TECHNICAL WHITEPAPER

Beginning Morpheus Custom Report Development

Last Updated: July 2021

INTRODUCTION

The Morpheus plugin architecture is a library which allows users to extend functionality in several categories, including new Cloud providers, Task types, UI views, custom reports, and more. In this guide, we will take a look at developing a custom report and adding it to Morpheus UI as a report selection for users to consume. Complete developer documentation including the full API documentation and links to Github repositories containing complete code examples are available in the Developer Portal.

Custom plugin development requires programming experience, but this guide is designed to break down the required steps into a digestible process that users can quickly run with. Morpheus plugins are written in Java or Groovy, our example here will be written in Groovy. Support for additional languages is planned but not yet available at the time of this writing. If you're not an experienced Java or Groovy developer, it may help to clone an example code repository which we link to in our developer portal. An additional example, which this guide is based on, is here. You can read the example code and tweak it to suit your needs using the guidance in this document.

Before you begin, ensure you have the following installed in your development environment:

- Gradle 6.5 or later
- Java 8 or 11

In this example, I'll create a custom report that shows the total number of Cypher items in Morpheus with a breakdown of the total number by type. Below that will be a table which shows each Cypher item individually and sorted alphabetically along with their create date and the date they were last accessed.

IDENTIFY TARGET DATA

To gather data for a custom report, you'll directly query the Morpheus database. To identify the relevant tables and columns, you can connect to the appliance node and use the MySQL client or remotely use a GUI tool (such as MySQL Workbench) to browse for the data you need. Using a GUI tool can be helpful if you aren't familiar with the Morpheus database layout since we can quickly view any table and toggle sorting to identify the targeted columns.

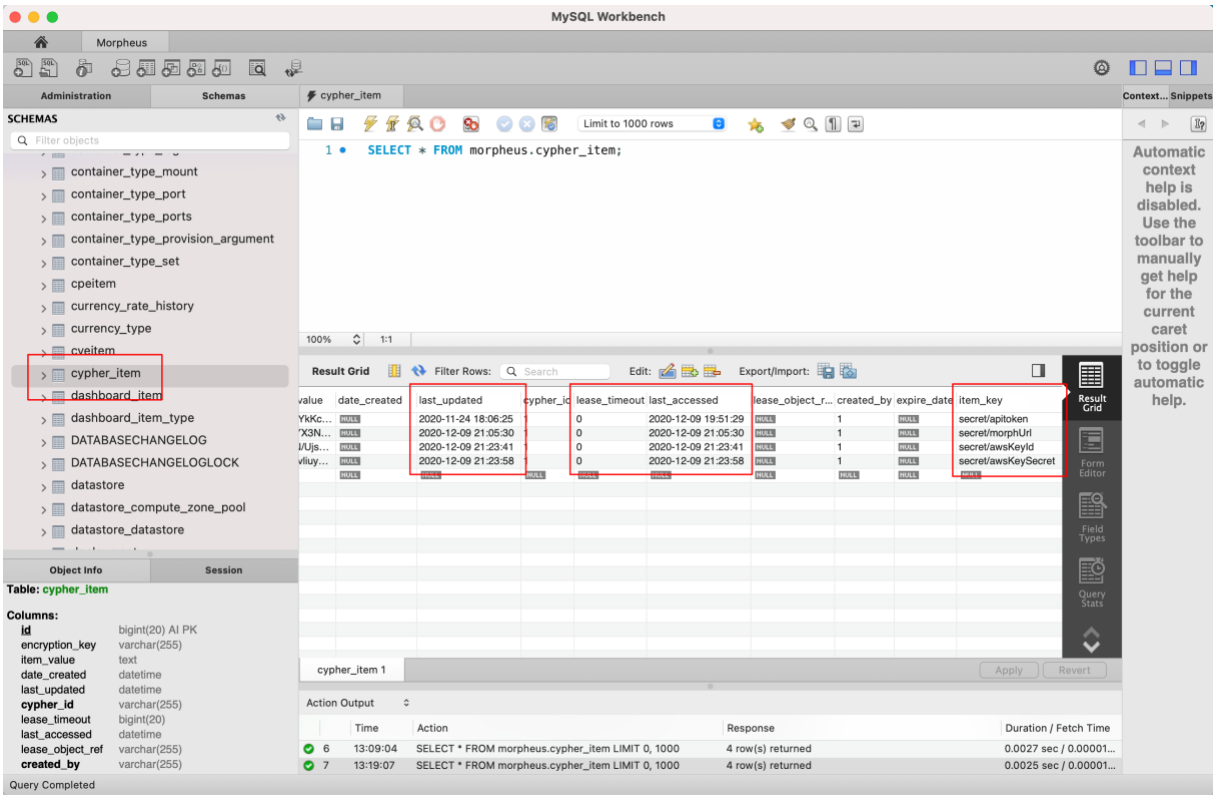
But first we'll need access credentials for the Morpheus database. We can get these from a secrets file which is created during Morpheus installation. Start an SSH session with your appliance node and `cat` the `morpheus-secrets.json` file:

```
cat /etc/morpheus/morpheus-secrets.json
```

You should receive an output like the one shown below. In the place of the redacted values, you will see the actual secret values.

```
{
  "mysql": {
    "root_password": "<REDACTED>",
    "morpheus_password": "<REDACTED>",
    "ops_password": "<REDACTED>"
  },
  "rabbitmq": {
    "morpheus_password": "<REDACTED>",
    "queue_user_password": "<REDACTED>",
    "cookie": "<REDACTED>"
  },
  "ui": {
    "ajp_secret": "<REDACTED>"
  }
}
```

Copy the `morpheus-password` value as you'll need it to create a new connection to the Morpheus database in MySQL Workbench. Open MySQL Workbench and configure a new connection. Once the connection is created, open the list of all database tables in the left-hand column of the application. In this case, I'm interested in the `cypher_item` table. I'm targeting the Cypher Item objects and planning to surface the `item_key`, `last_updated`, `lease_timeout`, and `last_accessed` values.



We'll structure our SQL query to target these selected columns and sort the table objects in the way we wish to display them in our report. Later on, this will become part of the Groovy code that makes up the custom report plugin.

```
SELECT item key,last updated,last accessed,lease timeout from cypher item order by item key asc;
```

DEVELOPING THE PLUGIN

To begin, create a new directory to house the project. You'll ultimately end up with a file structure typical of Java or Groovy projects, looking something like this:

```
./
.gitignore
build.gradle
src/main/groovy/
src/main/resources/renderer/hbs/
src/test/groovy/
src/assets/images/
src/assets/javascript/
src/assets/stylesheets/
```

Configure the `.gitignore` file to ignore the `build/` directory which will appear after performing the first build. Project packages live within `src/main/groovy` and contain source files ending in `.groovy`. View resources are stored in the `src/main/resources` subfolder and vary depending on the view renderer of choice. Static assets, like icons or custom javascript, live within the `src/assets` folder. Consult the table below for key files, their purpose, and their locations. Example code and further discussion of relevant files is included in the following sections.

File Name	Description	File Path
build.gradle	The Gradle build file	build.gradle
gradle.properties	The properties file for the Gradle build tool	gradle.properties
CustomReportProvider.groovy	Most of the custom code to fetch report data, determine how the data is aggregated, and categorize the report is written here	src/main/groovy/com/morpheusdata/reports/CustomReportProvider.groovy
ReportsPlugin.groovy	Create a new report plugin class which extends the plugin class here to register, name and describe the new plugin	src/main/groovy/com/morpheusdata/reports/ReportsPlugin.groovy
cypherReport.hbs	Handles creation of the new UI view which is displayed when the report is viewed	src/mainresources/renderer/hbs/cypherReport.hbs

CREATING THE BUILD.GRADLE FILE

Gradle is the build tool used to compile Morpheus plugins so `build.gradle` is required. An example build file is given below but some useful values to call out are as follows:

- **Group:** The package group in Java, typically your reverse DNS name
- **Version:** The version number for your plugin. This will be displayed in the Plugins section of Morpheus UI for reference when later versions of your plugin are developed
- **Plugin-class:** This will vary based on the plugin type being developed but for a custom report, use `com.morpheusdata.reports.ReportsPlugin`

```
plugins {
    id "com.bertramlabs.asset-pipeline" version "3.3.2"
    id "com.github.johnrengelman.plugin-shadow" version "2.0.3"
}

apply plugin: 'java'
apply plugin: 'groovy'
apply plugin: 'maven-publish'

group = ${'com.example'}
version = ${'1.2.2'}

sourceCompatibility = '1.8'
targetCompatibility = '1.8'

ext.isReleaseVersion = !version.endsWith("SNAPSHOT")

repositories {
    mavenCentral()
}

dependencies {
    compileOnly 'com.morpheusdata:morpheus-plugin-api:0.8.0'
    compileOnly 'org.codehaus.groovy:groovy-all:2.5.6'
    compileOnly 'io.reactivex.rxjava2:rxjava:2.2.0'
    compileOnly "org.slf4j:slf4j-api:1.7.26"
    compileOnly "org.slf4j:slf4j-parent:1.7.26"
}

jar {
    manifest {
        attributes(
            'Plugin-Class': 'com.morpheusdata.reports.ReportsPlugin', //Reference to Plugin class
            'Plugin-Version': archiveVersion.get() // Get version defined in gradle
        )
    }
}

tasks.assemble.dependsOn tasks.shadowJar
```

CREATING THE PLUGIN CLASS

Next, create a plugin class which handles registration of the new report, sets a name and description, and targets the appropriate report provider class which we'll go over in the next section.

```
package com.morpheusdata.reports

import com.morpheusdata.core.Plugin

class ReportsPlugin extends Plugin {

  @Override
  void initialize() {
    CustomReportProvider customReportProvider = new
      CustomReportProvider(this, morpheus)
    this.pluginProviders.put(customReportProvider.code, customReportProvider)
    this.setName("Custom Cypher Report")
    this.setDescription("A custom report plugin for cypher items")
  }

  @Override
  void onDestroy() {
  }
}
```

CREATING THE REPORT PROVIDER CLASS

The report provider class contains the code which will fetch and compile the targeted data so it can be rendered in the report view. An example report provider is reproduced below with comments to increase readability of the code.

```
package com.morpheusdata.reports

import com.morpheusdata.core.AbstractReportProvider
import com.morpheusdata.core.MorpheusContext
import com.morpheusdata.core.Plugin
import com.morpheusdata.model.OptionType
import com.morpheusdata.model.ReportResult
import com.morpheusdata.model.ReportType
import com.morpheusdata.model.ReportResultRow
import com.morpheusdata.model.ContentSecurityPolicy
import com.morpheusdata.views.HTMLResponse
import com.morpheusdata.views.ViewModel
import com.morpheusdata.response.ServiceResponse
import groovy.sql.GroovyRowResult
import groovy.sql.Sql
import groovy.util.logging.Slf4j
import io.reactivex.Observable;
import java.util.Date

import java.sql.Connection

@Slf4j
class CustomReportProvider extends AbstractReportProvider {
  Plugin plugin
  MorpheusContext morpheusContext

  CustomReportProvider(Plugin plugin, MorpheusContext context) {
    this.plugin = plugin
    this.morpheusContext = context
  }

  @Override
  MorpheusContext getMorpheus() {
    morpheusContext
  }

  @Override
  Plugin getPlugin() {
    plugin
  }

  // Define the Morpheus code associated with the plugin
  @Override
  String getCode() {
    'custom-report-cypher'
  }

  // Define the name of the report displayed on the reports page
  @Override
  String getName() {
    'Cypher Summary'
  }

   ServiceResponse validateOptions(Map opts) {
     return ServiceResponse.success()
   }

  @Override
  HTMLResponse renderTemplate(ReportResult reportResult, Map<String, List<ReportResultRow>>
reportRowsBySection) {
    ViewModel<String> model = new ViewModel<String>()
    model.object = reportRowsBySection
    getRenderer().renderTemplate("hbs/instanceReport", model)
  }
```

```
  void process(ReportResult reportResult) {
    // Update the status of the report (generating) -
https://developer.morpheusdata.com/api/com/morpheusdata/model/ReportResult.Status.html

morpheus.report.updateReportResultStatus(reportResult,ReportResult.Status.generating).blocking
Get();
    Long displayOrder = 0
    List<GroovyRowResult> results = []
    Connection dbConnection
    Long passwordResults = 0
    Long tfvarsResults = 0
    Long secretResults = 0
    Long uuidResults = 0
    Long keyResults = 0
    Long randomResults = 0
    Long totalItems = 0

    try {
      // Create a read-only database connection
      dbConnection = morpheus.report.getReadOnlyDatabaseConnection().blockingGet()
      // Evaluate if a search filter or phrase has been defined
        results = new Sql(dbConnection).rows("SELECT
item_key,last_updated,last_accessed,lease_timeout from cypher_item order by item_key asc;")
      // Close the database connection
    } finally {
      morpheus.report.releaseDatabaseConnection(dbConnection)
    }
    log.info("Results: ${results}")
    Observable<GroovyRowResult> observable = Observable.fromIterable(results) as
Observable<GroovyRowResult>
    observable.map{ resultRow ->
      log.info("Mapping resultRow ${resultRow}")
      Map<String,Object> data = [key: resultRow.item_key, last_updated:
resultRow.last_updated.toString(), last_accessed: resultRow.last_accessed.toString(),
lease_timeout: resultRow.lease_timeout ]
      ReportResultRow resultRowRecord = new ReportResultRow(section:
ReportResultRow.SECTION_MAIN, displayOrder: displayOrder++, dataMap: data)
      log.info("resultRowRecord: ${resultRowRecord.dump()}")
      totalItems++
      // Evaluate if the cypher item starts with password
      if (resultRow.item_key.startsWith('password')) {
        passwordResults++
      }
      // Evaluate if the cypher item starts with tfvars
      if (resultRow.item_key.startsWith('tfvars')) {
        tfvarsResults++
      }
      // Evaluate if the cypher item starts with secret
      if (resultRow.item_key.startsWith('secret')) {
        secretResults++
      }
      // Evaluate if the cypher item starts with uuid
      if (resultRow.item_key.startsWith('uuid')) {
        uuidResults++
      }
      // Evaluate if the cypher item starts with key
      if (resultRow.item_key.startsWith('key')) {
        keyResults++
      }
      // Evaluate if the cypher item starts with random
      if (resultRow.item_key.startsWith('random')) {
        randomResults++
      }
      return resultRowRecord
    }.buffer(50).doOnComplete {

morpheus.report.updateReportResultStatus(reportResult,ReportResult.Status.ready).blockingGet()
;
    }.doOnError { Throwable t ->
```

```
morpheus.report.updateReportResultStatus(reportResult,ReportResult.Status.failed).blockingGet(
);
    }.subscribe {resultRows ->
      morpheus.report.appendResultRows(reportResult,resultRows).blockingGet()
    }
    Map<String,Object> data = [total_items: totalItems, password_items: passwordResults,
tfvars_items: tfvarsResults, secret_items: secretResults, uuid_items: uuidResults, key_items:
keyResults, random_items: randomResults]
    ReportResultRow resultRowRecord = new ReportResultRow(section:
ReportResultRow.SECTION_HEADER, displayOrder: displayOrder++, dataMap: data)
        morpheus.report.appendResultRows(reportResult,[resultRowRecord]).blockingGet()
  }

  //
https://developer.morpheusdata.com/api/com/morpheusdata/core/ReportProvider.html#method.summar
y
  // The description associated with the custom report
   @Override
   String getDescription() {
     return "View an inventory of Cypher items"
   }

   // The category of the custom report
   @Override
   String getCategory() {
     return 'inventory'
   }

   @Override
   Boolean getOwnerOnly() {
     return false
   }

   @Override
   Boolean getMasterOnly() {
     return true
   }

   @Override
   Boolean getSupportsAllZoneTypes() {
     return true
   }
  }
```

CREATE THE CUSTOM REPORT VIEW

By default, custom plugin views are handled by a Handlebars template provider to populate HTML sections with your own content. Though it can be overridden, we'll use the default template provider for this example. There is more information on view rendering in the Morpheus Developer Portal.

```
<div id="hypervisor-inventory-report">
   <div class="intro-stats">
      <h2>Overview</h2>
      <div class="count-stats">
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.total_items }}</span>
            <span class="stat-label">Items</span>
         </div>
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.password_items }}</span>
            <div class="stat-label">Password</div>
         </div>
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.tfvars_items }}</span>
            <div class="stat-label">TF Vars</div>
         </div>
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.secret_items }}</span>
            <div class="stat-label">Secret</div>
         </div>
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.uuid_items }}</span>
            <div class="stat-label">UUID</div>
         </div>
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.key_items }}</span>
            <div class="stat-label">Key</div>
         </div>
         <div class="stats-container">
            <span class="big-stat">{{ header.0.dataMap.random_items }}</span>
            <div class="stat-label">Random</div>
         </div>
      </div>
   </div>
   <h2>Cypher Items</h2>
   <table>
      <thead>
         <th>Key</th>
         <th>Last Updated</th>
         <th>Last Accessed</th>
         <th>Lease Timeout</th>
      </thead>
      <tbody>
         {{#each main}}
         <tr>
            <td>{{dataMap.key}}</td>
            <td>{{dataMap.last_updated}}</td>
            <td>{{dataMap.last_accessed}}</td>
            <td>{{dataMap.lease_timeout}}</td>
         </tr>
         {{/each}}
      </tbody>
   </table>
</div>
```

BUILD THE JAR

With the code written, use gradle to build the JAR which we can upload to Morpheus so the report can be viewed. To do so, change directory into the location of the directory created earlier to hold your custom plugin code.

```
cd path/to/your/directory
```

Build your new plugin.

```
gradle shadowJar
```

Once the build process has completed, locate the JAR in the `build/libs` directory.

UPLOAD THE CUSTOM REPORT PLUGIN TO MORPHEUS UI

Custom plugins are added to Morpheus through the Plugins tab in the Integrations section (Administration > Integrations > Plugins). Navigate to this section and click CHOOSE FILE. Browse for your JAR file and upload it to Morpheus. The new plugin will be added next to any other custom plugins that may have been developed for your appliance.

Once uploaded, navigate to the Reports section (Operations > Reports). You new report will appear correctly categorized, labeled, and described according to your code. Just like any other report, it can be run now or scheduled for future runs.