeatureChangePercent

Minimum feature change for this request

```
SecureModelFeatureChangePercent = 8
```

SecureModelNumberOfReqests

Number of requests from single source IP

```
SecureModelNumberOfReqests = 50
```

SecureModelAction

If true, blank result returned if suspicious request received

```
SecureModelAction = false
```

SecureModelTrackingAgeMS

How long to remember requests from the same source ip, defaults to 1 hour.

```
SecureModelTrackingAgeMS = 3600000
```

SecureModelTrackingSize

Maximum number of source locations to remember

```
SecureModelTrackingSize = 1000
```

SecureModelMSBetweenReqests

Minimum time between requests in milliseconds from the same source.

```
SecureModelMSBetweenReqests = 4000
```

setConvertInvalidNumbersToNa

Specific setting for H2O-3 models.

```
setConvertInvalidNumbersToNa = false
```

setConvertUnknownCategoricalLevelsToNa

Specific setting for H2O-3 models

```
setConvertUnknownCategoricalLevelsToNa = false
```

IgnoreEmptyStrings

For both Driverless AI and H2O-3 models, if the input is parsed but contains a empty string should the model receive an empty string.

```
IgnoreEmptyStrings = false
```

ResultLength

Truncates the returned prediction to this many digits, zero (0) disables truncation.

```
resultLength = 0
```

IncludeUUID

When writing scoring details to standard out, include the models UUID.

```
includeUUID = true
```

numberWorkers

A Snowflake specific setting that controls the number of threads used for scoring.

```
numberWorkers = 10
```

SnowflakeAllowedFunctions

A Snowflake specific setting, that ensures only requests from this function will be accepted.

```
SnowflakeAllowedFunctions = H2O.*
```

EnableModelBuild

A Snowflake specific setting, that enables model building for this instance.

```
EnableModelBuild = True
```

SecureModelAllowedAgent

A Snowflake specific setting, that will only accept requests from these specific source locations

```
SecureModelAllowedAgent = .*snowflake.*
```

SF_function

A Snowflake specific setting, that defines the template for returning populated SQL function create call.

```
sf_function =H2OPredicit
sf_url=<ADD_AWS_API_GATEWAY_NAME.execute-api.us-east-
1.amazonaws.com/<ADD_AWS_API_GATEWAY_STAGE_NAME
```

RemoveQuotes

If the input features are in quotes, should the quotes be removed before scoring.

```
RemoveQuotes = True
```

AutoGen

If auto generation of scoring code templates is enabled.

Setting Autogen = False also disables OpenAPI generation via http://hostname:port/v3/api-docs/

```
Autogen = True
```

AutogenHost

The endpoint used to generate the code; this has a separate interface for isolation

```
AutogenHost = 127.0.0.1:8080
```

AutogenType

This setting is used to allow or restrict what templates are available, for example maybe only Snowflake templates should be available.

```
AutogenHost = <valid regex expression>
```

Examples:

Expression	Description
.*	This is the default and lists all templates
.*Snowflake.*	Return only Snowflake templates
((?!PowerBI).)*$	Do not return templates for PowerBI

The following call will return all the available template types that are available:

```
curl -u h2o:h2o123 -s http://127.0.0.1:8080/autogen?notebook=listall
```

TempWorkingDir

This setting is used to define the location that the eScorer will use for operations where temporary files are required to be written and accessed.

```
TempWorkingDir=<valid directory location>
```

Notice, the setting ends with a / for eample /opt/H2O/temp/

Default Value

If not set and /opt/H2O/temp/ does not exist then the JVM runtime setting java.io.tempdir is used.

Executing the Server

Using the properties file to define specific runtime settings is the easiest way to run the server.

Adding a License

A DAI license is required to score, specify the license on the command line, using one of the following:

1. Environment variable DRIVERLESS_AI_LICENSE_FILE : A location of file with a license.
2. Environment variable DRIVERLESS_AI_LICENSE_KEY : A license key.
3. System properties of JVM (-D option) 'ai.h2o.mojos.runtime.license.file' : A location of license file.
4. System properties of JVM (-D option) 'ai.h2o.mojos.runtime.license.key' : A license key.
5. Classpath: The license is loaded from resource called '/license.sig' The default resource name can be changed via system property 'ai.h2o.mojos.runtime.license.filename'

Using a Properties File

The Server will read its settings from a properties file specified on the command line argument, propertiesfilename if the parameter is not passed then the default name H2OaiRestServer.properties is used.

If a properties file is not available, then the built-in defaults as documented in the *parameters* section of this document is used.

Example Override

```
-Dpropertiesfilename =production.properties
```

Causes the Server to use the settings in the file production.properties.

Server Parameters as Arguments

If required, the parameters in the properties file, can be specified on the command line using -D arguments, but this is a way of not using a properties so all the parameters come from the command line.

Example Override

If setting the ModelDirectory to a different location, then using the command line argument -DModelDirectory=/mnt/prod/models/ would

Override Ordering

This is the precedence of parameters.

1. Properties file
2. Command line (-D)
3. Server Default

Use Example

If you wanted a common properties file for all instances, but for business isolation reasons, the model's directory are different.

- Create a properties file, but comment out the ModelDirectory setting
- Add -DModelDirectory= to the command line with the instance's location

At runtime the Server will use the parameters from the properties file for all the common settings and the -D command line argument for the ModelDirectory.

Command Line

The Server needs only a few command line settings.

```
java  -Xms4g -Xmx4g -jar ai.h2o.mojos.jar
```

Memory Settings

Adding the -Xms (minimum) and -Xmx (maximum) memory for the process is a good idea, the -Xmx should not exceed 75% of available physical memory.

Server Overrides

The Server application will accept most Spring settings after the .jar parameter on the command line.

A few examples that are common to set:

Server Parameter	Definition
--spring.servlet.multipart.max-file-size=50MB	Used with file upload scoring
--spring.servlet.multipart.max-request-size=50MB	Used with file upload scoring
--server.tomcat.max-threads=10	Number of threads to use for scoring
--server.tomcat.accesslog.enabled=true	Additional logging
--server.port=9080	Change the listen port

Service Demon

The following is an example of how to setup the Server to start when the system boots and also use the Linux system commands to control its execution.

Start Script

Create a script that the 'root' user can execute for example StartRestServer.sh

```
#!/bin/bash
echo starting H2O.ai Rest Server > /var/tmp/StartRestServer.out
java -jar /root/ai.h2o.mojos.jar >> /var/tmp/StartRestServer.out
```

Correct the paths in the above, but the paths should be absolute
Make sure the script is executable cmod a+x StartRestServer.sh

Service Definition

Create a service definition for the Rest Server /etc/systemd/system/H2OaiRestServer.service

```
[Unit]
Description=H2O.ai Rest Server for Model serving
After=network.Target

[Service]
Type=simple
ExecStart=/root/StartRestServer.sh
TimeoutStartSec=0

[Install]
WantedBy=default.target
```

Service Commands

As root or sudo the following commands are available.

Command	Usage
systemctl daemon-reload	This will load the service definition when it created or edited above
systemctl enable H2OaiRestServer	Enables the service to auto start on the next reboot
systemctl start H2OaiRestServer	Use to manually start the Server
systemctl stop H2OaiRestServer	Use to manually stop the Server

Monitoring

Monitoring the runtime and model is critical in production environments, several functions in the server accomplish this task.

Management Beans

Java Management Beans (JMX) can be enabled so that any monitoring tool within the environment can collect runtime metrics.

Tools such as JConsole, VirtualVM as well commercial tools like NewRelic and Splunk can use JMX to collect metrics.

Enabling JMX

Add the following command line parameters to the Server and restart the server.

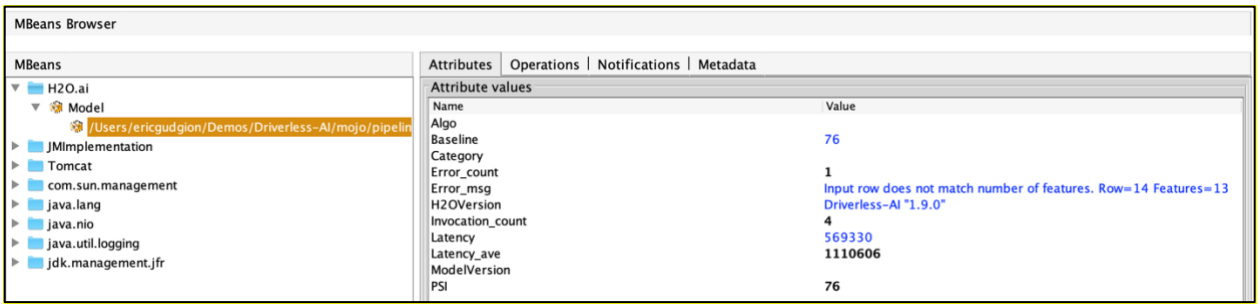
- -Dcom.sun.management.jmxremote.ssl=false
- -Dcom.sun.management.jmxremote.authenticate=false
- -Dcom.sun.management.jmxremote.port=8889

Reported Metrics

Name	Description
Algo	For H2O-3 Models reported algorithm
Baseline	Population stability on the results
Category	For H2O-3 models the model category
Error_count	Number of scoring errors
Errror_msg	Text of last error message
H2OVersion	Product version string
Invocation _count	Number of scoring requests
Latency	Last scoring latency in nanoseconds
Latency_ave	Average scoring latency
ModelVersion	If available this is the UUID of the model
PSI	Population stability on the results

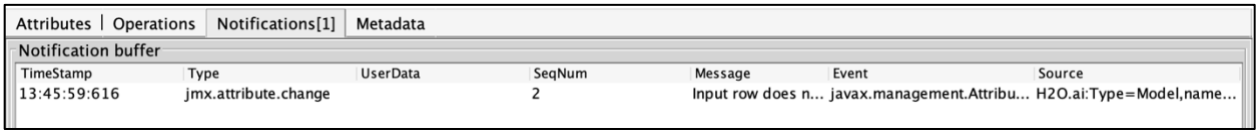
VisualVM Example

If using VisualVM this is how the H2O.ai management been looks.



Notifications

Applications that register for notification can receive alerts when errors occur at scoring time.



Model Monitoring

If the H2O.ai Model Monitoring component is available, then setting the option ModelMonitor = True will create log records for that tool.

Monitoring URI's

The Server has some specific URI's that provide monitoring details, these all require authentication at the Admin level.

URI	Description
/modeljmx	Java Management metrics via a http request
/modelstats	Similar metrics to the modeljmx request but at the thread level
/monitor	System, process and thread status details

Telemetry

The eScorer writes key metrics to standard out when executing at a fixed interval, this is in addition to the monitoring previously described.

Controlling the Frequency

The metrics are written by default at startup and then every 24 hours. The frequency can be overridden by adding --fixedDelay.in.milliseconds to the end of the command line parameters for example the following would set the frequency to 15 minutes:

```
--fixedDelay.in.milliseconds=900000
```

Telemetry Settings

The following settings define where the metrics are saved, these are defined in the properties file.

Parameter	Default	Description
TelemetryDB	None	JDBC connection string to DB instance
TelemetryTable	telemetry	Table where metrics are stored
TelemetryUser	None	Telemetry service account name
TelemetryPass	None	Telemetry service account password

Reported Metrics

This is an example of the output written to standard out.

```
Telemetry Processors: 16
Telemetry Free Memory: 4253431752
Telemetry Max Memory: 5368709120
Telemetry System Memory: 17179869184
Telemetry Number Loaded Models: 1
Telemetry Mojos Memory Bytes (approx available): 1115277368
Telemetry GC Details :G1 Young Generation 18 551
Telemetry GC Details :G1 Old Generation 0 0
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Latency 57766
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Invocation_count 2754
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Latency_ave 81834
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Error_msg
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Error_count 0
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, H2OVersion Driverless-AI "1.8.4.1"
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, ModelVersion
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Algo null
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Category null
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Baseline 10
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, PSI 10
Telemetry: H2O.ai:Type=Model,name=/opt/H2O/pipelineSF.mojo, Rows 0
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Latency 10129736870
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Invocation_count 46
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Latency_ave 5640058735
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Error_msg WARNING input row (Verfied,
income) contains internalFieldSeperator ,
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Error_count 46
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, H2OVersion null
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, ModelVersion null
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Algo null
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Category null
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Baseline 0
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, PSI 0
Telemetry: H2O.ai:Type=Model,name=pipeline191.mojo, Rows 1209899
```

Security

Secure the Server is critical, the server and models should not be directly exposed to the internet, but behind a firewall and IP routers to protect them.

Some additional configuration settings can further protect the server.

User Authentication

The Server has two users and matching roles, Admin and Restuser these are used to control what functions each can perform.

Set Passwords

Add the parameter -DsetPassword=true and start the server you will be prompted for the Default (restuser) and Admin passwords

```
Please enter password to use for Default user aaaa
set the parameters in properties file as follows:
RestUser = h2o
RestPass = YWFhYWFhYQ==
Please enter password to use for Admin user eeee
AdminUser = h2oadmin
AdminPass = ZWVlZWVlZWU=
```

Authenticated URI's

The parameter SecureEndPoint define the entry point where authentication is required, the default is /modelsecure** however if the parameter was changed to /** then any request would need authentication.

If authentication is required:

- If you are using a web browser, a logon panel will prompt you.
- If using a Rest API call, then pass a base64 encoded username password.

URI	Authenticated Role	Details
/snowflakeconfig	Admin	Used to configure external function
/modelupload	Admin	Uploads a model to the Server
/modeljmx	Admin	Returns JMX metrics
/modelstats	Admin	Returns thread level metrics
/monitor	Admin	Displays system level metrics
/modeldefence	Admin	Returns details about the model defense function
/modeldelete	Admin	Deletes a model from the running servers memory
/modelreload	Admin	Flushed all loaded models from memory
/version	Admin	Displays version information about the runtime
/modelsecure**	User	Full authenticated access is required

Enabling HTTPS

The Server can use a keystore, to enable this following these steps to create a JKS keystore

```
keytool -genkeypair -alias H2OaiRestServer -keyalg RSA -keysize 2048 -keystore keystore.jks -validity 3650 -storepass h2oh2o

keytool -importkeystore -srckeystore keystore.jks -destkeystore keystore.p12 -deststoretype pkcs12
```

The following settings from be added to the command line arguments of the startup command after the .jar.

```
--server.port=8443 --server.ssl.key-store-type=pkcs12 --server.ssl.key-store=keystore.p12 --server.ssl.key-store-password=h2oh2o --server.ssl.key-alias= H2OaiRestServer --security.require-ssl=true
```

IP Based Checking

The Server can limit access from a specific IP prefix, this allows ranges to be accepted or a specific IP.

For example:

SecureEndPointsAllowedIP	Details
0.0.0.0/0	All hosts. This is the default
127.0.0.0/1	Only local hosts
192.168.1.0/24	Only requests from this subnet
10.19.200.194	Only this specific host

Additional Settings

The server has additional setting described in the *Parameters* section of this document, review the following parameters.

Parameter	Description
SecureEndPoints	The URI root that requires authentication
SecureEndPointsAllowedIP	IP prefix-based access restriction
SecureModel	Tracks requests based on source IP
SecureModelAllowedAgent	Specific HTTP agent checking
Autogen	Consider disabling so that caller cannot retrieve templates and API details
SnowflakeAllowedFunctions	Rejects requests not using these functions

Audit Logging

The Server has the following logs that can be used for audit functions:

Server Access log

Add this parameter to the command line --server.tomcat.accesslog.enabled=true

Score Log

The writes a line for each prediction, see ScoreLog in the *Parameters* section.

Model Monitoring

A log record for each prediction, see ModelMonitor in the *Parameters* section.

KeyCloak Security

The eScorer has KeyCloak enabled by default, it supersedes the HTTP authentication / challenge by providing full authentication and authorization based on roles.

Runtime Parameters

The following parameters are used to define the KeyCloak Server and the Realm, these and any KeyCloak overrides can be added at the end of the java command line.

Parameter	Default	Description
--keycloak.auth-server-url	http://localhost:8080	location of KeyCloak server
--keycloak.realm	EScorer	Realm
--keycloak.resource	ScorerEndPoint	KeyCloak client-app name
--logging.level.org.keycloak	INFO	Set to DEBUG while installing to help verify operation

Environment Variables

The following environment variables can be used to pass specific Admin and User role names to override the default names.

```
export keycloakUserRoleName=new-user-rolename
export keycloakAdminRoleName=escorer-admin
```

These variables are used in the Keycloak Security Constraints definitions.

Roles

Roles enable specific authorization for functions to execute, this allows for admin and user functionality to be separated and controlled.

The following role names need to be defined in the Realm and users assigned to these roles. Users can belong to multiple roles if required.

Role Name	Description
escorer-user	User role
escorer-admin	Administrative function's role

Application Functions

The Roles enforce separation of functions across users within the application, all calls to the eScorer require an authenticated session with the escorer-user role ***except*** for the following:

Function	Role Name	Description
Logout		Enables session logout from KeyCloak
Modelstats	escorer-admin	retrieve model statistics
Modeljmx	escorer-admin	retrieve model JMX details via rest call
Monitor	escorer-admin	print JVM metrics to log
Modeldefense	escorer-admin	displays runtime details of model defense feature
Modelreload	escorer-admin	removes all models from memory cache and reloads on demand
Modeldelete	escorer-admin	deletes a specific model from memory cache
Modelupload	escorer-admin	upload a H2O-3 or Driverless-AI mojo model
Modeluploadpython	escorer-admin	upload a Driverless-AI python whl model
Snowflake		API to API call for external functions
SnowflakeBuild		Call a Driverless-AI instance to train a model
SnowflakeConfig	escorer-admin	Configure external function endpoint
Version	escorer-admin	print runtime version
Ping		URL to verify liveliness used by load balancers
Licenseupload	escorer-admin	load a Driverless-AI license
LicenseCheck	escorer-admin	verify the license

Security Constraints

The Application Functions detailed above map to specific roles, that can be configured at runtime using the following definitions.

```
--keycloak.securityConstraints[0].authRoles[0]=${keycloakUserRoleName}
--keycloak.securityConstraints[0].securityCollections[0].patterns[0]=/model
--keycloak.securityConstraints[0].securityCollections[0].patterns[1]=/modelh2o
--keycloak.securityConstraints[0].securityCollections[0].patterns[2]=/model/score
--keycloak.securityConstraints[0].securityCollections[0].patterns[3]=/model2JSON
--keycloak.securityConstraints[0].securityCollections[0].patterns[4]=/modeljson
--keycloak.securityConstraints[0].securityCollections[0].patterns[5]=/modelh2ojson
--keycloak.securityConstraints[0].securityCollections[0].patterns[6]=/modelchain
--keycloak.securityConstraints[0].securityCollections[0].patterns[7]=/mli
--keycloak.securityConstraints[0].securityCollections[0].patterns[8]=/modelsecure
--keycloak.securityConstraints[0].securityCollections[0].patterns[9]=/modelfeatures
--keycloak.securityConstraints[0].securityCollections[0].patterns[10]=/modeltargets
--keycloak.securityConstraints[0].securityCollections[0].patterns[11]=/score
--keycloak.securityConstraints[0].securityCollections[0].patterns[12]=/scoreimage
--keycloak.securityConstraints[0].securityCollections[0].patterns[13]=/scoredatabase
--keycloak.securityConstraints[0].securityCollections[0].patterns[14]=/invocations
--keycloak.securityConstraints[0].securityCollections[0].patterns[15]=/modelvars
--keycloak.securityConstraints[0].securityCollections[0].patterns[16]=/modeltext
--keycloak.securityConstraints[0].securityCollections[0].patterns[17]=/batch
--keycloak.securityConstraints[0].securityCollections[0].patterns[18]=/autogen
--keycloak.securityConstraints[0].securityCollections[0].patterns[19]=/qlik
--keycloak.securityConstraints[0].securityCollections[0].patterns[20]=/quaero
--keycloak.securityConstraints[0].securityCollections[0].patterns[21]=/googlebq
--keycloak.securityConstraints[0].securityCollections[0].patterns[22]=/modellist
--keycloak.securityConstraints[1].authRoles[0]=${keycloakAdminRoleName}
--keycloak.securityConstraints[1].securityCollections[0].patterns[0]=/modelstats
--keycloak.securityConstraints[1].securityCollections[0].patterns[1]=/modeljmx
--keycloak.securityConstraints[1].securityCollections[0].patterns[2]=/monitor
--keycloak.securityConstraints[1].securityCollections[0].patterns[3]=/modeldefense
--keycloak.securityConstraints[1].securityCollections[0].patterns[4]=/modelreload
--keycloak.securityConstraints[1].securityCollections[0].patterns[5]=/modeldelete
--keycloak.securityConstraints[1].securityCollections[0].patterns[6]=/upload
--keycloak.securityConstraints[1].securityCollections[0].patterns[7]=/modelupload
--keycloak.securityConstraints[1].securityCollections[0].patterns[8]=/modeluploadpython
--keycloak.securityConstraints[1].securityCollections[0].patterns[9]=/modelgetgcs
--keycloak.securityConstraints[1].securityCollections[0].patterns[10]=/snowflakeconfig
--keycloak.securityConstraints[1].securityCollections[0].patterns[11]=/version
--keycloak.securityConstraints[1].securityCollections[0].patterns[12]=/licenseupload
--keycloak.securityConstraints[1].securityCollections[0].patterns[13]=/licensecheck
```

KeyCloak Runtime Properties

The security constraints and other runtime options can be passed in a runtime.properties file this files default name is runtime.properties the name and location of the file can be passed with using the -Druntime.properties=/full-path/runtime.properties or in the H2OaiRestServer.properties file on the parameter runtimeproperties.

The options from the runtime.properties file are dynamically loaded and appended to the existing application arguments before the application is started.

Enable SSL/TLS

Highly recommend setting the following settings to enable these SSL/TLS for the server, also see the section *Enabling HTTPS.*

```
--server.port=8443 --server.ssl.key-store-type=pkcs12 --server.ssl.key-store=keystore.p12 --
server.ssl.key-store-password=h2oh2o --server.ssl.key-alias=H2OaiRestServer --
security.require-ssl=true
```

Calling the EndPoint

When calling a function, a token needs to be used.

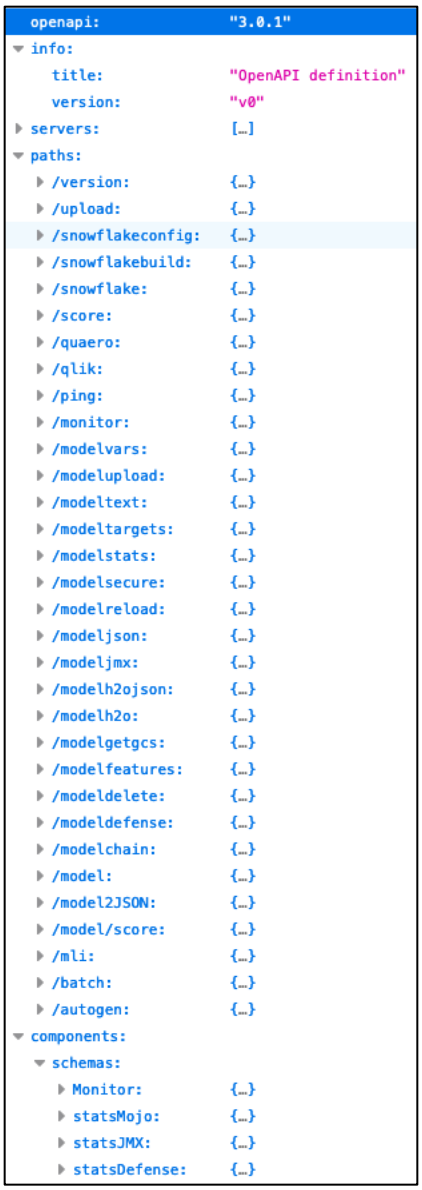
```
KEYCLOAK_TOKEN=$(curl --data
"grant_type=password&client_id=ScorerEndPoint&username=h2ouser&password=h2ouser"
http://kcserver:8080/realms/EScorer/protocol/openid-connect/token | jq -r '.access_token')

curl
"https://127.0.0.1:8443/model?name=pipelineSF.mojo&row=26918,%2036%20months,17,315.
94,3,38000,VERIFIED%20-%20income,NY,19.93,0,0,74.7" -H "Authorization: Bearer
$KEYCLOAK_TOKEN" --insecure
```

OpenAPI / Swagger

This enables the Rest API to be retrieved and used programmatically, if the parameter Autogen is set to false, then the OpenAPI function is disabled.

Use the following url to retrieve the API details http://hostname:port/v3/api-docs/



Supported Requests

The Server supports the following requests sent via a HTTP request.

Depending on the security settings, some will require authorization.

version

This returns the details about the Driverless AI or H2O-3 version that was used to build the specified model.

http://127.0.0.1:8080/version?name=/tmp/pipeline.mojo


```
0:  "Mojo2_Runtime "
1:  "Details of model /Users/ericgudgion/Demos/Driverless-AI/mojo//tmp/pipeline.mojo"
2:  "Java Version 11.0.10"
3:  "suffix=\"\""
4:  "build=\"143\""
5:  "commit=\"1e095cf2cf8fa1a1ff1d0f7e25b8d17e873f72dc\""
6:  "describe=\"1e095cf\""
7:  "build_os=\"Linux c2e9d96dbbf4 4.15.0-55-generic #60-Ubuntu SMP Tue Jul 2 18:22:20 UTC 2019 x86_64 GNU/Linux\""
8:  "build_machine=\"c2e9d96dbbf4\""
9:  "build_date=\"20191211\""
10: "build_user=\"2117\""
11: "base_version=\"1.8.1\""
12: "branch=\"rel-1.8.1 \""
13: "h2o_mli_version=\"0.1.106\""
14: "vis_data_server_version=\"2.0.2\""
15: "deployment_templates_version=\"1.0.4\""
16: "terraform_version=\"0.11.10\""
17: "mojo2_runtime_version=\"2.1.12\""
18: "h2o4gpu_version=\"0.3.2\""
19: "procsy_version=\"0.6.0\""
20: "pydatatable_version=\"0.9.0.dev119\""
21: "h2oai_version=\"\""
22: "h2oai_commit=\"1e095cf2cf8fa1a1ff1d0f7e25b8d17e873f72dc\""
23: "h2oai_commit_timestamp=\"2019-12-10_16:28:35_-0800\""
24: "docker_version_tag=\"latest\""
25: "branch_name=\"rel-1.8.1\""
26: "build_num=\"local\""
27: "my_cuda_version=\"10.0\""
28: "model2proto_version=\"2.1.12+master.106\""
29: "http_proxy=\"\""
30: "version=\"1.8.1\""
31: "normalized_version=\"1.8.1\""
```


Request Params

Parameter	Default	Details
Name	/tmp/pipeline.mojo	Name of Model

Example

HTTP

GET /version?name=/tmp/pipeline.mojo

upload

Uploads a CSV file to score, a file is returned with predictions appended to the end of each row of the file.

http://127.0.0.1:8080/upload?name=/tmp/pipeline.mojo& skipheader=false&type=2

Request Headers

Header	Value
Content-Type	application/json

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
row		Ignored
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
skipheader	false	Set to True if the first line is the file header
sortresults	false	If results should be sorted
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Body

Bodyraw (json)

JSON

```
{
  "file": "<binary>"
}
```

Example

HTTP

PUT

//upload?name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&skipheader=false&sortresults=false&type=2

snowflakeconfig

Used to configure the API endpoint that should be returned for Snowflake SQL Function requests. Refer to the Snowflake Integration documentation for more details: http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/snowflake-integration.html#snowflake-integration

http://127.0.0.1:8080/snowflakeconfig?function=&url=

Request Params

Parameter	Default	Details
function		Name of API gateway function used as a HTTP passthrough
url		Complete URL of gateway

Example

HTTP

PUT //snowflakeconfig?function=&url=

snowflakebuild

This is used by the Snowflake Integration, review documentation for more details: http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/snowflake-integration.html#snowflake-integration

http://127.0.0.1:8080/snowflakebuild?payload=<string>

Request Params

Parameter	Default	Details
payload		A Snowflake specific format for passing data

Example

HTTP

GET /snowflakebuild?payload=<string>

snowflake

This is used by the Snowflake Integration, review documentation for more details: http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/snowflake-integration.html#snowflake-integration

http://127.0.0.1:8080/snowflake?payload=<string>

Request Params

Parameter	Default	Details
payload		A Snowflake specific format for passing data

Example

HTTP

GET /snowflake?payload=<string>

score

This is used to pass a request to a Python HTTP service and is used to support Python only WHL and Recipe deployments. See *Recipe Scoring* section of this document for configuration details.
http://127.0.0.1:8080/score?payload=<string>

Request Params

Parameter	Default	Details
payload		JSON body to pass to external scoring HTTP listener

Example

HTTP

GET /score?payload=<string>

scoreimage

This is used to pass a request to a Python HTTP service and is used to support Python only WHL and Recipe deployments that are using images.
http://127.0.0.1:8080/scoreimage

Request Params

Parameter	Default	Details
Image		Name of image to upload
Replace	No	Replaces image if one exists on the current server

Example

HTTP

PUT / -F "image=@myimage.jpg" "http://n.n.n.n scoreimage?replace=true"

scoredatabase

This is used to pass a properties file and initiate the Database Scorer, see the *Database Scoring* section in this document and the separate *Database Scoring in Production Documentation.*
http://127.0.0.1:8080/scoredatabase

Request Params

Parameter	Default	Details
File		Name of database properties file
File		Name of Model
Replace	No	Replaces image if one exists on the current server

Example

HTTP
PUT / -F "file=@/ DAIMojoRunner_DB.properties" -F "file=@/ pipeline.mojo"http://n.n.n.n scoreimage?replace=true"

quaero

Used for integration with Quaero Customer Data Platform, used to return model predictions to the CDP engine for market targeting.

http://127.0.0.1:8080/quaero?name=/tmp/pipeline.mojo&row=

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
row		Encoded data csv seperated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
sortresults	false	If results should be sorted
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP
GET /quaero?name=/tmp/pipeline.mojo&row=

qlik

Used for integration with Qlik, so that real time predictions can be included in the Qlik reports a nd insights.

http://127.0.0.1:8080/qlik?name=/tmp/pipeline.mojo&row

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
row		Encoded data csv seperated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
sortresults	false	If results should be sorted
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP

```
GET /qlik?name=/tmp/pipeline.mojo&row
```

licenseupload

This enables a Driverless AI license to be uploaded to the eScorer, providing the paraemeter All owLicenseUpload is set to true in the configuration (true is the default).

http://127.0.0.1:8080/licenseupload

Request Params

Parameter	Default	Details
name	license.sig	Name of license file (not required)
Replace	false	Replace existing license file
verbose	false	Used to output additional logging on the Server

Example

HTTP

```
PUT -F “file=@license.sig” /licenseupload
```

licensecheck

If the eScorer is executing with with JAVA 11, then this will the number of days until the license expires to the Standard out log file.

http://127.0.0.1:8080/licensecheck

Request Params

Parameter	Default	Details
name	license.sig	Name of license file (not required)
verbose	false	Used to output additional logging on the Server

Example

HTTP

```
GET /licensecheck
```

googlebq

This is the endpoint used by the Google Big Query integration which enables SQL calls to invoke models.

http://127.0.0.1:8080/googlebq

Request Params

The calling and return parameters are programmatically defined by the Google function definition, refer to the *Google Big Query* section in the documentation.

Example

HTTP

```
PUT /googlebq
```

logout

This is used to release the session in KeyCloak

http://127.0.0.1:8080/logout

Request Params

None.

Example

HTTP

```
GET /logout
```

modellist

This lists the models that are available to the eScorer.

http://127.0.0.1:8080/modellist

Request Params

Parameter	Default	Details
type	0	Model types to list. 0=all, 1=H2O-3, 2=Driverless-AI
verbose	false	Used to output additional logging on the Server

Example

HTTP

```
GET /modellist
```

ping

Use this URI for health checks from a load balancer, this will return a HTTP 200 (OK) is the Server is running.

http://127.0.0.1:8080/ping

Example

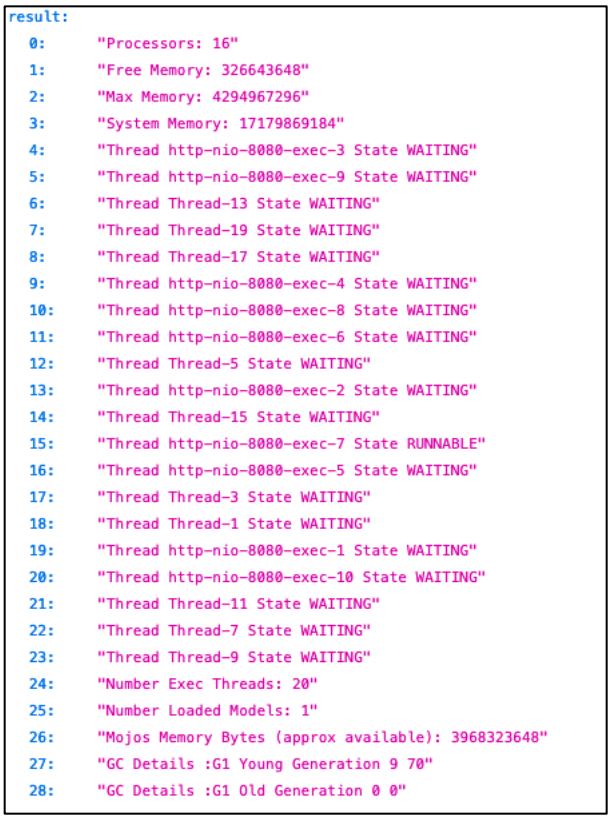
HTTP

GET /ping

monitor

Returns runtime details to the caller and prints them in the Server standard out.

http://127.0.0.1:8080/monitor



Example

HTTP

GET /monitor

modelvars

Used to call a model and pass the model feature names and feature values on the URL

For example, if the model feature was ***temperature*** then the call would be:

http://127.0.0.1:8080/modelvars?name=/tmp/pipeline.mojo&temperature=75

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
Model Feature		Varies by model
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP

GET /modelvars?name=/tmp/pipeline.mojo&type=2&verbose=false&explainability=false

modeluploaddriver

If a custom database JDBC driver is required for the Database Scorer, then an authorized user can upload a JDBC type 4 jar file using this call.

Request Params

Parameter	Default	Details
name		Name of jdbc driver file
verbose	false	Additional logging to help with any file operations
replace	false	Replace existing JDBC file

Example

curl -F "file=@/sqljdbc_6.0/enu/jre8/sqljdbc42.jar"
"http://127.0.0.1:8443/modeluploaddriver?&verbose=true&name=sqljdbc42.jar" -H
"Authorization: Bearer $KEYCLOAK_TOKEN" --insecure

modelupload

Uploads a model to the server. Models can be saved directly to the file directory (ModelDirectory) specified in the properties file or uploaded using a HTTP request, this is useful when deploying from a remote Driverless AI instance

http://127.0.0.1:8080/modelupload?name=pipeline.mojo&replace=false

Request Headers

Header	Value
Content-Type	application/json

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
replace	False	Overwrite existing model
verbose	False	Used to output additional logging on the Server
validate	True	Checks the file uploaded is a H2O-3 or Driverless AI mojo

The validate parameter is used to check that the uploaded file is a H2O-3 or Driverless AI mojo, if the file is not one of these types then the file is deleted and a HTTP 500 status code is returned.

Body

Bodyraw (json)
JSON

```
{
  "file": "<binary>"
}
```

Example

HTTP
PUT //modelupload?name=pipeline.mojo&verbose=false&replace=false

modelfetch

This will retrieve a Python model from an external location as defined in the pyFetchCMD system parameter, this enables a Driverless AI python model to be fetched and installed as a callable model.

Request Headers

Header	Value
Content-Type	application/json

Request Params

Parameter	Default	Details
name	python	Name of the .py file this model will be known
tta	False	Is this a time TTA model
verbose	false	Used to output additional logging on the Server

The System property pyFectchCMD defines the name of an existing python script to execute to fetch and install the python model. This is not a parameter that can be set in the properties files for security reasons, but instead is defined on the process (eScorer) so that the value or script cannot be compromised.

Example

HTTP

GET //modelfetch?name=pymodel.py

modeltext

Used to call a model, but only the first prediction target is returned as text. This is an easy way to integrate micro service calls, for example call the endpoint from Microsoft Excel, so that the prediction is retuned to a cell in the worksheet.

http://127.0.0.1:8080/modeltext?name=/tmp/pipeline.mojo&row=

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
row		Encoded data csv separated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	2	Model type (1 = H2O-3 2 = Driverless AI)

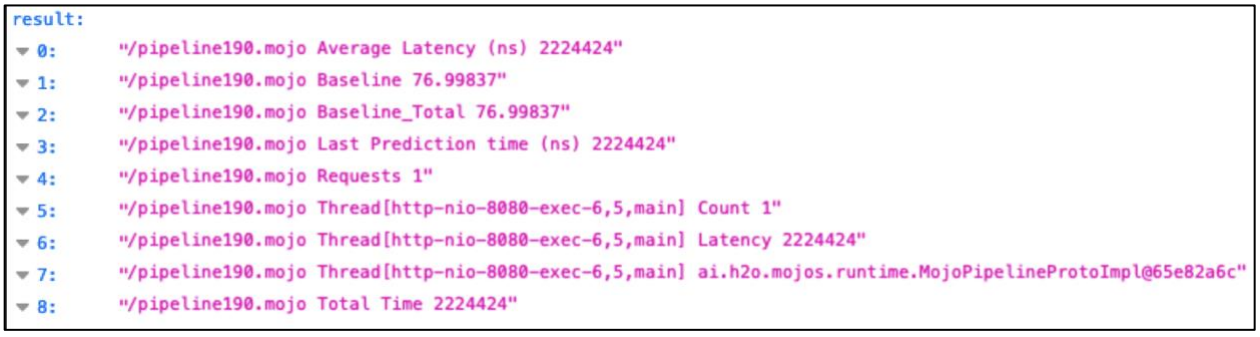
Example

HTTP
GET
/modeltext?name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&type=2

modelstats

Displays metrics on the models the Server has executed since starting.

http://127.0.0.1:8080/modelstats



Example

HTTP
GET /modelstats

modelsecure

Requests to this URI require authentication as a User Role.

http://127.0.0.1:8080/modelsecure?name=/tmp/pipeline.mojo&row

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
row		Encoded data csv separated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP

```
GET
/modelsecure?name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&sort
results=false&type=2
```

modelreload

All models that are in memory will be flushed, requests to score will cause the model to be load ed from the ModelDirectory.

http://127.0.0.1:8080/modelreload

Example

HTTP

```
GET /modelreload
```

modeljson

Calls a Driverless AI model with a JSON payload, the payload defines all the options.

http://127.0.0.1:8080/modeljson?payload=<string>

Request Params

Parameter	Value	Details
Payload		JSON payload

JSON Payload Format

```
{
 "name": "pipeline.mojo",
 "verbose": "true",
 "explainability": "false",
  "params": {
        "row": [
        ]
        }
}
```

Example

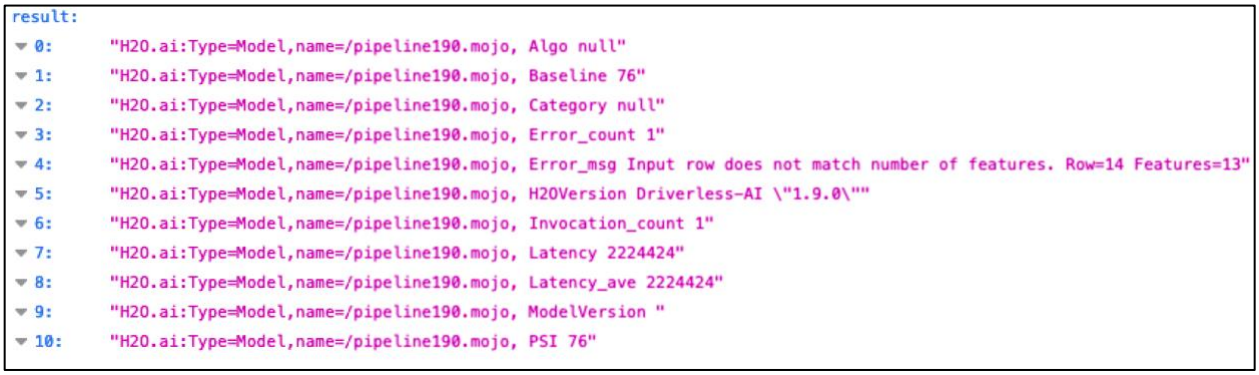
HTTP

GET /modeljson?payload=<string>

modeljmx

Displays the Servers management bean.

http://127.0.0.1:8080/modeljmx



Example

HTTP

GET /modeljmx

modelh2ojson

Call a H2O-3 model with a JSON payload.

http://127.0.0.1:8080/modelh2ojson?payload=<string>

Request Params

Parameter	Value	Details
Payload		JSON payload

JSON Payload Format

```
{
  "name": "pipeline.zip",
  "verbose": "true",
  "explainability": "false",
   "params": {
          "row": [
          ]
          }
}
```

Example

HTTP

GET /modelh2ojson?payload=<string>

modelh2o

Call a H2O-3 model for scoring.
http://127.0.0.1:8080/modelh2o?name=/tmp/mojo.zip&row

Request Params

Parameter	Default	Details
name	/tmp/mojo.zip	Name of Model zip file
row		Encoded data csv separated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	1	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP
//modelh2o?name=/tmp/mojo.zip&row=&verbose=false&explainability=false&sortresults=false&type=1

modelgetgcs

Used on the Google Compute Service to retrieve a model and save it to the Server.

http://127.0.0.1:8080/modelgetgcs?file=<binary>&name=pipeline.mojo&verbose=false&replace=false

Request Params

Parameter	Default	Details
File		Name of file on GCS
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
replace	False	Overwrite existing model
verbose	false	Used to output additional logging on the Server

Body

Bodyraw (json)
JSON

```
{
  "file": "<binary>"
}
```

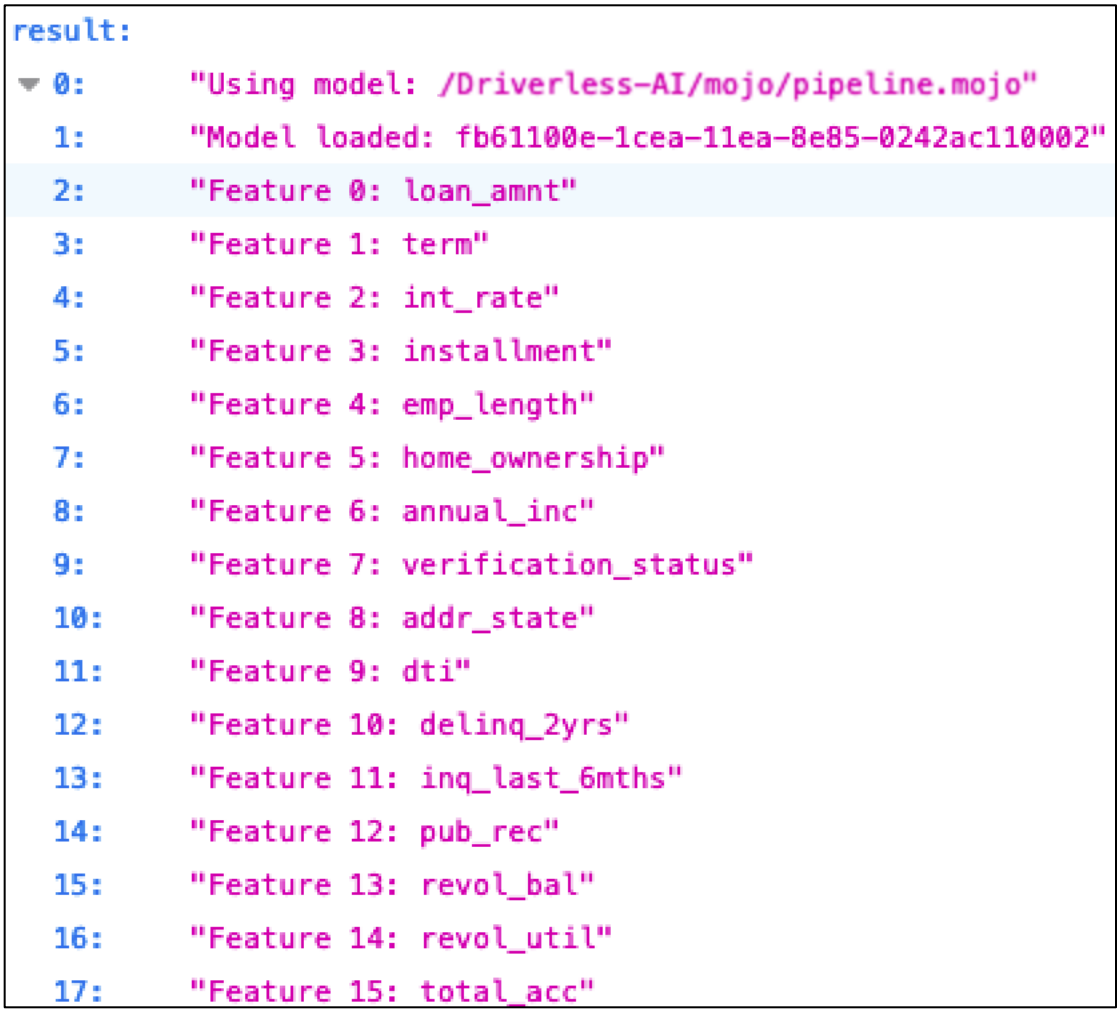
Example

HTTP
GET /modelgetgcs?file=<binary>&name=pipeline.mojo&verbose=false&replace=false

modelfeatures

Display the models features.

http://127.0.0.1:8080/modelfeatures?name=/tmp/pipeline.mojo&type=2&verbose=false



Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
type	2	Model type (1 = H2O-3 2 = Driverless AI)
verbose	false	Used to output additional logging on the Server
primitive	false	Return field type

Example

HTTP

GET /modelfeatures?name=/tmp/pipeline.mojo&type=2&verbose=false

modeldelete

Flushes the specified model from the Servers memory.

Used as an alternative to /modelreload which flushes all models, this replaces just one.

http://127.0.0.1:8080/modeldelete?name=/tmp/pipeline.mojo

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file

Example

HTTP

GET /modeldelete?name=/tmp/pipeline.mojo

modeldefense

If modelsecure is enabled in the properties file, then details of the request source are available.

http://127.0.0.1:8080/modeldefense

Example

HTTP

GET /modeldefense

modelchain

This call enables multiple models to be used to score the row in one request.

For example, if a user wanted to compare the prediction from one model vs a new model with t he same data in one call.

http://127.0.0.1:8080/modelchain?names=/tmp/pipeline.mojo&row

Request Params

Parameter	Default	Details
name	/tmp/mojo.zip	Name of Model zip file
row		Encoded data csv separated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP
GET
/modelchain?names=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&sort results=false&type=2

model/score

Accepts a request in the format used by the Driverless AI Rest API.

http://127.0.0.1:8080/model/score

Request Params

The payload defines which model and the data to use for scoring.

JSON Body Format

The ***model-feature*** is unique to the specific model.

```
{
 "id": 5,
 "method": "pipeline.mojo",
 "params": {
        "row": {
               "model-feature": "feature-value"
               }
        }
}
```

Example

HTTP
GET
/model/score?name=pipeline.mojo&type=2&verbose=false&explainability=false&payload=<string>

model

This allows an encoded row to scored.

http://127.0.0.1:8080/model?name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&sortresults=false&type=2

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model file
row		Encoded data csv separated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP

GET

/model?name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&sortresults=false&type=2

model2JSON

This call is like /model but the returns prediction is in a standard JSON format. For useful where the input is a data row but the output should be JSON.

http://127.0.0.1:8080/model2JSON?name=/tmp/pipeline.mojo&row=C1,C2&verbose=false&explainability=false&sortresults=false&type=2

```
{“target.0”:”prediction”,”target.1”,”prediction”}
```

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model file
row		Encoded data csv separated
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP

GET

/model2JSON?name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&sortresults=false&type=2

mli

Returns the klime reason codes. Usually this is more efficient to use the explainability option on the original scoring request.

http://127.0.0.1:8080/mli?name=/tmp/klime_mojo.zip&row=

Request Params

Parameter	Default	Detail
name		/tmp/klime_mojo.zip
row		Data to pass to klime
verbose	false	Used to output additional logging on the Server

Example

HTTP

GET /mli?name=/tmp/klime_mojo.zip&row=&verbose=false

score

This allows a Python Model to be called from a pool of running HTTP scorers

Request Params

Parameter	Default	Details
name	pipeline.mojo	Name of Model file
verbose	false	Used to output additional logging on the Server
explainability	false	If reason codes should be appended to each line
id	1	Used for legacy compatibility
method		One of the following: score, scorecontrib, scorecontriborig

Example

HTTP

```
rest-scorer.h2o.ai/score --header "Authorization: Bearer $KEYCLOAK_TOKEN" --Header
"Content-Type: application/json" --data @- <<EOF
{
  "id": 1,
  "method": "scorecontrib",
  "verbose":"false",
  "params": {
    "row": {
      "feature": "value”
    }
  }
}
EOF
```

scoreimage

This calls the Hydrogen Torch image model to score a single or batch of images.

Request Params

Parameter	Default	Details
name		Name of Model file
verbose	false	Used to output additional logging on the Server
Returntype	image	Image return a per image prediction. Average returns the average prediction across the images in the batch
Replace	True	Replaces the image if it's on the file system
id	1	Used for legacy compatibility
Method	Score_batch	Score_batch is the only supported value

Example

```
HTTP
rest-scorer.h2o.ai/score --header "Authorization: Bearer $KEYCLOAK_TOKEN" --Header
"Content-Type: application/json" --data @- <<EOF
{
  "id": 1,
  "method": "score_batch",
 "params": {
    "row": {
      "feature": "value”
    }
  }
}
EOF
```

scorefile

This call retrieves a file from an external source and then will call the Python C++ model to score the file.

Request Params

Parameter	Default	Details
name		Name of Model file
verbose	false	Used to output additional logging on the Server
Download	False	Used by the Python to indicate if the file/model should be downloaded
Contribs	False	If Shapley contributions should be returned
Contribs_original	False	If Shapley original contributions should be returned
Infile		Input filename
Outfile		Output filename

Example

HTTP

```
rest-scorer.h2o.ai/score --header "Authorization: Bearer $KEYCLOAK_TOKEN" --Header
"Content-Type: application/json" --data @- <<EOF
{
  "id": 1,
  "method": "score_batch",
 "params": {
    "row": {
      "feature": "value”
    }
  }
}
EOF
```

batch

This is similar to the /upload uri except only the predictions are returned not the original row.

http://127.0.0.1:8080/batch?file=<binary>&name=/tmp/pipeline.mojo

Make things easier for your teammates with a complete request description.

Request Params

Parameter	Default	Details
name	/tmp/pipeline.mojo	Name of Model either Mojo or zip file
row		Ignored
verbose	false	Used to output additional logging on the Server

explainability	false	If reason codes should be appended to each line
skipheader	false	Set to True if the first line is the file header
sortresults	false	If results should be sorted
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Body

Bodyraw (json)

JSON

```
{
  "file": "<binary>"
}
```

Example

HTTP

GET

/batch?file=<binary>&name=/tmp/pipeline.mojo&row=&verbose=false&explainability=false&sortresults=false&type=2

autogen

This is used to automatically generate example code on how to send scoring requests. The setting autogen = true must be in the properties file for this request to return an example.

The list of support types:

Type	Details
Listall	Returns a list of available types (run this for the complete list)
Snowflake_SQL	Snowflake SQL for using the model in a worksheet
PY	Python code for scoring using either the native library or a remote endpoint
IPYNB	Jupyter notebook showing how to use the model
Snowpark	Scala using Snowpark to score table using local model
Snowpark.scala_score	Score in Snowflake using the JavaUDF
Snowflake.UDF	Snowflake UDF SQL definition
Snowflake.ipynb	Jupyter notebook to use model in Snowflake
SnowSQL	Generate SnowSQL commands to upload model for UDF
Data.marketplace	Example of using Snowflake Data Marketplace with scala
Powerbi	Python code to call model from PowerBI
PowerBI.web	Web template to use in PowerBI dashboard to call model
Python	Pure Python scoring code for the model
Jupyter	Jupyter notebook
Tableau	Tableau definition
Curl	Generate curl syntax for different ways to get predictions
Test_URL	Generate a test url to verify the availability for web browser
Sagemaker-Jupyter	Create a notebook for a model hosted in Sagemaker
Sagemaker-Container	Create a Sagemaker container for the specified model
Sagemaker-start	Stsart script for SageMaker integation
Sagemaker-Commands	AWS / Docker commands required to deploy model
RedShift-SQL	AWS RedShift database integration and example SQL

The autogen type listall can be used to list all the configured templates, see AutogenType for configuration details.

```
curl -s "http://127.0.0.1:8080/autogen?notebook=listall"
```

Additionally, OpenAI/Swagger can also be invoked to return the Server Rest API syntax see *OpenAPI* section in this document.

http://127.0.0.1:8080/autogen?name=pipeline.mojo&notebook=SQL

Request Params

Parameter	Default	Detail
name	pipeline.mojo	Name of model to generate template
notebook	SQL	Type of template
verbose	false	Used to output additional logging on the Server
type	2	Model type (1 = H2O-3 2 = Driverless AI)

Example

HTTP

GET /autogen?name=pipeline.mojo&notebook=SQL&verbose=false&type=2

invocations

Accepts a request in the format used by Amazon Sagemaker Inference Server

http://127.0.0.1:8080/invocations

Request Params

The payload defines which model and the data to use for scoring.

JSON Body Format

The body is passed from Sagemaker to the model, name, verbose, explainability and data supported in the JSON body.

Note: the data is a single quoted coma separated string.

{"name":"pipeline.mojo","verbose":"true","data":"feature-value1, feature-valu2”}

Example

HTTP

GET /invocation

Sending Requests

Can be sent to the Server using any language or http client, the goal is to make calling the server flexible and support different request and response formats.

As described in the *Security* section some requests require authentication.

Kubernetes

Executing the Server in a Kebernetes environment is very simple, models can either be saved to a local storge for the container or a shared location.

Additionally using the /modelupload call, models can be dynamically deployed as needed. Using the /ping uri form a load balancer is also useful to verify availability.

Docker Container

Here is a simple dockerfile configuration of the minimium settings, review the section on *Override Ordering* as that would allow for a common configuration is multiple Server are required but each for have a local container setting override.

```
FROM openjdk:8-jre
EXPOSE 8080
COPY target/ai.h2o.mojos.jar .
CMD java -Xms5g -xmx5g -Dai.h2o.mojos.runtime.license.filename=/License/license.sig -
DModelDirectory=/Models/ -jar ai.h2o.mojos.jar
```

The command “docker build -t h2orestserver .” will create the docker container image, then the following commands will execute it, change the IP and ports as needed for the environment.

```
kubectl run h2orestserver --replicas=2 --labels="run=load-balancer- h2orestserver " --image=
h2orestserver:v1  --port=8080

kubectl expose rs daimojorestserver --type="LoadBalancer" --name=" h2orestserver -service" --
external-ip=192.168.1.40

kubectl get pods --selector="run=load-balancer-daimojorestserver" --output=wide
kubectl get service daimojorestserver-service  --output=wide
```

Probes

Both Liveness and Readiness probes are supported the urls for each are:

State	URL
Readiness	/actuator/health/readiness
Liveness	/actuator/health/liveness

State Transition

If using the probes, then the eScorer will report its state in response to the probes during startup and shutdown.

Application State	Liveness	Response	Readiness	Response
Starting	BROKEN	{ "status": "BROKEN", "components": { "livenessstate": { "status": "BROKEN" } } }	REFUSING_TRAFFIC	{ "status": "REFUSING_TRAFFIC", "components": { "readiness": { "status": "REFUSING_TRAFFIC" } } }
Started	CORRECT	{ "status": "CORRECT", "components": { "livenessstate": { "status": "CORRECT" } } }	REFUSING_TRAFFIC	{ "status": "REFUSING_TRAFFIC", "components": { "readiness": { "status": "REFUSING_TRAFFIC" } } }
Ready	CORRECT	{ "status": "CORRECT", "components": { "livenessstate": { "status": "CORRECT" } } }	ACCEPTING_TRAFFIC	{ "status": "ACCEPTING_TRAFFIC", "components": { "readiness": { "status": "ACCEPTING_TRAFFIC" } } }

Shutdown	BROKEN	{
		"status":
		"BROKEN",
		"components":
		{
		"livenessstate":
		{
		"status":
		"BROKEN"
		}
		}
		}
			UNREADY	{
				"status": "UNREADY",
				"components": {
				"readiness": {
				"status":
				"UNREADY"
				}
				}
				}

AWS Sagemaker

The Rest Server can be hosted as a Sagemaker Inference, this means that both H2O-3 and Driverless AI models can be used for scoring with an AWS that can call Sagemaker.

The Rest Server can automatically generate the required artifacts for Sagemaker:

- Sagemaker docker container
- Required commands to deploy the container into the AWS ECR
- Test command to verify the integration is working

Integration Details

The integration allows models to be deployed using Sagemaker by embedded in the model, license, Rest Server and properties file in the container.

All the AWS deployment and management options are inherited from the Sagemaker platform.

Create a Model Repository

The deployment requires an AWS repository (ECR) this may need to be created using the AWS Console (web UI) if one does not exist or if a separation of other repositories is desired.

https://console.aws.amazon.com/ecr/repositories

Create Artifacts

The following steps will create the artifacts for deployment.

1. Create the container using auto generation

http://127.0.0.1:8080/autogen?name=riskmodel.mojo&notebook=sagemaker-container

2. Create the commands and example test data for Sagemaker

http://127.0.0.1:8080/autogen?name=riskmodel.mojo&notebook=sagemaker-commands

3. Optionally generate a Jupyter Notebook with an example of calling Sagemaker to invoke the model.

http://127.0.0.1:8080/autogen?name=riskmodel.mojo&notebook=sagemaker-jupyter

4. Optionally generate a RedShift SQL function that can invoke the model from AWS databases such as RedShift, Athena or Aurora.

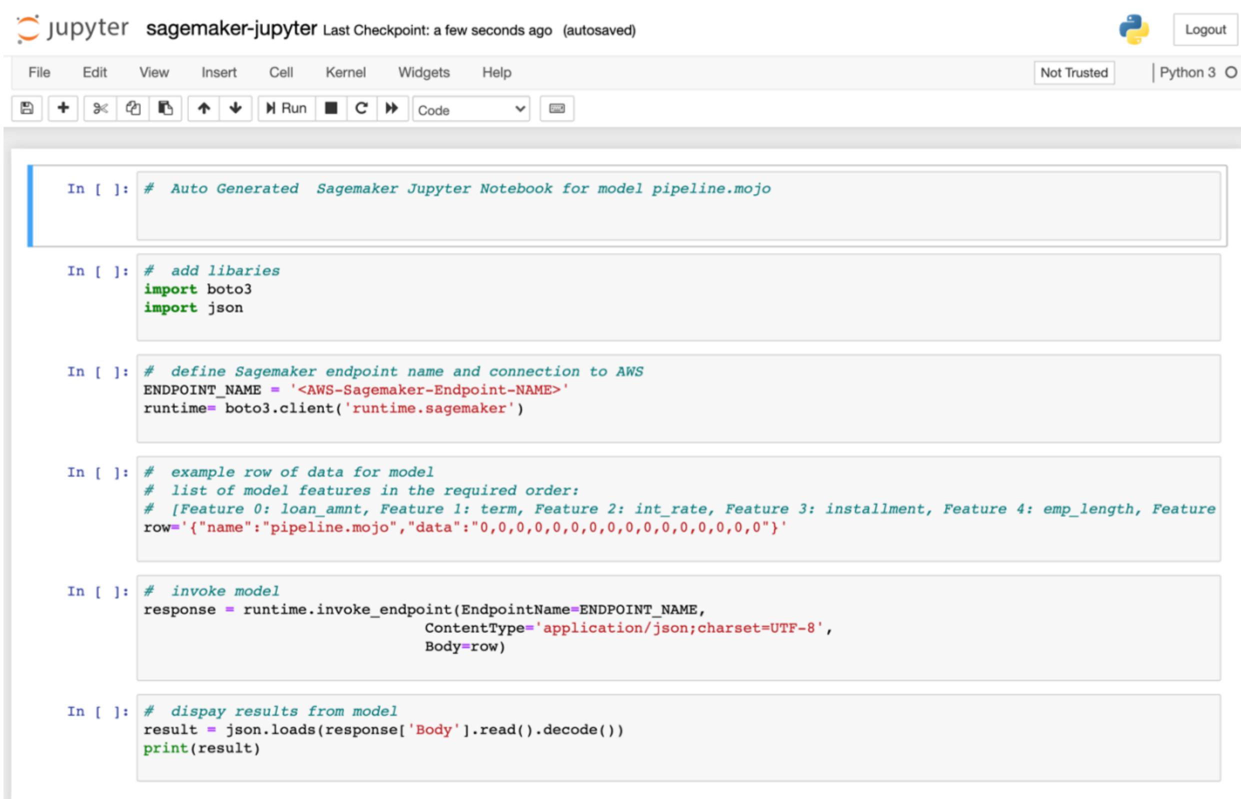
http://127.0.0.1:8080/autogen?name=riskmodel.mojo&notebook=Redshift-sql

Call a model

A Jupyter notebook can be generated for a specific model (see AutoGen settings) this will create an example of how to call the model hosted in Sagemaker with an example payload for the model.

Notebook

This is an example of the notebook that would be generated for the specific model using the command: curl "http://127.0.0.1:8080/autogen?name=pipeline.mojo&notebook=sagemaker-jupyter" -o sagemaker-jupyter.ipynb



Command line

The AWS CLI can also be used to call the model hosted in Sagemaker. This command can be generated using the command: curl "http://127.0.0.1:8080/autogen?name=pipeline.mojo&notebook=sagemaker-commands"

aws sagemaker-runtime invoke-endpoint --endpoint-name <Inference-Endpoint-Name> --body fileb://sagemaker-example.json --content-type=application/json SageMaker.out

Example data for testing using file sagemaker-example.json
{"name":"pipeline.mojo","verbose":"true","data":"0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0"}

SageMaker Endpoint

The model can be deployed as an Endpoint that is managed by the SageMaker Inference Runtime.

This enables models H2O-3 or Driverless AI models to be used for inferencing by passing either a JSON body or a CSV formatted row of data.

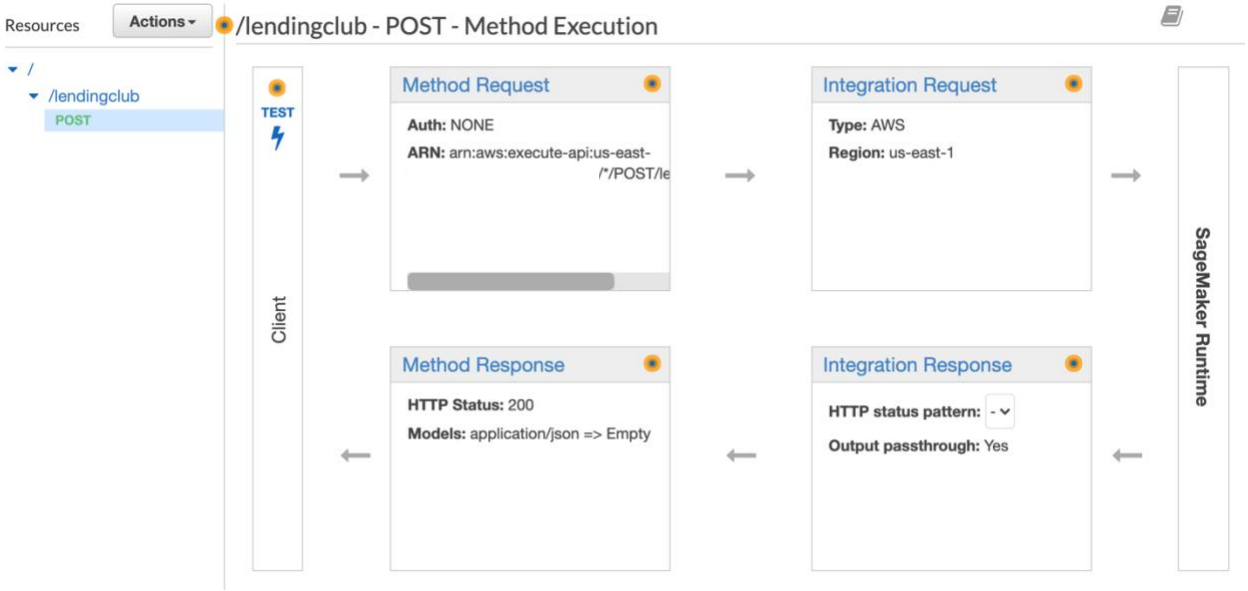
API Gateway / HTTP Endpoint

The Sagemaker instance can also have an externally reachable connection, if the API Gateway is configured, the setup steps on the AWS website for setting up a HTTP.

Overview of Configuration Steps

The AWS instructions documented in the link below in the Reference section should be used for the current steps, but the process consists of the following at a high level.

1. Creating an execution role for the REST API
   Before you build the REST API, you need to create an execution role that gives your API the necessary permission to invoke your Amazon SageMaker endpoint.
2. Building an API Gateway endpoint
   Select the REST API option
3. Only a POST method is required
4. No Mapping is required
5. The Autogen sagemaker-commands output will generate an example json payload for testing.



Reference

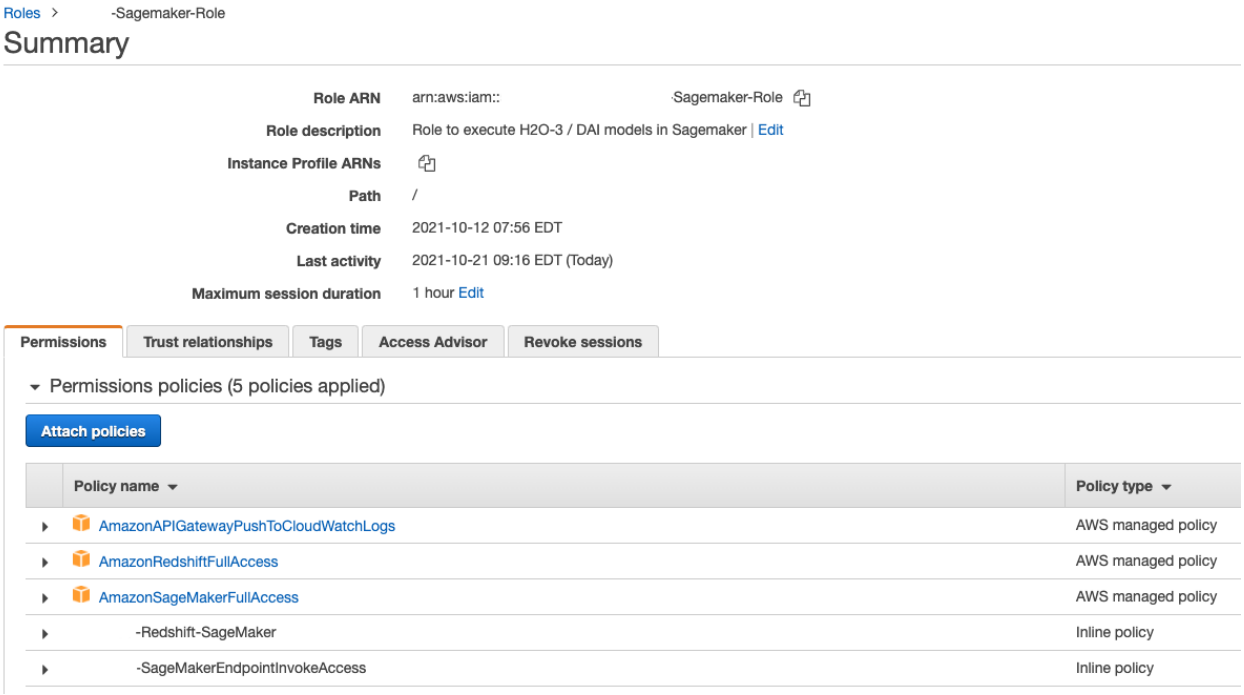
This link provides step by step instructions, please review these carefully to ensure the correct setup, IAM role and Sagemaker Invocation privileges are created.

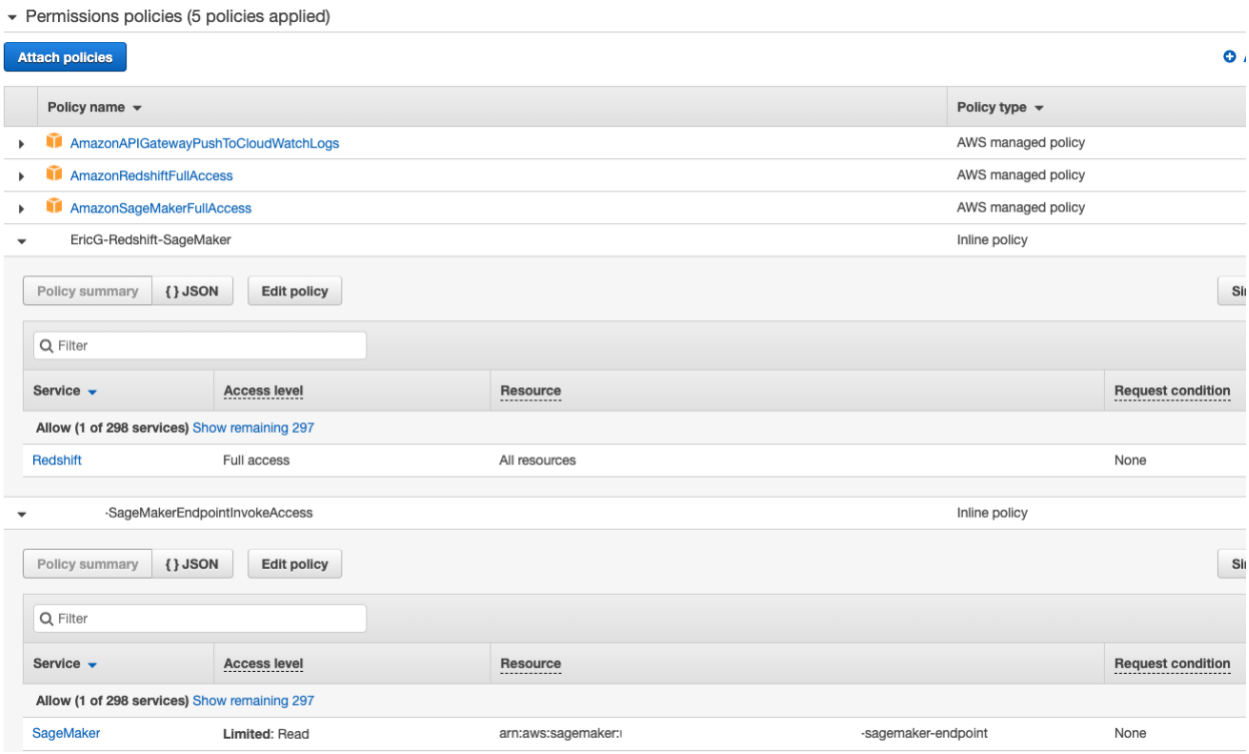
https://aws.amazon.com/blogs/machine-learning/creating-a-machine-learning-powered-rest-api-with-amazon-api-gateway-mapping-templates-and-amazon-sagemaker/

Security Roles

Each of the integration points will require specific access to the AWS services being used, for example if you are using the SageMaker Runtime Inference endpoint only, the RedShift access would not be required.

The following example is using full access settings, these should not be used in a production environment and should be set to the minimum required privileges.





Inline Policy

Depending on your security settings, you may need to allow SageMaker usage of resources.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "sagemaker:*",
      "Resource": "*"
    }
  ]
}
```

Trust Relationship

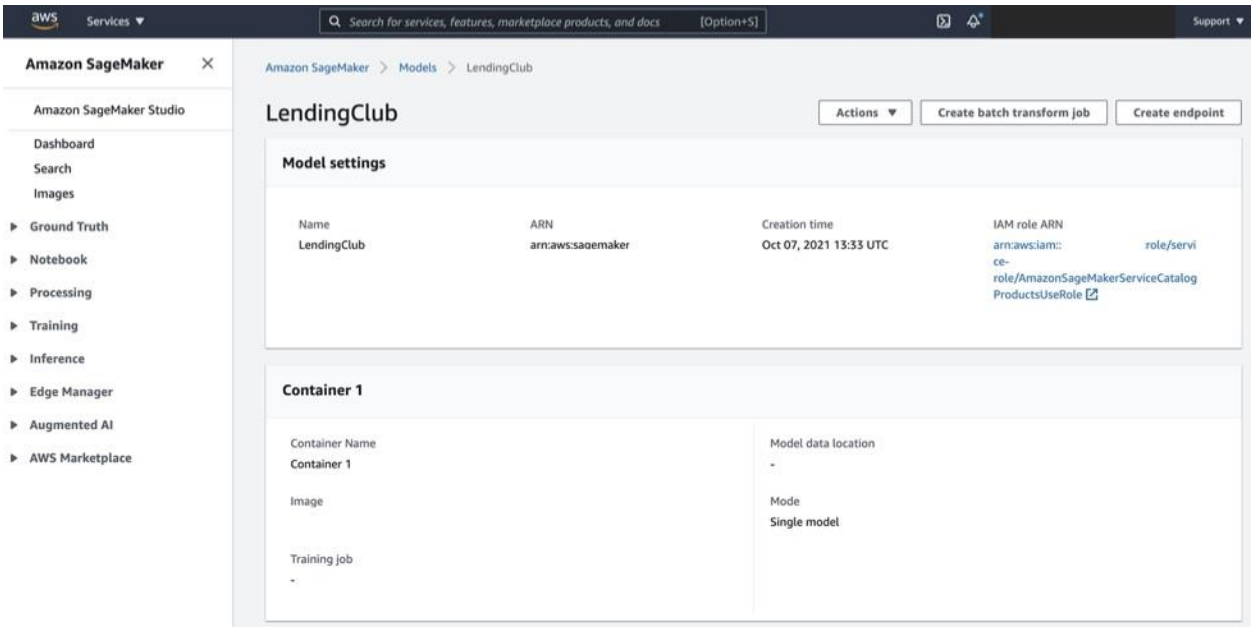
The *Create Model* step can require access to some services, if it is called from other AWS services.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": [
               "firehose.amazonaws.com",
               "cloudformation.amazonaws.com",
               "apigateway.amazonaws.com",
               "sagemaker.amazonaws.com",
               "redshift.amazonaws.com",
               "glue.amazonaws.com",
               "states.amazonaws.com",
               "events.amazonaws.com",
               "lambda.amazonaws.com",
               "codebuild.amazonaws.com",
               "codepipeline.amazonaws.com"
            ]
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

Define the Model

Using the AWS console define the model that will be deployed, this points to the docker image that was uploaded to the ECR.

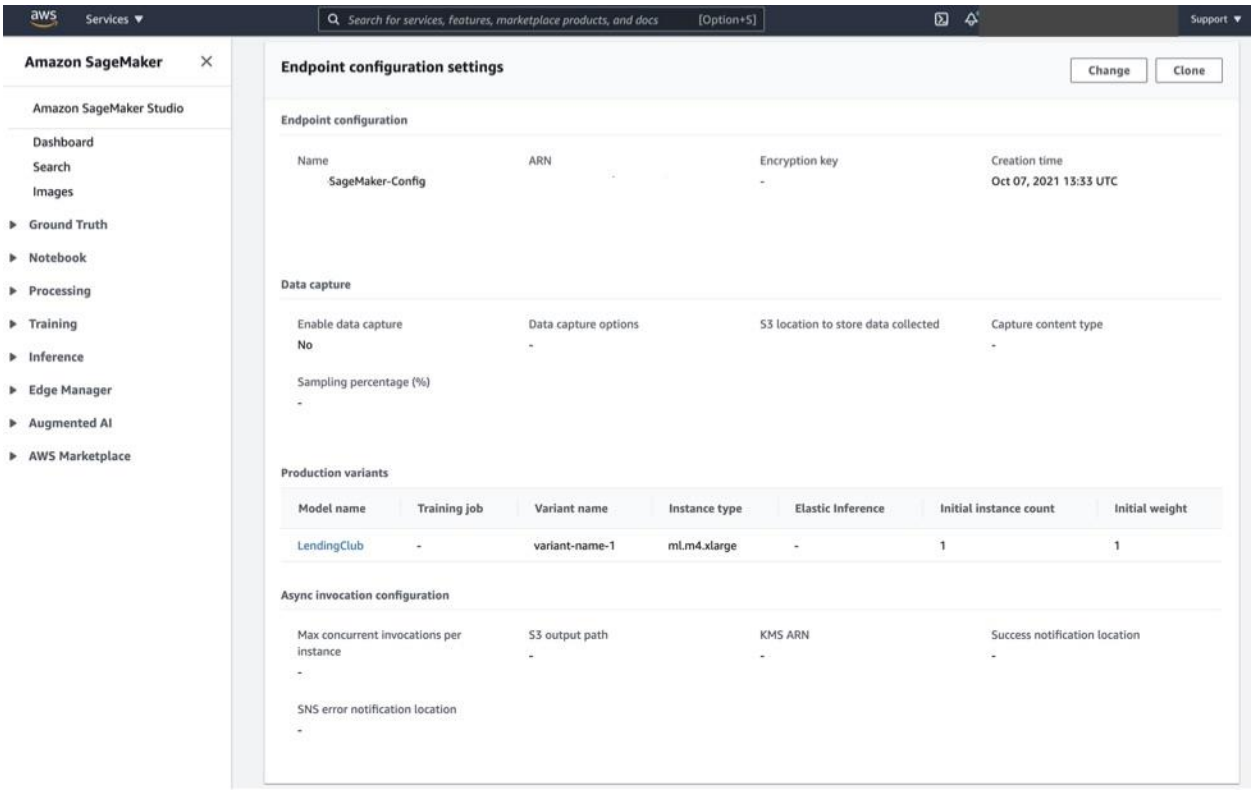
https://console.aws.amazon.com/sagemaker/home?region=us-east-1#/models



Configure the Endpoint

This step defines how the endpoint will scale with the number of instances and is any data logging is required.

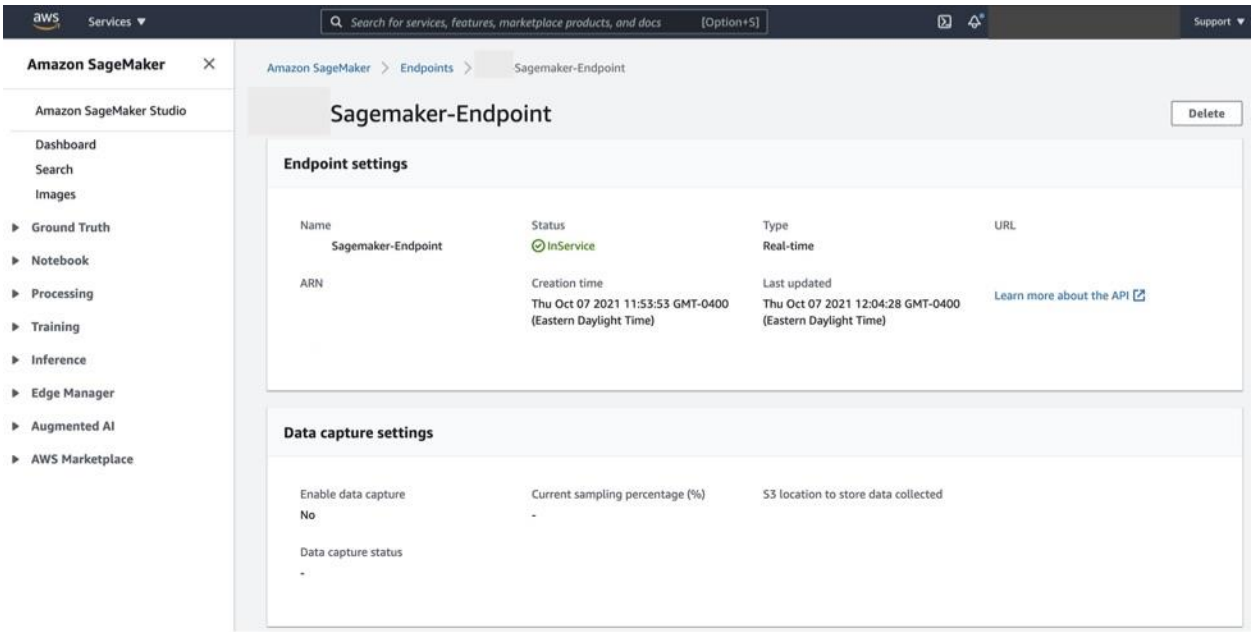
https://console.aws.amazon.com/sagemaker/home?/models&region=us-east-1#/endpointConfig



Deploy Endpoint

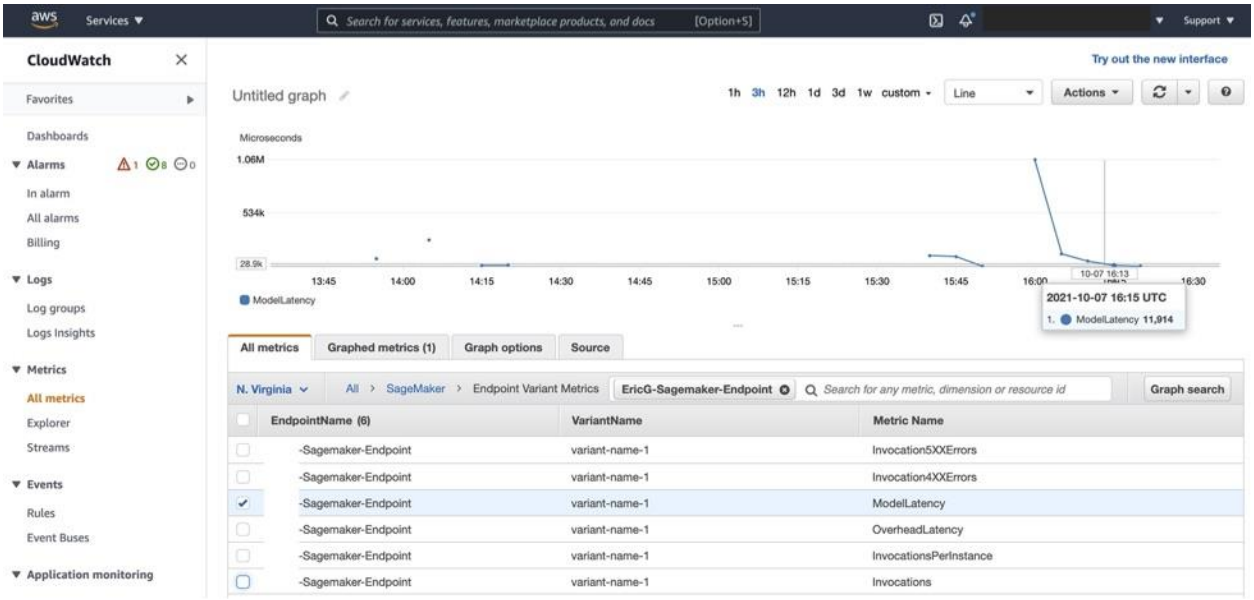
This step starts the instance. The deployment takes a few minutes (~10) to start the health check and logs will be available in Cloud Watch.

https://console.aws.amazon.com/sagemaker/home?/models&region=us-east-1#/endpoints



Cloud Watch

The integration supports the Sagemaker calls to /ping and /invocations as well as the logging of data to standard out is available in Cloud Watch.



SageMaker Batch Inference

Batch inferencing allows input data to be sent via a S3:// location for inferencing, the resulting predictions are written to a S3:// location.

Setup Steps

The Batch inference uses the same SageMaker configuration steps as the SageMaker Endpoint, follow those steps to define the ECR image and the Endpoint configuration, once the endpoint is deployed continue with the following steps.

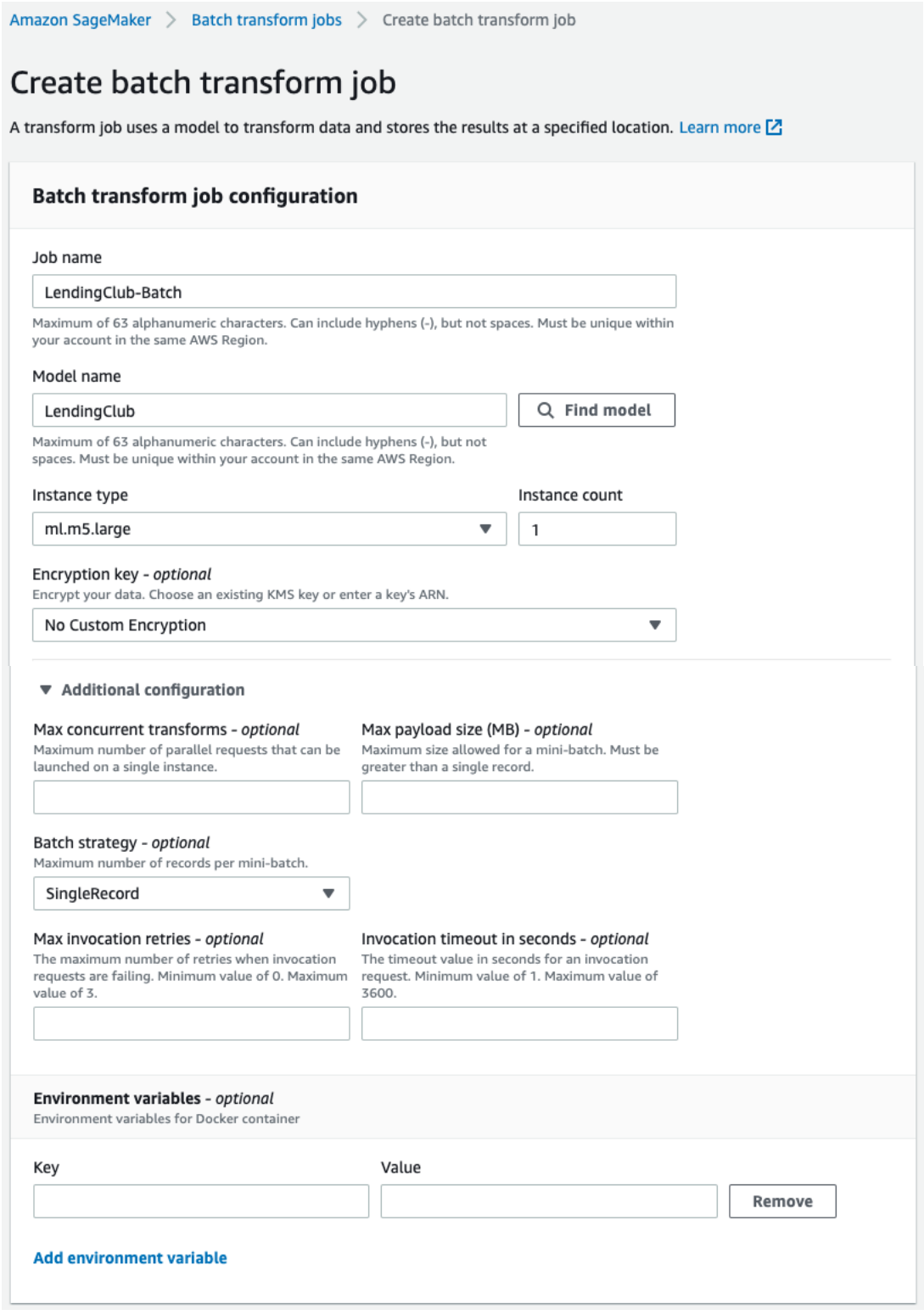
Define Batch Transformation Job

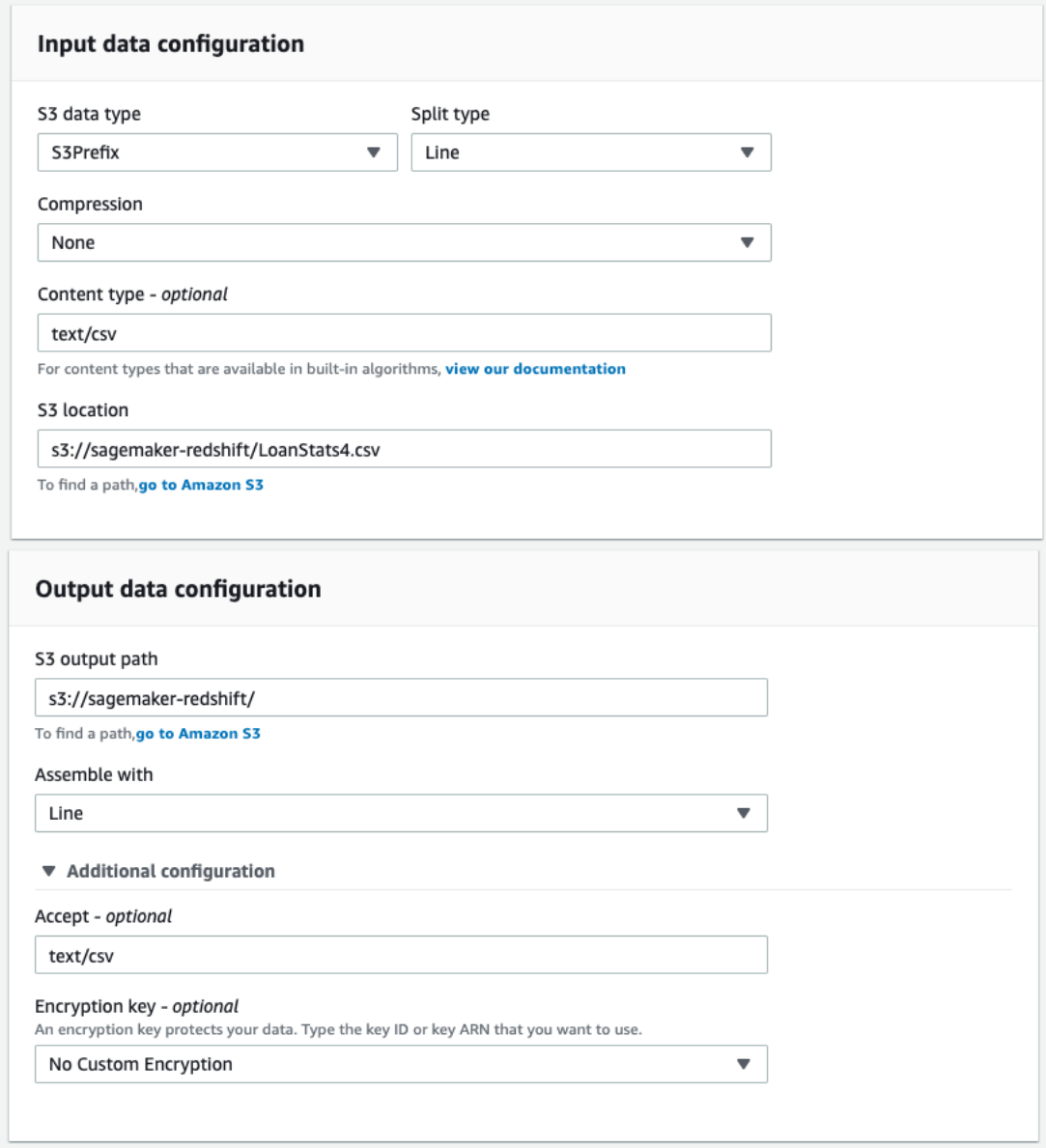
Using the SageMaker console select Inference and Batch Transform Job, this will enable the definition of the job's settings.

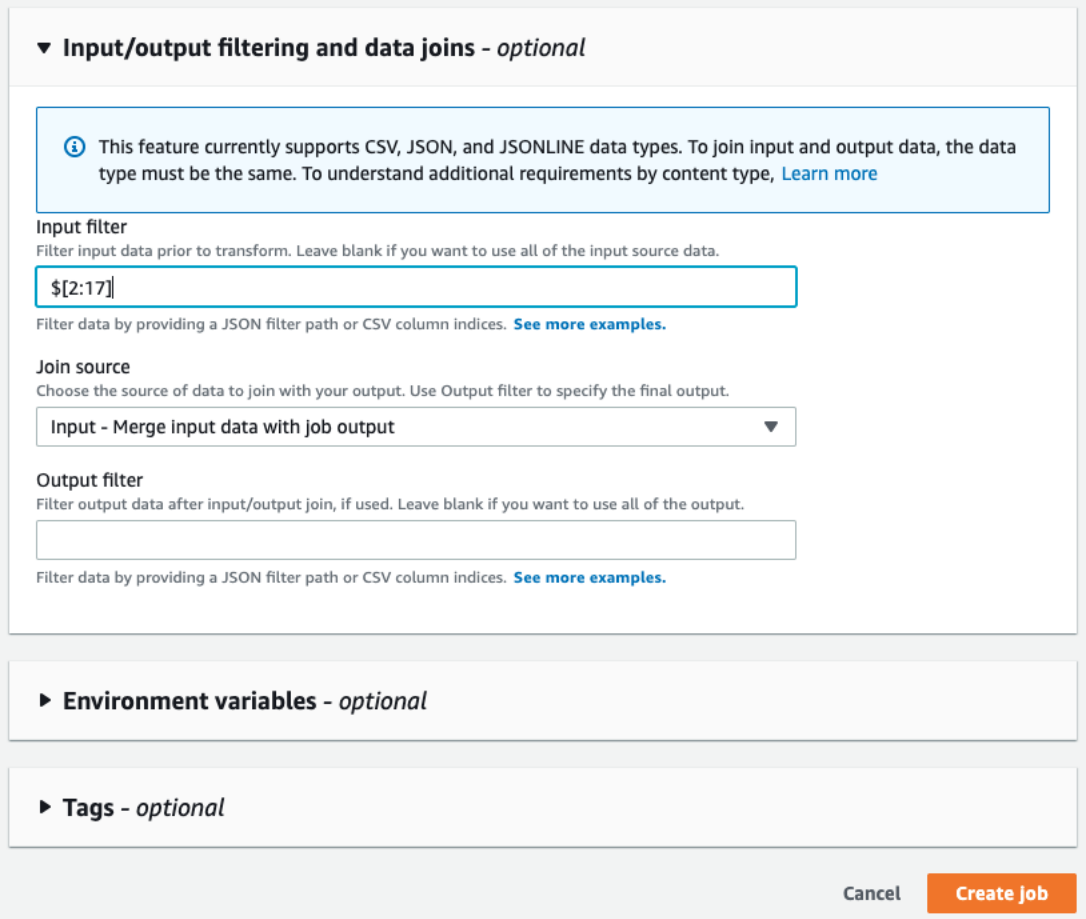
The following settings should be used:

- Job name:
- Model Name: select the model that was defined in the Define Model step
- Instance Type and Instance Count: a minimum of 2 vcpu and 8GB memory
- Batch Strategy: SingleRecord
- Split Type: Line
- Content type: text/csv
- S3 location (input data): Full s3 location of csv file
- S3 location (output data): s3 directory
- Accept: test/csv
- Input Filter: specify only the columns needed for the model, for example to use only columns 2 to 17 but drop columns 0, 1 and 18 the filter would be $[2:17]
- Join Source: Input – Merge input data with job output

Click Create Job to start, the progress of the job can be observed in the Cloud Watch link for this job.


Amazon SageMaker > Batch transform jobs > Create batch transform job
Create batch transform job
A transform job uses a model to transform data and stores the results at a specified location. Learn more
Batch transform job configuration
Job name
LendingClub-Batch
Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in the same AWS Region.
Model name
LendingClub
Find model
Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in the same AWS Region.
Instance type
ml.m5.large
Instance count
1
Encryption key - optional
Encrypt your data. Choose an existing KMS key or enter a key's ARN.
No Custom Encryption
Additional configuration
Max concurrent transforms - optional
Maximum number of parallel requests that can be launched on a single instance.
Max payload size (MB) - optional
Maximum size allowed for a mini-batch. Must be greater than a single record.
Batch strategy - optional
Maximum number of records per mini-batch.
SingleRecord
Max invocation retries - optional
The maximum number of retries when invocation requests are failing. Minimum value of 0. Maximum value of 3.
Invocation timeout in seconds - optional
The timeout value in seconds for an invocation request. Minimum value of 1. Maximum value of 3600.
Environment variables - optional
Environment variables for Docker container
Key
Value
Remove
Add environment variable


Input data configuration
S3 data type
S3Prefix
Split type
Line
Compression
None
Content type - optional
text/csv
For content types that are available in built-in algorithms, view our documentation
S3 location
s3://sagemaker-redshift/LoanStats4.csv
To find a path,go to Amazon S3
Output data configuration
S3 output path
s3://sagemaker-redshift/
To find a path,go to Amazon S3
Assemble with
Line
Additional configuration
Accept - optional
text/csv
Encryption key - optional
An encryption key protects your data. Type the key ID or key ARN that you want to use.
No Custom Encryption



File Format

Batch inferencing requires an input file in a S3:// location that is available to the job, the format of the file is a row, comma separated with a newline terminating each line and no header.

Runtime Configuration

When executing as a Sagemaker Batch Transformation Job the following environment variables can be used to control the type of output produced.

- H2OAI_BATCH_HEADER if set to true, will output a header line, that is the models target and optionally Shapley contribution header
- H2OAI_BATCH_SHAPLEY if set to true, will output the column names of the Shapley contributions

These are set in either the eScorer runtime as environment settings when starting the image or using the AWS BOTO library to run the Sagemaker runtime for real-time or batch inference.

RedShift-SQL

The SageMaker integration enables SQL calls to invoke models that are hosted in the SageMaker Inference Runtime.

Setup Steps

The Batch inference uses the same SageMaker configuration steps as the SageMaker Endpoint, follow those steps to define the ECR image and the Endpoint configuration, once the endpoint is deployed continue with the following steps.

SQL Definition

A SQL function definition must be created the describes the calling parameter types that are going to be sent from RedShift to SageMaker, the return type as well as the name of the SageMaker endpoint name and role to use when invoking the call.

Example Definition

```
CREATE MODEL riskmodel
FUNCTION h2oscores ( int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int, int)
RETURNS varchar
SAGEMAKER H2Oai-Sagemaker-Endpoint'
IAM_ROLE 'arn:aws:iam::nnnnnnnnnn:role/H2Oai-Sagemaker-Role';
```

Example Select / Insert

```
INSERT INTO RESULT (ID, bad_loan_0)
id, h2oscore(loan_amnt, term, int_rate, installment, emp_length, home_ownership,
annual_inc, verification_status, addr_state, dti, delinq_2yrs, inq_last_6mths, pub_rec,
revol_bal, revol_util, total_acc)
from lendingclub ;
```

Calling the AutoGen function with RedShift-SQL will generate the SQL for creating the model link to SagaMaker and return an example SQL statement for the specific model.

Microsoft Azure

The Rest Server can be deployed as a custom container within the Microsoft Azure environment, this allows both H2O-3 and Driverless AI mojo's to be used within the Azure eco system.

Many of the artifacts for deployment can be created using the Rest Servers AutoGen functionality, these include:

Artifact Name	Description
Azure_Commands	Azure CLI commands to deploy container
Azure_Container	Azure example Dockerfile for custom container
Azure_Serving_Deployment_yml	Deployment YAML for az commands
Azure_Serving_Endpoint_yml	Endpoint YAML for az commands
Azure_Start	Custom container entry point

Additionally, the following Microsoft Product specific artifacts are available:

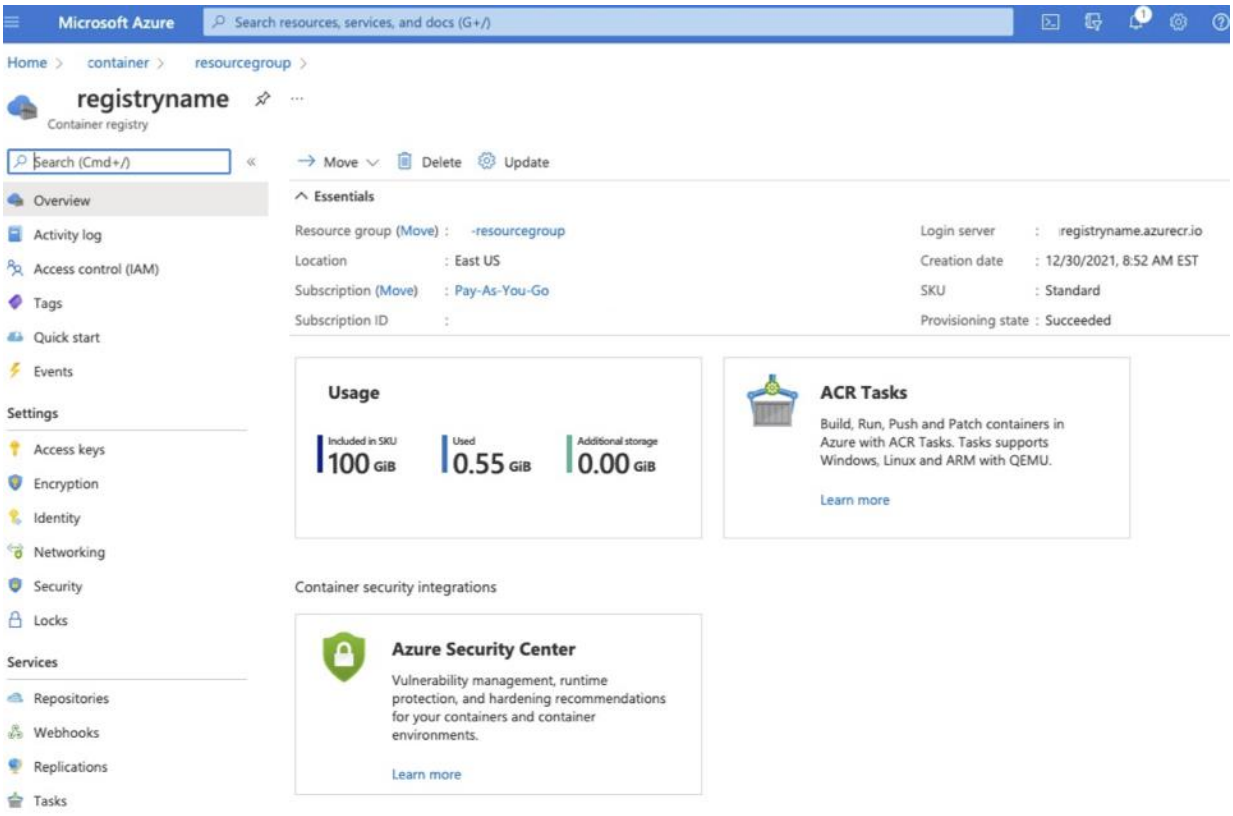
Artifact Name	Description
Excel_PowerQuery	Using ODBC to models in a database UDF
Excel_WebQuery	Using Excel web query syntax
Excel_WebService	For Windows environments using a Web Service
CSharp	C# example of calling the Rest API
PowerBI	Dashboard integration using Python
PowerBI_web	Dashboard content example

Creating Custom Container

Using the Autogen function the Container and the az commands can be created for the specific model.

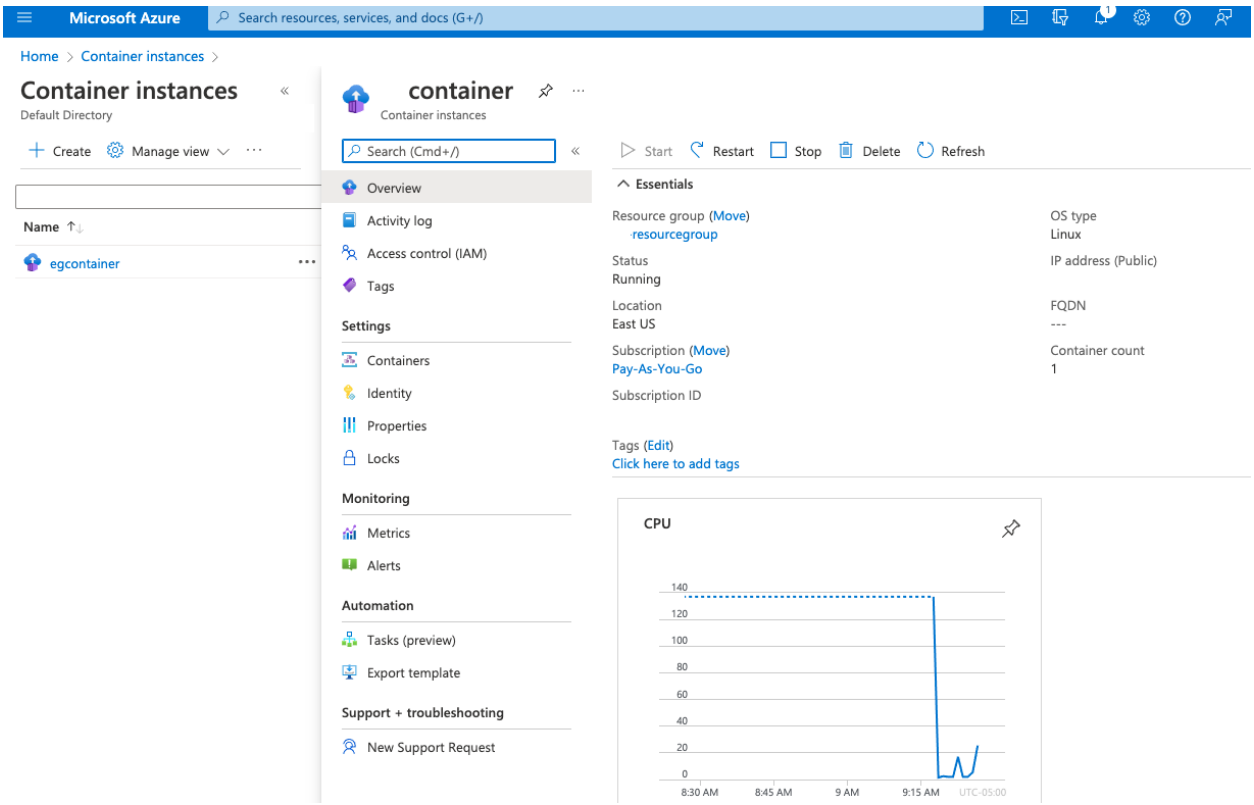
Saving Image to Azure Container Registry (ACR)

Please refer to the Azure ACR documentation for the latest steps in saving the image to ACR, once a repository has been created the image can be pushed for example:



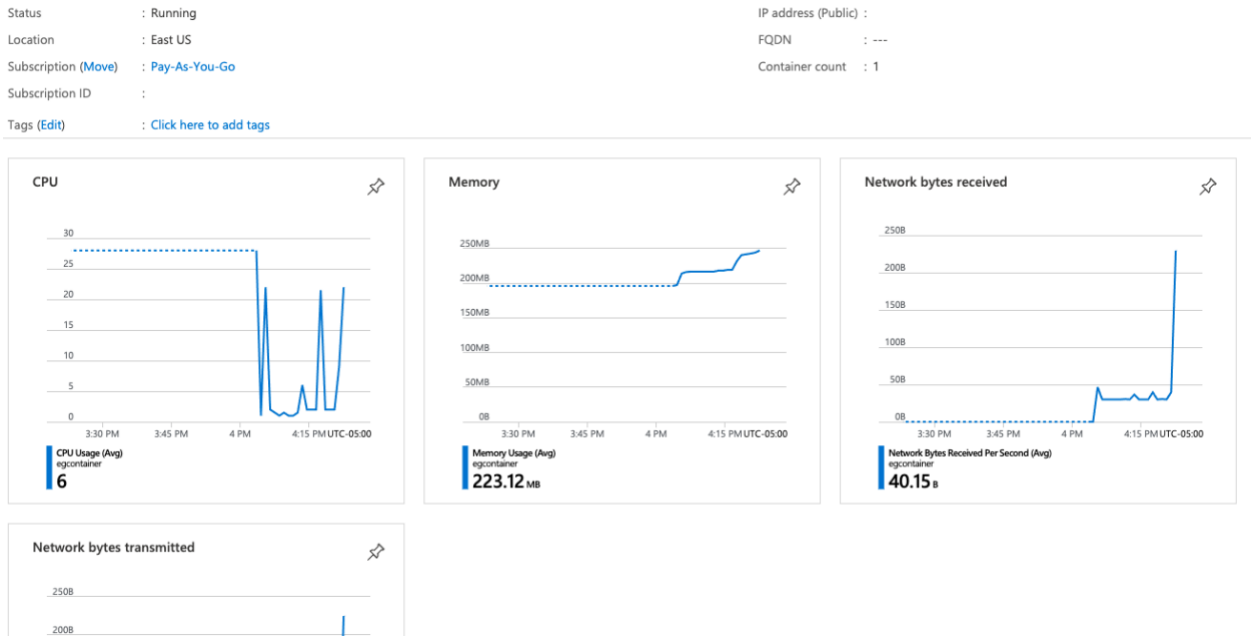
Deploying the Container

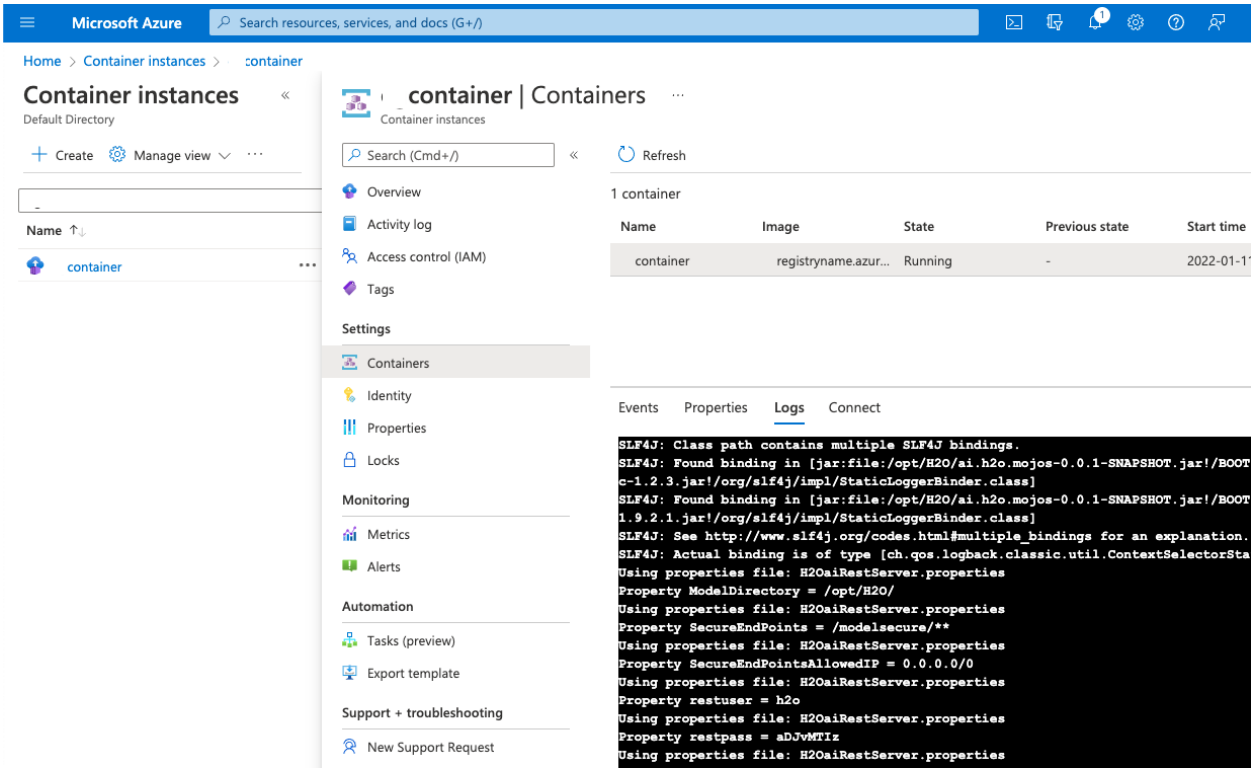
Once the container is in the registry it can be deployed using either the az commands or using the Azure UI.



Monitoring

When the container executes DevOps metrics (CPU, Memory, Network) will appear in the Azure UI. Additionally, the standard Rest Server logging output will appear in the Azure UI logs tab for the container.





Advanced Configuration Options

The Azure environment has several features that can be used with the custom container, here are a few options to consider.

- Use Azure Data Fabric to distribute the models and license artifact to multiple containers
- Use Curated containers and the private Python library support Azure has to deploy the models.

References

The following links may be helpful as a reference.

Container Set up: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-custom-container

CLI Reference: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-configure-cli

Azure UI:
https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-managed-online-endpoint-studio

Google Cloud Platform (GCP) App Engine

The following describes the sets to run the Rest Server on GCP.

Prerequisites

- Google Cloud SDK
- gsutil Tool
- A GCP project with App Engine enabled
- MOJO REST server JAR file

Setting Up

Deployment to App Engine is performed with a flexible custom runtime that lets you specify a custom Dockerfile for GCP to build. This lets you add any additional components necessary to your container and also makes porting out deployment to GKE easier. A flexible custom runtime for GCP App Engine requires the following:

1. An *app.yaml* file
2. A *Dockerfile*
3. A GCP Bucket for storing models (optional)

The following is an example of a minimal *app.yaml* file:

app.yaml

```
# Tells GCP we want to specify our own Dockerfile
runtime: custom
# Tells GCP We won't be using one of the standard environments
env: flex
# Which service we want to deploy to
service: default
# Optional bucket where models are stored
env_variables:
  MODELS_BUCKET: [BUCKET-NAME]
```

Note: If this is your first deployment to App Engine for your project, you must deploy to the default service first. If this is not your first deployment, you can specify another service.

The following is an example of a minimal *Dockerfile*:

Dockerfile

```
# You can specify any base image you want as long as it contains Java 8
FROM openjdk:8-jre
# The default port App Engine relays incoming request on
EXPOSE 8080
# Copy the MOJO REST server Jar file into our Image
COPY src-dir/ai.h2o.mojos-X-X.XX.jar .
# Optional step to copy MOJO file into image
COPY src-dir/pipeline.mojo .
# Required license file for using Driverless AI MOJOs
COPY src-dir/license.sig .
# Specify model bucket environment variable
ENV MODELS_BUCKET [BUCKET NAME]
# Start the REST server
CMD java -Xms5g -Xmx5g -Dai.h2o.mojos.runtime.license.filename=license.sig
-DModelDirectory=$MODELS_BUCKET -jar ai.h2o.mojos-X-X.XX.jar
```

Note: You can either copy MOJOs directly into the image or upload them to GCP storage. The above assumes the latter approach. Additionally, the above *Dockerfile* is just a template that you can modify to suit your specific needs.

Creating A GCP Bucket For Models

```
gsutil mb -p [PROJECT ID] gs://[BUCKET NAME]
```

After creating a models bucket for your project, you can then upload any models you wish to serve using the REST server to this bucket. You can upload models to the above bucket with:

Uploading Models To A GCP Bucket

```
gsutil cp [SOURCE FILE] gs://[BUCKET NAME]
```

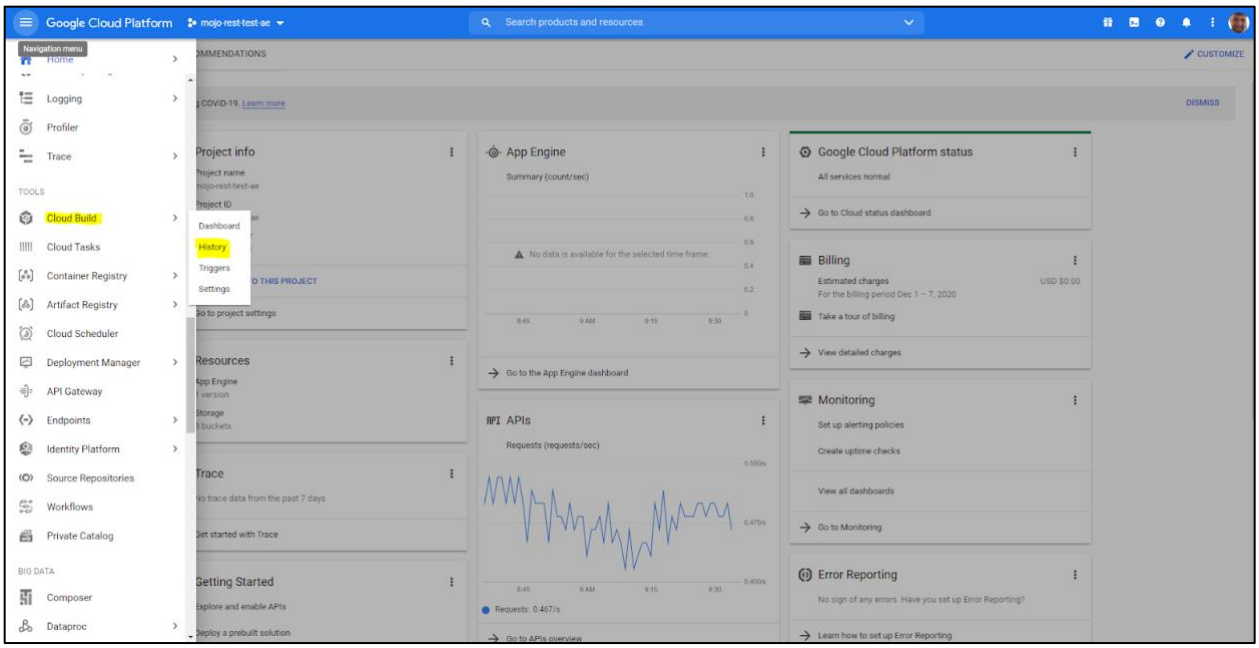
Deploying To GCP App Engine

To deploy your REST server to App Engine, navigate to the directory that contains your *app.yaml* and *Dockerfile* and run the following command:

Deploying Your App

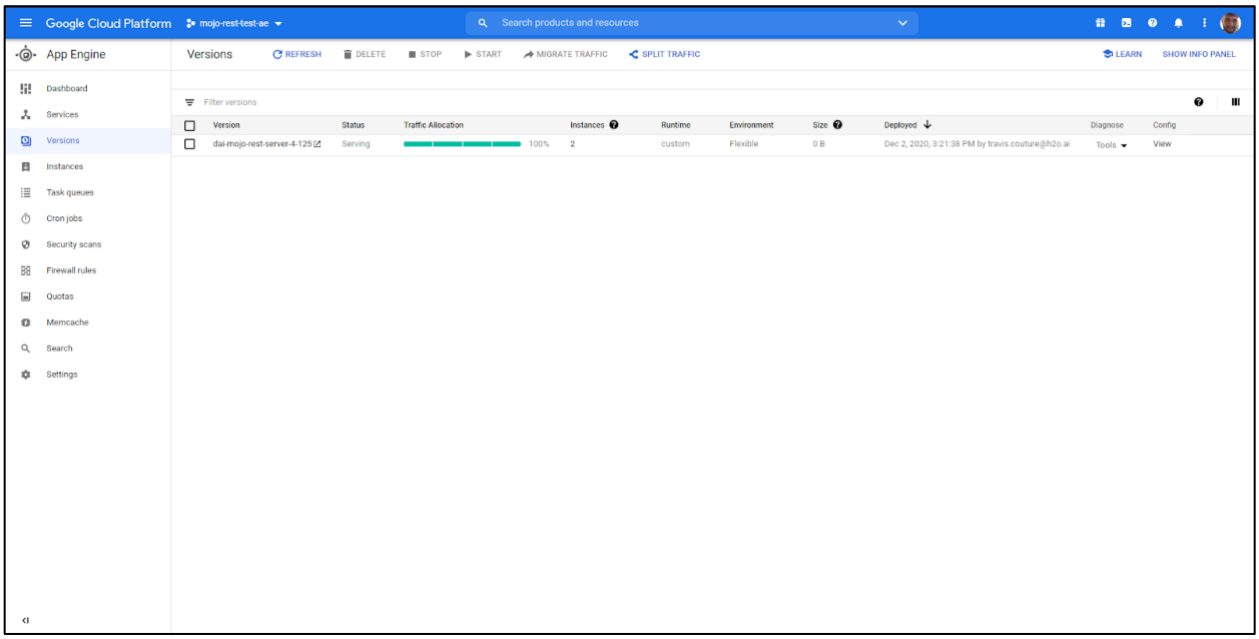
```
gcloud app deploy --version [APP VERSION] --project [PROJECT NAME]
```

Depending on the size of your image, it may take a few minutes or more to build everything for your app. You can monitor build progress by visiting the Cloud Build History page of your project:



Validating Deployment

To validate that the REST server is running without issue, first check that your service has some instances running. Navigate to the App Engine page of your project and select **Services**, then click on your service name. A page similar to the one in the following image should appear:



Assuming there are no immediate issues above, we are ready to test the endpoint of our REST server. You can visit the [MOJO REST Server documentation] for additional endpoint options but for now we can simply use:

Testing Model Availability

```
curl
"https://PROJECT_ID.REGION_ID.r.appspot.com/modelfeatures?name=pipeline.moj
o"
```

If the request is successful you should see something similar to the following depending on the features in your model:

```
{"result":["Feature 0: term","Feature 1: int_rate","Feature 2:
installment","Feature 3: grade","Feature 4: sub_grade","Feature 5:
emp_title","Feature 6: emp_length","Feature 7: home_ownership","Feature 8:
annual_inc","Feature 9: issue_d","Feature 10: purpose","Feature 11:
title","Feature 12: dti","Feature 13: earliest_cr_line","Feature 14:
inq_last_6mths","Feature 15: revol_bal","Feature 16: revol_util"]}%
```

You can also check general REST server health with the following:

Rest Server Health Check

```
curl "https://PROJECT_ID.REGION_ID.r.appspot.com/ping"
```

Resources

- App Engine Custom Runtime Quickstart
- Configuring Your App with app.yaml
- Building Custom Runtimes

Google Kubernetes Engine (GKE)

This describes the steps to deploy the Server using Google Kubernetes Engine

Prerequisites

- Google Cloud SDK
- A GCP project with Kubernetes Engine enabled
- kubectl installed on your machine of use
- MOJO REST server JAR file

Setup

A cluster is needed in order to deploy to GKE. You can use an already existing cluster or create one for your project with the following command:

Creating A GKE Cluster

```
gcloud container clusters create [CLUSTER NAME] --num-nodes [NUM NODES]
```

Note: The preceding command creates the cluster in your project's default region.

Next, run the following command to retrieve the credentials that are needed to access the target cluster:

Getting GKE Cluster Credentials

```
gcloud container clusters get-credentials [CLUSTER NAME]
```

The following commands set up *kubectl* to work with the targeted cluster in GKE and validate that *kubectl* is configured with the correct cluster:

Checking Cluster Connection

```
kubectl cluster-info
kubectl get nodes
```

Deploying To A GKE Cluster

To deploy to the GKE Cluster, you need to specify an image for running your containers. For this we need to start with a *Dockerfile.* The example below should get you started and the only requirement is that the base image must have a Java 8 runtime.

Dockerfile

```
# You can specify any base image you want as long as it contains Java 8
FROM openjdk:8-jre
# The default port App Engine relays incoming request on
EXPOSE 8080
# Copy the MOJO REST server Jar file into our Image
COPY src-dir/ai.h2o.mojos-X-X.XX.jar .
# Optional step to copy MOJO file into image
COPY src-dir/pipeline.mojo .
# Required license file for using Driverless AI MOJOs
COPY src-dir/license.sig .
# Start the REST server
CMD java -Xms5g -Xmx5g -Dai.h2o.mojos.runtime.license.filename=license.sig
-jar ai.h2o.mojos-X-X.XX.jar
```

Now we can use the above *Dockerfile* to create an image with GCP's cloud build. The resulting image will be stored in GCP's container registry and we will be able to provide it to our GKE cluster. Before you perform the following step ensure that the Cloud Build API is enabled for your project.

Building Our Image With Cloud Build

```
gcloud builds submit --tag gcr.io/[PROJECT ID]/mojo-rest-gke
```

Depending on the final size of your image, the above process could take a while to complete. You should be able to verify a successful build in your terminal or through the Cloud Build History page on the GCP console.
Now we need to define a Kubernetes Deployment and Service. We'll start with a *deployment.yaml*

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mojo-rest-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mojo-rest-app
  template:
    metadata:
      labels:
        app: mojo-rest-app
    spec:
      containers:
      - name: mojo-rest-server
        # Our image we built using GCP Cloud Build
        image: gcr.io/[PROJECT ID]/mojo-rest-gke
        ports:
        - containerPort: 8080
        env:
        - name: PORT
          value: "8080"
```

Then we use the following to deploy our Deployment to our GKE cluster:

Deploy Deployment To GKE

```
kubectl apply -f deployment.yaml
```

If the deployment was successful you should see something similar to the follow after running the command below.

Validate Deployment

```
kubectl get deployments
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
mojo-rest-deployment   1/1     1            1           43s

kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
mojo-rest-deployment-5b9df86fb6-jpvl8   1/1     Running   0          43s
```

Exposing Our Deployment

Now that we have successfully deployed our MOJO REST server Deployment to our GKE cluster, we need to expose robustly so that we can interact with it from outside the cluster without having to worry about individual Pods in our Deployment. To accomplish this, we will use a Service and more specifically, a Load Balancer. We define our Service below in the *service.yaml* file.

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mojo-rest-lb
spec:
  type: LoadBalancer
  selector:
    app: mojo-rest-app
  ports:
  - port: 80
    targetPort: 8080
```

Now we can deploy our service the same way as our Deployment:

Deploy Service To GKE

```
kubectl apply -f service.yaml
```

We can validate that our Service was successfully deployed with the command below. **Note:** It can take a moment for an External IP to be given to our Service.

Validate Service

```
kubectl get services
NAME           TYPE           CLUSTER-IP     EXTERNAL-
IP     PORT(S)        AGE
kubernetes     ClusterIP      10.3.240.1     <none>          443/TCP
12d
mojo-rest-
lb   LoadBalancer   10.3.247.112   35.238.49.112   80:31621/TCP   52s
```

Testing The MOJO REST Server

With our Service in place, we can test our MOJO REST server to validate functionality. You can upload a MOJO to the REST server with the following:

Model Upload

```
curl -u h2oadmin:h2o123 -F "file=@src-dir/pipeline.mojo" "[SERVICE EXTERNAL IP]/modelupload?name=pipeline.mojo&replace=true"
```

Note: There is a default file size upload limit which can be changed using:

File/Model Upload Size

```
--spring.servlet.multipart.max-file-size=[FILE SIZE]
--spring.servlet.multipart.max-request-size=[FILE SIZE]
```

Testing Model Availability

```
curl "[SERVICE EXTERNAL IP]:80/modelfeatures?name=pipeline.mojo"
```

If the request is successful you should see something similar to the following depending on the features in your model:

```
{"result":["Feature 0: term","Feature 1: int_rate","Feature 2: installment","Feature 3: grade","Feature 4: sub_grade","Feature 5: emp_title","Feature 6: emp_length","Feature 7: home_ownership","Feature 8: annual_inc","Feature 9: issue_d","Feature 10: purpose","Feature 11: title","Feature 12: dti","Feature 13: earliest_cr_line","Feature 14: inq_last_6mths","Feature 15: revol_bal","Feature 16: revol_util"]}%
```

You can also check general REST server health with the following:

Rest Server Health Check

```
curl "[SERVICE EXTERNAL IP]:80/ping"
```

Resources

- GKE Quickstart
- Kubernetes Deployments
- Kubernetes Services

HPE Kubernetes platform

The following are the configuration steps for using the eScorer on the HPE platform

Versions

HPE version: 5.4.1
Kubernetes version: 1.21.10

eScorer image (private): 524466471676.dkr.ecr.us-east-1.amazonaws.com/h2o-escorer-dev:latest or current standalone version

Deployment steps

1. Push eScorer image to a registry
2. Create PVC with default storage class - https://github.com/h2oai/hpe-escorer-deployment/blob/main/pvc.yml
   The cluster has a default storage class.
3. Create Kubernetes deployment - https://github.com/h2oai/hpe-escorer-deployment/blob/main/deployment.yml
4. Create service - https://github.com/h2oai/hpe-escorer-deployment/blob/main/service.yml
5. Apply all Kubernetes resources (I used **h2o-escorer** namespace to deploy resources)

Considerations when creating the deployment

Ports need to be exposed

- 8443 - API endpoint
- 8889
- 101101

Mounting license file

- Create secret for the license file content
  - kubectl create secret generic dai-license --from-file license.sig -n h2o-escorer
- Since Kubernetes can't directly mount a file to an existing directory mount it somewhere else and then copy it to `/opt/H2O/license.sig` using podStart command

Creating a volume mount for models

- **/mnt/model** path has a volume mount with PVC
- If the container is run as non-root user, permissions of volume mount should be changed via an **init-container** to match the UID and GUID of the container user. In HPE default MapR user id is 5000. Thus volume mounts initially owned by UID of 5000
- In this case, the container is run with UID of 600, Then init container changes the permission of model volume mount to 600 UID
- Following is the deployment file

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: h2o-escorer
 labels:
  app: h2o-escorer
spec:
 replicas: 1
 selector:
  matchLabels:
   app: h2o-escorer
 template:
  metadata:
   labels:
    app: h2o-escorer
  spec:
   containers:
    - name: h2o-escorer
      image: 524466471676.dkr.ecr.us-east-1.amazonaws.com/h2o-escorer-dev:latest
      env:
       - name: CUSTOMER_ID
         value: DEMO

      volumeMounts:
       - mountPath: /license/
         name: license
       - mountPath: /mnt/models
         name: models
         readOnly: false
      ports:
       - containerPort: 10101
       - containerPort: 8889
       - containerPort: 8443
      securityContext:
       runAsNonRoot: true
       runAsUser: 600
      lifecycle:
       postStart:
        exec:
         command:
          - /bin/sh
          - -c
          - cp /license/license.sig /opt/H2O/license.sig
   volumes:
```

```
  - name: license
    secret:
      secretName: dai-license
      items:
        - key: license.sig
          path: license.sig
  - name: models
    persistentVolumeClaim:
      claimName: models-pvc
initContainers:
  - name: change-mount-permissions
    image: busybox:1.28
    command: ['chown','-R','600:600', '/mnt/models' ]
    volumeMounts:
      - name: models
        mountPath: /mnt/models
```

Testing deployment locally

- Upload mojo model to **/mnt/models** folder using **kubectl cp**
- kubectl cp riskmodel.mojo <pod-name>:/mnt/models/
- Port Forward or Proxy traffic of 8443 port of the pod or service
- Call the API:

```
curl -k
"https://127.0.0.1:34567/model?name=riskmodel.mojo&row=1000,%2036%20months,17
,315.94,3,38000,VERIFIED%20-%20income,NY,19.93,0,0,74.7,1&verbose=true"
```

Database Scoring

The scorer can receive a properties file that describes a database connection, select, and output table and use a model to score.

This uses the Database Scorer, but it can be invoked from the eScorer.

Creating the properties file

java -Dmodelname=pipeline.mojo -Dcreateproperties=true -cp ai.h2o.mojos.jar ai.h2o.mojos.db.DAIMojoRunner_DB

The resulting properties file should be edited and the SQLConnection, SQLKey, SQLSelect and SQLWriteBatch parameters should be reviewed.

Verify the EnvVariables is set to true

An example properties file can also be create using the eScorer Wave application, look for the resource “ManagedCloud_Database_Scorer_properties” in the Autogen tab.

Starting scoring

The properties and mojo files are send using a Keycloak enabled secure connection.

curl -H "Authorization: Bearer $KEYCLOAK_TOKEN" --insecure
-F "file=@/DAIMojoRunner_DB.properties"
-F "file=@/pipeline.mojo"
"https://host-name:port/scoredatabase”

Output

The predictions are written to the Database defined in the properties file and a summary is returned to the curl command, for example

Total selected rows 39029 Total Read time (ms) 9569
Thread-1 Rows Read 39029 Scored 39029 Error 0 Queue Empty true

Monitoring

In addition to the Output summary the health status is available. Sending a request to **Error! Hyperlink reference not valid.** will return the live status of the number of rows selected, processed and errors.