The Agentforce Action Blueprint.

How to build deterministic boundaries, prevent LLM hallucinations, and execute secure AI tasks on Core.

The Hallucination Trap

Enterprises are rushing to turn on Salesforce's Agentforce, expecting it to magically resolve support cases and qualify leads out of the box. But here is the harsh reality of enterprise AI:

An autonomous agent is only as smart as the strict parameters you build for it.

If you plug a Large Language Model (LLM) into a messy CRM without defining exact operational boundaries, you are not buying automation. You are paying for a highly articulate chatbot that will confidently hallucinate bad data, promise discounts you don't offer, and expose records it shouldn't see.

> Agentforce does not guess how to execute tasks. It relies on a "ReAct" (Reasoning + Acting) architecture. If an explicit Action (Flow or Apex) does not exist, the Agent cannot perform the task.

The Architecture of Control

To deploy Agentforce safely, structure deployments around three foundational pillars. If any of these are missing, the AI implementation will fail.

1. Topics (The Boundaries)

The "Job Description." Topics restrict the LLM. If the Agent's Topic is "Service Resolution," it will strictly refuse to answer questions about marketing campaigns or sales pricing.

2. Instructions (The Rules)

The natural language guardrails. Instructions tell the Agent exactly *when* and *how* it is allowed to fire an Action, and what data it must collect first.

3. Actions (The Execution Engine)

The actual code. Agentforce cannot write to the database directly. It must pass parameters to an **Action** (Apex Invocable or Autolaunched Flow). The Action executes securely using standard Salesforce permission models, and returns the result to the AI.

In this blueprint, we provide 4 battle-tested, enterprise-grade Actions. You will see the exact System Prompts and the accompanying backend logic required to make Agentforce safe for production.

Action 1: Secure Order Lookup

Scenario (Service Cloud): A customer asks the Agent for the status of their order. Without guardrails, an AI might try to query orders based on the user's name, potentially exposing another customer's data via a loose SOQL match. We must enforce a strict Action that requires an exact ID.

1. The Agent Instruction (System Prompt)

TOPIC: ORDER MANAGEMENT

```
You are a secure Order Management Agent. Your primary goal is to provide order statuses.

CRITICAL RULES:
1. You MUST NEVER attempt to look up an order using a customer's name, email, or phone number.
2. You may ONLY execute the OrderLookup Action if the user provides a valid, 10-character alphanumeric Order ID.
3. If the user does not provide the ID, politely ask them for it before proceeding.
4. Do not invent, infer, or assume an Order ID under any circumstances.
```

2. The Apex Action (The Execution Engine)

Agentforce communicates with Apex via the `@InvocableMethod` annotation. The AI automatically extracts the 10-character ID from the natural language chat and maps it to the `orderId` variable.

```
/**
 * @description Invocable method called by Agentforce to fetch order status.
 * Enforces WITH USER_MODE to respect sharing rules and FLS.
 */
public class AgentAction_OrderLookup {

    public class OrderRequest {
        @InvocableVariable(required=true description='The 10-character Order ID')
        public String orderId;
    }

    public class OrderResult {
        @InvocableVariable(description='The status of the order to return to the AI')
        public String statusMessage;

        public OrderResult(String msg) { this.statusMessage = msg; }
    }

    @InvocableMethod(label='Lookup Order Status' description='Retrieves secure order status')
    public static List<OrderResult> getOrderStatus(List<OrderRequest> requests) {
        List<OrderResult> results = new List<OrderResult>();

        for(OrderRequest req : requests) {
            // 1. Server-side validation — never trust AI input blindly
            if(String.isBlank(req.orderId) || req.orderId.length() != 10) {
                results.add(new OrderResult('Error: Invalid Order ID format.'));
                continue;
            }

            // 2. Query safely, respecting Sharing Rules and FLS
            List<Order> targetOrder = [
                SELECT Status
                FROM Order
                WHERE OrderReferenceNumber = :req.orderId
                WITH USER_MODE
                LIMIT 1
            ];

            if(!targetOrder.isEmpty()) {
                results.add(new OrderResult('Order Status: ' + targetOrder[0].Status));
            } else {
                results.add(new OrderResult('No order found matching this exact ID.'));
            }
        }
        return results;
    }
}
```


Action 2: Cross-Cloud Marketing Suppression

Scenario (Service to Marketing): The fastest way to ruin a brand's reputation is sending a promotional email while a customer is actively complaining to a Service Agent. This Action allows the AI to autonomously flag an angry Contact in Core, which Data Cloud reads during its next ingestion cycle to suppress Journey Builder sends.

1. The Agent Instruction (System Prompt)

TOPIC: CUSTOMER SUCCESS & ESCALATION

```
You are a Customer Success Agent. You must monitor the sentiment of the user during the conversation.

CRITICAL RULES:
1. If the user uses profanity, threatens to cancel their account, or expresses severe frustration (e.g., "this is unacceptable", "I want a refund"), you must immediately execute the Escalate_and_Suppress Action.
2. Pass the ContactId of the current authenticated user to the Action.
3. After executing the action, apologize for the frustration and explicitly state: "I have paused marketing communications to your account while we resolve this. Transferring you to a supervisor now."
```

2. The Execution (Autolaunched Flow)

Instead of writing custom Apex, this Action utilises a standard Salesforce **Autolaunched Flow** exposed to Agentforce. This is highly maintainable for Admins.

> **Architecture Flow:**
> Agentforce Intent Detection → Flow Execution (Core) → Data Cloud Ingestion → SFMC Suppression.

- **Input Variable:** Create a variable `var_ContactId` (Data Type: Text, Available for Input). The Agent automatically maps the context user to this variable.
- **Step 1 (Update Records):** Configure an Update Record element targeting the Contact object where `Id` equals `var_ContactId`.
- **Step 2 (Set Field Value):** Update the custom field `Is_Actively_Escalated__c` to `TRUE`.
- **Step 3 (Return State):** Pass a text variable back to the Agent confirming "Record Updated Successfully."

The Data Cloud Impact: Because Data Cloud is connected to Core CRM, this flag is ingested on the next scheduled ingestion cycle. The Contact falls out of the 'Active Marketing' DMO segment, ejecting them from all live Marketing Cloud journeys. Note: ingestion latency depends on your org's Data Cloud batch schedule configuration.

Action 3: BANT Lead Qualification

Scenario (Sales Cloud): An unauthenticated user is chatting with an Agent on your website. Instead of just creating a blank Lead record, the Agent is instructed to run a conversational BANT (Budget, Authority, Need, Timeline) framework, and only execute the creation Action once the data is gathered.

1. The Agent Instruction (System Prompt)

TOPIC: INBOUND SALES

```
You are an Inbound Sales Development Representative (SDR). Your goal is to qualify website visitors before routing them to an Account Executive.

CRITICAL RULES:
1. You must naturally guide the conversation to discover the user's Timeline (when they need the solution) and Need (what problem they are solving).
2. DO NOT ask these questions like a robot. Weave them into natural conversation.
3. Once you have identified the user's First Name, Last Name, Email, Company, Timeline, and Need, execute the Create_Qualified_Lead Action.
4. If the user refuses to provide an email address, you may not execute the Action.
```

2. The Execution (Autolaunched Flow via Agentforce)

The AI Agent handles the unstructured, messy human conversation. It uses its LLM reasoning to extract the BANT data, maps the extracted values to the Flow's input variables, and passes them to an Autolaunched Flow that enforces CRM validation rules.

Input Variables mapped by AI:

- `var_FirstName (Text)`
- `var_LastName (Text)`
- `var_Email (Text)`
- `var_Company (Text)`
- `var_Timeline (Text)`
- `var_ExtractedNeed (Text)`

Flow Logic Steps:

- **Check:** Does Email exist in System?
- **Create:** Lead Record.
- **Assign:** Map 'ExtractedNeed' to Lead Description.
- **Route:** Push to Omni-Channel queue.

If the Flow fails (e.g., a duplicate matching rule fires in Salesforce), the Flow passes the specific error message back to the Agent. The Agent then seamlessly says to the user: *"It looks like you already have an account with us under that email. Would you like me to connect you with your dedicated Account Manager?"*

Action 4: Data Cloud Profile Retrieval

Scenario (Data Cloud + Agentforce): Agentforce is incredibly powerful when combined with Data Cloud. In this scenario, the Agent needs to know the customer's Lifetime Value (LTV) and Churn Risk — which are Calculated Insights stored in Data Cloud — before deciding whether to offer a 20% retention discount.

1. The Agent Instruction (System Prompt)

TOPIC: RETENTION & BILLING

```
You are a Retention Specialist. A customer is asking to cancel their subscription.

CRITICAL RULES:
1. Before responding to a cancellation request, you MUST execute the Fetch_Customer_Insights Action using the user's ContactId.
2. The Action returns a single decision flag, discountAuthorized. If it is TRUE, offer the 20% retention discount. If it is FALSE, process the cancellation politely without offering one.
3. Do not calculate eligibility yourself, and do not infer LTV or churn risk from the conversation. Rely only on the flag the Action returns.
```

2. The Apex Action (Querying Data Cloud)

This Apex Invocable uses the `ConnectApi` namespace to query a Data Cloud Calculated Insight Object (CIO), identified by the `__cio` suffix. Critically, it evaluates the discount rule **server-side** and returns a single `discountAuthorized` flag, so the financial decision is never left to the LLM's reading of raw numbers. Verify the exact `ConnectApi.CdpQuery` method signature against your installed Data Cloud package version, as the API surface evolves across releases.

```
/**
 * @description Retrieves Calculated Insights from Data Cloud and returns a
 * server-side retention decision. The discount rule is evaluated in Apex,
 * never delegated to the LLM.
 * Note: Verify ConnectApi.CdpQuery method names against your Data Cloud package version.
 */
public class AgentAction_FetchInsights {

    public class InsightRequest {
        @InvocableVariable(required=true description='The ContactId of the customer requesting cancellation')
        public String contactId;
    }

    public class InsightResult {
        @InvocableVariable(description='Server-side decision: may the Agent offer the discount?')
        public Boolean discountAuthorized;
        @InvocableVariable(description='Human-readable summary for the Agent')
        public String message;
    }

    @InvocableMethod(label='Fetch Data Cloud LTV' description='Queries Data Cloud CIO and returns a discount decision')
    public static List<InsightResult> getCustomerInsights(List<InsightRequest> requests) {
        List<InsightResult> results = new List<InsightResult>();

        for(InsightRequest req : requests) {
            InsightResult res = new InsightResult();
            res.discountAuthorized = false;

            // Query the Calculated Insight Object in Data Cloud
            String sql = 'SELECT LifetimeValue__c, ChurnRiskScore__c'
                       + ' FROM LTV_Insight__cio'
                       + ' WHERE ContactId__c = \'' + String.escapeSingleQuotes(req.contactId) + '\'';

            try {
                ConnectApi.CdpQueryInput queryInput = new ConnectApi.CdpQueryInput();
                queryInput.sql = sql;

                ConnectApi.CdpQueryOutput response = ConnectApi.CdpQuery.queryAnsiSql(queryInput);

                if(response.data != null && !response.data.isEmpty()) {
                    Map<String, Object> row = (Map<String, Object>)response.data[0];

                    Decimal ltv = (row.get('LifetimeValue__c') != null)
                        ? Decimal.valueOf(String.valueOf(row.get('LifetimeValue__c'))) : 0;
                    String risk = String.valueOf(row.get('ChurnRiskScore__c'));

                    // Business rule evaluated server-side — the LLM only relays this flag
                    res.discountAuthorized = (risk == 'High' && ltv > 5000);
                    res.message = res.discountAuthorized
                        ? 'Retention discount authorized.'
                        : 'Retention discount not authorized.';
                } else {
                    res.message = 'No insights found for this contact.';
                }
            } catch (Exception e) {
                res.message = 'System Error: ' + e.getMessage();
            }
            results.add(res);
        }
        return results;
    }
}
```


The Agentforce Readiness Checklist

Before purchasing Agentforce licenses or deploying agents to external-facing channels, pass this 5-point data hygiene and security audit.

1. Flawless Field-Level Security (FLS)

Agentforce respects Salesforce security models. If your Admins historically granted "View All Data" or broad FLS permissions to standard profiles just to avoid writing sharing rules, the Agent will expose sensitive internal data. FLS must be properly restricted before AI deployment.

2. Action Modularity

Are your Flows built as massive, single-canvas monoliths? Agentforce cannot easily interact with a Flow that has 40 decision nodes and screen elements. Legacy logic must be refactored into modular, Autolaunched Flows that handle one specific task (e.g., "Create Case", "Update Address") to serve as clean Actions.

3. Data Cloud Identity Resolution

If you have 4 duplicate Contact records for "John Smith" in Salesforce, Agentforce will struggle to apply Actions to the correct record. Identity Resolution via Data Cloud (or strict MDM deduplication) must be complete before AI deployment.

4. Apex User Mode Enforcement

Review all legacy Apex code that Agentforce might invoke. Any such code must enforce `WITH USER_MODE` in SOQL queries to guarantee the AI cannot bypass sharing rules and FLS.

5. Grounding Data Quality

If you are using Knowledge Articles to ground the Agent's responses via RAG (Retrieval-Augmented Generation), those articles must be audited. An AI will confidently serve an outdated, 4-year-old Knowledge Article to a customer as if it were current fact.