Rest API integration with PolicyCenter

claude.ai/chat/0a1951b9-e96b-45df-95fe-58f0fbdfddf6

I'll help you publish a REST API in PolicyCenter to integrate with an MVR (Motor Vehicle Records) application. Let me walk you through an end-to-end example.

Example: Publishing an MVR Order REST API

Let's create a REST API that allows the MVR application to receive order requests from PolicyCenter when a driver needs their motor vehicle record checked.

Step 1: Create the API Configuration in Studio

1. **Open Guidewire Studio** and connect to your PolicyCenter project
2. **Create a new REST API configuration:**
    - Navigate to `Integration` → `Outbound` → `REST APIs`
    - Right-click and select `New` → `REST API Configuration`
    - Name it: `MVROrderAPI`

Step 2: Define the REST Service in GOSU

Create a new Gosu class for your service:

gosu

```
package com.yourcompany.pc.integration.mvr

uses gw.api.rest.RestEndpoint
uses gw.api.rest.RestResponse
uses gw.api.json.JsonObject
uses javax.ws.rs.*
uses javax.ws.rs.core.MediaType
uses javax.ws.rs.core.Response

@Path("mvr/v1")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
class MVROrderRestAPI {

  /**
   * Submit an MVR order request
   */
  @POST
  @Path("orders")
  function createMVROrder(orderRequest : JsonObject) : Response {
    try {
      // Parse the request
      var driverLicenseNumber = orderRequest.get("driverLicenseNumber") as String
      var driverLicenseState = orderRequest.get("driverLicenseState") as String
      var policyNumber = orderRequest.get("policyNumber") as String
      var driverName = orderRequest.get("driverName") as String

      // Find the policy
      var policy = gw.api.database.Query.make(entity.Policy)
        .compare("PolicyNumber", Equals, policyNumber)
        .select().first()

      if (policy == null) {
        return Response.status(Response.Status.NOT_FOUND)
          .entity({"error": "Policy not found"})
          .build()
      }

      // Create MVR order (custom logic)
      var mvrOrder = createMVROrderRecord(policy, driverLicenseNumber,
                                         driverLicenseState, driverName)

      // Send to external MVR provider
      sendToMVRProvider(mvrOrder)

      // Return success response
      var responseData = new JsonObject()
      responseData.put("orderId", mvrOrder.OrderID)
      responseData.put("status", "SUBMITTED")
      responseData.put("message", "MVR order created successfully")

      return Response.status(Response.Status.CREATED)
        .entity(responseData)
        .build()

    } catch (e : Exception) {
      return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
        .entity({"error": e.Message})
        .build()
    }
  }

  /**
   * Get MVR order status
   */
  @GET
  @Path("orders/{orderId}")
  function getMVROrderStatus(@PathParam("orderId") orderId : String) : Response {
    try {
      // Query for the MVR order
      var order = findMVROrder(orderId)

      if (order == null) {
        return Response.status(Response.Status.NOT_FOUND)
          .entity({"error": "Order not found"})
          .build()
      }

      var responseData = new JsonObject()
      responseData.put("orderId", order.OrderID)
      responseData.put("status", order.Status.Code)
      responseData.put("driverLicenseNumber", order.DriverLicenseNumber)
      responseData.put("orderDate", order.OrderDate)

      return Response.ok(responseData).build()

    } catch (e : Exception) {
      return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
        .entity({"error": e.Message})
        .build()
    }
  }

  // Helper methods
  private function createMVROrderRecord(policy : Policy,
                                       licenseNum : String,
                                       state : String,
                                       name : String) : MVROrder {
    // Implementation to create your custom MVR order entity
    // This would be specific to your data model
  }

  private function sendToMVRProvider(order : MVROrder) {
    // Logic to call external MVR provider API
  }

  private function findMVROrder(orderId : String) : MVROrder {
    // Query to find MVR order by ID
  }
}
```

Step 3: Configure Authentication

In `config/config.xml` or through Studio, configure authentication:

xml

~~~
<authentication>
  <rest-api>
    <service name="MVROrderRestAPI">
      <security-scheme>basic-auth</security-scheme>
      <allowed-roles>
        <role>mvrintegration</role>
      </allowed-roles>
    </service>
  </rest-api>
</authentication>
```

### Step 4: Deploy and Test

1. **Build and deploy** your changes to PolicyCenter

2. **Access the API** at:
```
   https://your-pc-server:port/pc/rest/mvr/v1/orders
~~~

3. **Test with a POST request:**

bash

```
   curl -X POST \
     https://your-pc-server/pc/rest/mvr/v1/orders \
     -H 'Content-Type: application/json' \
     -H 'Authorization: Basic <base64-encoded-credentials>' \
     -d '{
       "driverLicenseNumber": "D1234567",
       "driverLicenseState": "CA",
       "policyNumber": "PC-0001234",
       "driverName": "John Doe"
     }'
```

4. **Test GET request:**

bash

```
curl -X GET \
  https://your-pc-server/pc/rest/mvr/v1/orders/ORD-12345 \
  -H 'Authorization: Basic <base64-encoded-credentials>'
```

Step 5: Create Webhook for MVR Results (Optional)

Create an inbound REST endpoint to receive MVR results:

gosu

```
@POST
@Path("orders/{orderId}/results")
function receiveMVRResults(@PathParam("orderId") orderId : String,
                          results : JsonObject) : Response {
  try {
    var order = findMVROrder(orderId)

    if (order == null) {
      return Response.status(Response.Status.NOT_FOUND).build()
    }

    // Update order with results
    order.Status = MVROrderStatus.TC_COMPLETED
    order.ViolationCount = results.get("violationCount") as Integer
    order.AccidentCount = results.get("accidentCount") as Integer
    // ... update other fields

    return Response.ok({"message": "Results received"}).build()

  } catch (e : Exception) {
    return Response.status(Response.Status.INTERNAL_SERVER_ERROR).build()
  }
}
```

Key Considerations:

- **Error Handling**: Implement comprehensive error handling for network failures, data validation, etc.
- **Security**: Use OAuth 2.0 or API keys for production instead of basic auth
- **Logging**: Add detailed logging for troubleshooting
- **Rate Limiting**: Consider implementing rate limiting
- **Versioning**: Include API version in the path (v1, v2, etc.)
- **Documentation**: Generate OpenAPI/Swagger documentation

Would you like me to elaborate on any specific part of this integration, such as the data model for MVR orders or error handling patterns?