Artificial Genius *Orthix*:

Non-Hallucinatory Agentic Unstructured Data Workflow Platform

Customer Manual

Version AWS 2.5x (October 23, 2026)

Mission: Unlocking Commercial Returns from Unstructured Data

Enterprises have a lot of ***valuable unstructured data*** which, if ***the accurate and relevant information hidden in it*** could be delivered to the ***right person at the right time*** – would create significant opportunities for sales, revenue, and other ***measurable commercial returns***.

Making It Possible: A New Generation of Language Models

Achieving the until now elusive combination of reliability, accuracy, and relevance required for enterprise use of language models necessitates a generational shift in how language models are created and applied to obtain the maximum commercial value from their capabilities. A perspective on the history of language models explains the need for a new paradigm:

First Generation (1950s): Researchers used symbolic logic to build deterministic, rule-based models. While safe, these models lacked fluency and could not scale.

Second Generation (1980s–present): The shift to probabilistic models (culminating in the Transformer architecture) unlocked incredible fluency. However, because these models predict the next token based on probability, they suffer from unbounded failure modes (hallucinations) that are difficult to engineer away.

Third Generation (Artificial Genius approach): Rather than a new generation that replaces the old, we're moving from the rigidity of symbolic logic and the unpredictability of probabilistic models toward a hybrid architecture. This approach uses the generative power of Second-Generation base models to understand context but applies a deterministic layer to verify and produce output. It's the convergence of fluency and factuality.

It's mathematically difficult to prevent standard generative models from hallucinating because the extrapolative, generative process itself causes errors. Artificial Genius addresses this by using the model strictly ***non-generatively***. In this paradigm, the vast probability information learned by the model is used only interpolatively on the input. This allows the model to comprehend the innumerable ways a piece of information or a question can be expressed

without relying on probability to generate the answer. To create this third-generation capability, Artificial Genius post-trains existing Second-Generation base models (Amazon Nova).

This patented method ***intelligently removes the output probabilities from the model***. While standard solutions attempt to ensure determinism by lowering the temperature to zero (which often fails to address the core hallucination issue), Artificial Genius post-trains the model to tilt log-probabilities of next-token predictions toward absolute ones or zeros. This fine-tuning forces the model to follow a single system instruction: don't make up answers that don't exist.

This creates a mathematical loophole where the model retains its genius-level understanding of data but operates with the safety profile required for finance and healthcare.

Non-Hallucinatory Foundation Models

Artificial Genius' third-generation, non-hallucinatory foundation models are post-trained from well-respected, second-generation foundation models. Artificial Genius has gone beyond solving the original hallucination problem, to understand the broader contextual needs for reliability in the enterprise and is therefore offering a variety of multiple models for different applications.

After purchasing the ***Artificial Genius Orthix*** on AWS Marketplace, Artificial Genius will securely share your selected non-hallucinatory foundation model(s) with you. Please contact Artificial Genius to select which of the below model(s) you would like to be securely shared with you.

Our fast, lightweight models are post-trained from ***Amazon Nova Lite 1.0.*** These models have very low inference costs, but are text-only. These models will be securely shared with you via ***Bedrock Custom Model Import***. Once imported, you will need to note the resulting ***model deployment ARN(s)*** as the agent identifier in the workflow platform. The terms of your license to the ***Orthix*** models incorporate the terms of the Amazon Nova license. These models are:

Orthix-30303: this is the original ***pure non-hallucinatory*** language model with 0.03% hallucination rate. It has been trained to distinguish strictly answerable from strictly non-answerable non-generative questions. In practice, when given questions with some degree of ambiguity, will mostly refuse to answer them. It is application-dependent whether this is too strict a notion of non-hallucination.

Orthix-33333-10-Pct: this is a language model that has been trained to distinguish non-generative questions that are either answerable or non-answerable, up to a ***controlled amount of ambiguity*** that would be contextually resolvable by a human. This model has a hallucination rate of 0.6%, not counting as hallucinations answers that would be contextually correct based on human judgment despite the controlled amount of logical ambiguity.

Our full-featured, frontier-level models are post-trained from ***Qwen 3.6 27B***. These models have multi-modal capabilities, such understanding computer source code, other specialized text, and even images. These ***SageMaker*** models will be securely shared with you via ***AWS Resource Access Manager***. Once imported and deployed, you will need to note the resulting ***inference endpoint name(s)*** as the agent identifier in the workflow platform. These models are:

Orthix-50000: this is a ***pure non-hallucinatory, multi-modal frontier*** model with 0.06% false negative and 0.06% false positive hallucination rate. It has been trained to distinguish strictly answerable from strictly non-answerable non-generative questions. The base model it is trained from is a full-featured foundation model, considered one of the top open-weight models for understanding computer source code.

What is a Non-Generative Question?

It is important to understand that these Third-Generation language models are ***not chatbots***. To work around the impossibility of removing hallucinations, these models are designed to reliably answer ***non-generative questions***. A non-generative question is one for which the model ***wouldn't need to use any probabilities***, in principle, to generate the sequence of tokens for the answer, but rather could extract or verify information based solely on the input context.

While short answers (such as dates or names) are obviously non-generative, it's also possible to output long sequences deterministically as well. For example, asking for a direct quote from a document to justify a previous answer is a non-generative task.

We'll show some examples of both answerable and non-answerable, non-generative questions, regarding the following input context:

> **Context**: "Financial performance remained strong through the third quarter. Our revenue grew by 15% year-over-year, driven by robust sales in the enterprise segment."

Answerable, non-generative, short answer:

> **Question**: "What was the annual revenue growth?"
>
> **Answer**: "15%"

Answerable, non-generative, long-answer, follow-up question, based on the answer to the previous one:

> **Question**: "Provide a quote from the document showing that the annual revenue growth was 15%."
>
> **Answer**: '"Our revenue grew by 15% year-over-year, driven by robust sales in the enterprise segment."'

Unanswerable, short-answer question:

Question: “What was the CEO’s bonus this year?”

Answer: “Unknown”

Enterprise Workflow Platform

Enterprise workflows ***process unstructured data*** from selected input databases or feeds, to produce and ***disseminate unstructured data*** as entries or reports in output databases or feeds.

On the ***Orthix*** platform, Artificial Genius’ third-generation language non-hallucinatory foundation models can be used to make these workflows safe, reliable, and domain-relevant. The method is to decompose the information processing step into a set of hallucination-free, ***non-generative extractions*** of relevant information from the input data, followed by ***reassembly*** of these extracted items into the final report. When the number of input sources is large and stored in a document database, the extracted items may be precomputed and stored in an ordinary relational database.

Orthix enables each domain-specialized department to create their own workflows on demand, using a special semi-structured prompt.

Specifically, a **Product Requirements Document (PRD)** is an industry-standard methodology for specifying a desired software artifact. In Artificial Genius’ platform, the Agent accepts the PRD in the form of a ***semi-structured prompt*** in the industry-standard ***Markdown*** format to specify the workflow. This PRD includes the specification of:

- Non-hallucinatory language model;
- Input source of unstructured data;
- Output sink for the resulting structured or unstructured data;
- Non-generative questions to extract relevant information from the inputs;
- A Markdown template to reassemble the extracted information into the outputs.

Each domain-specialized department can be supplied with suitable PRD templates by the AI leadership that enable them to use the ***Orthix*** platform in a self-service manner to create custom, domain-specific workflows on demand.

In more detail, the unique Artificial Genius twist is that the Markdown in the PRD can include both special, **“eager”** and regular, **“lazy”** Markdown link definitions. An “eager” link has the property that any reference it will be automatically and immediately substituted by its value, while a “lazy” link is followed only when the user actively clicks on it. “Eager” links are distinguished by having labels of the form ***dash-separated-words-or-numbers***, while “lazy” links have labels of the form ***space separated words or numbers*** (in either case, case-insensitive).

The ***parameters*** that govern the PRD are defined by **"eager"** links, including those with the following reserved labels:

- Top-level reserved labels (**"Name"**, **"Agent"**, **"Input"**, **"Output"**), that are interpreted by the workflow platform itself, to configure the overall Agent and Input/Output servers required for the operation of the workflow.
- All labels of the form **"-…-"**, starting and ending with a dash, that are passed as parameters to the Agent (without the enclosing dashes).
- All labels of the form **"-…"**, starting with a dash, that are passed as parameters to the Input server (without the enclosing dashes).
- All labels of the form **"…-"**, ending with a dash, that are passed as parameters to the Output server (without enclosing dashes).
- All labels starting with the letter **"Q"**, followed by a series of ***one or more dash-separated numbers***, that are interpreted as non-generative questions to ask the Agent about the Input to produce the Output, in the hierarchically numbered order. A number in the label may in certain cases be replaced by an ***open wildcard "?"*** or ***exact wildcard "!"***, as explained below.
- All other labels starting with ***dash-separated-words***, followed by a series of ***one or more dash-separated numbers***, that are interpreted as user-defined "eager" links. A number in the label may, again, be replaced by an ***open wildcard "?"*** or ***exact wildcard "!"***.

Product Requirements Document (PRD)

Before providing a detailed specification of the PRD fields, we'll run through an example PRD to show the concept of how to design a workflow using this semi-structured Markdown prompt.

This example uses the open-source "***enronsent***" unstructured dataset, consisting of emails between Enron officers and employees leading up to the company's collapse. The goal of the workflow is to establish a communication graph between the various personnel. The complete PRD looks like this:

```
[Name]: Enron Analysis
[Agent]: arn:aws:bedrock:us-east-1:*:custom-model-import/*
[Input]: fs
[Root-]: enronsent
[Div-]: mailbox
[Output]: s3
[-Bucket]: my-bucket
[-Root]: enronsent
[Q1]: To whom did the author of the Document say something?
[Q2]: Provide a quote from the text showing what the author of the
Document said to [Q1].
[Q3]: Who said something to the author of the Document?
[Q4]: Provide a quote from the text showing what [Q3] said to the
author of the Document.
```

```
# Communication Analysis

[[From-] said "[-]" to [Q1].][Q2]

[[Q3] said "[-]" to [From-].][Q4]
```

Note that the non-generative questions asked in the PRD can depend on answers (if they exist) to previous questions. This supports deeper information analysis, where assertions can be backed up with further evidence and deeper, structured interrogation contingent on certain information existing in the input.

Note also that the PRD output template can use both Markdown shortcut link references like `[Q1]`, which are replaced by their values (for questions, their answers, if they exist), and Markdown regular link references like `[…][Q2]`, specifying link text, where the link text is only rendered if the link `[Q2]` has a valid value, which can be substituted into the link text as `[-]`. Finally, note that the choice of `[Agent]` in the PRD means that the Orthix platform can make use of a variety of non-hallucinatory language models created via the Orthix IP, including custom models adapted to specific customer use cases.

This example PRD template is available for download at: https://agipub.s3.us-east-1.amazonaws.com/artificial-genius-orthix-prd-example.md .

Now we'll run this PRD through the Orthix platform, on some actual entries in the database, using the ***Orthix-30303*** third-generation foundation model. This demo includes some entries where the PRD questions are unanswerable, and so would normally risk hallucinations with a second-generation language model.

Document 1

```
I just spoke with Clark Smith, head of Elpaso's merchant arm. I told
him that we had been hearing that El Paso was blaming Enrononline for
problems in Western gas markets. He asked for some more specifics about
who exactly was spreading the rumor (I told him we had heard it from 3-
4 sources). He acknowledged that EOL was not the problem; said he
couldn't believe that it had been identified as such; and said he would
bring it up on his call with his Washington team this afternoon.
I think he will put it to rest (except for whatever damage has already
been done). I did promise to get some more specifics on who has told us
that El Paso pointed to us. Can anybody give some info on that?
```

Questions 1

```
[Q1]: Clark Smith
[Q2]: I told him that we had been hearing that El Paso was blaming
Enrononline for problems in Western gas markets
[Q3]: Clark Smith
[Q4]: He acknowledged that EOL was not the problem; said he couldn't
believe that it had been identified as such; and said he would bring it
up on his call with his Washington team this afternoon
```

Answers 1

Communication Analysis

Steven J Kean@ENRON said "I told him that we had been hearing that El Paso was blaming Enrononline for problems in Western gas markets" to Clark Smith.

Clark Smith said "He acknowledged that EOL was not the problem; said he couldn't believe that it had been identified as such; and said he would bring it up on his call with his Washington team this afternoon" to Steven J Kean@ENRON.

Document 2

Dear Mr. Kean,
Heidi Van Genderen asked me to send you a reminder notice. We will be loading all the power point presentations onto 1 laptop. If you could please send me your presentation by friday morning (5/18), it would be greatly appreciated.
Christine Velez Badar The Centers at University of Colorado-Denver 1445 Market St., Suite 380 Denver, CO 80202 phone: 303.820.5674 fax: 303.820.5656 email: cbadar@carbon.cudenver.edu

Questions 2

[Q1]: Mr. Kean
[Q2]: We will be loading all the power point presentations onto 1 laptop
[Q3]: Heidi Van Genderen
[Q4]: Heidi Van Genderen asked me to send you a reminder notice

Answers 2

Communication Analysis

cbadar@carbon.cudenver.edu said "We will be loading all the power point presentations onto 1 laptop" to Mr. Kean.

Heidi Van Genderen said "Heidi Van Genderen asked me to send you a reminder notice" to cbadar@carbon.cudenver.edu.

Document 3

Attached is a draft of the letter we'd like to send to our 16,000 residential customers on Friday. Please review and let me know your comments by 12 noon on Wednesday.

Questions 3

[Q1]: Unknown

```
[Q2]: <not asked>
[Q3]: Unknown
[Q4]: <not asked>
```

Answers 3

```
Communication Analysis

<blank>
```

Sample Enterprise Use Cases

Summarization

Portfolio managers need timely access to key information from analyst reports as soon as they are published. The volume and length of analyst reports is such that a portfolio manager does not have time to read all of them carefully.

Instead, the portfolio manager would like concise and timely summaries of the reports, targeted to their investment objectives. Asking a chatbot to “summarize” the reports would result in irrelevant and hallucinated information.

So the solution is to prepare a PRD to prompt the ***Orthix*** platform to create a workflow to deliver relevant summaries of the reports to the portfolio manager in a timely manner. The PRD specifies:

- The data feed for the analyst reports.
- The specific items of information that might or might not be in an analyst report, that are relevant to the portfolio manager’s objectives.
- A template to reassemble the specific items of information into a well formatted summary.
- The delivery channel for the formatted summary.

Search

A representative fields numerous requests for proposal (RFPs) from clients with particular investment objectives, who would like an investment plan, with specific portfolio assets, entry and exit triggers, etc., justified by the client’s investment objectives.

In many cases, elements of the client’s investment objectives may be similar to previous clients, for whom an investment plan has been prepared. To increase productivity, representatives

would like an AI agent to prepare a draft investment plan in response to an RFP, based on the plans prepared for similar clients, that the representative can use as a starting point, and save hours of research time.

The solution is for the representative's department to prompt the ***Orthix*** platform to create a workflow to input new RFPs, and deliver relevant investment plan drafts to representative in a timely manner. This involves two PRDs, one to analyze the investment plans, and the other to analyze the RFPs.

The investment plan PRD specifies:

- The email inbox to which the investment plans sent to clients are CC'd.
- The specific items of information which characterize the elements of the investment plan that are responsive to specific client investment objectives, which are used to label the elements of the plan.
- The relational database in which these labeled items should be stored.

The RFP PRD specifies:

- The email inbox in which the RFPs are received.
- The specific items of information from the RFP that characterize the client's investment objectives.
- The relational database to search for the investment plan elements, labeled by those items characterizing the investment objectives.
- The template to reassemble the labeled investment plan elements into a draft investment plan.
- The email for the appropriate representative to receive the draft investment plan.

More specifically, the following reserved link definitions are used within the PRD to configure the workflow:

Markdown Specification

[Name]: specifies the name of the workflow.

[Agent]: specifies the non-hallucinatory language model, as a Bedrock model ARN.

[-Field-]: specifies parameters for the Agent, surrounded by dashes. Currently available Agent fields include **[-Timeout-]**, **[-Tries-]**, and **[-Max-Tokens-]**. The **[-Timeout-]** parameter governs how long the workflow will wait for an Agent response (default 60 seconds).

[Input]: specifies the data input server according to ***Apache OpenDAL*** standards (not MCP). Currently whitelisted input servers are ***S3*** (Amazon Simple Storage Service), ***HTTP*** (Web server), and ***FS*** (local filesystem, internal to the platform container, for test purposes). The workflow will ***process all available documents*** on the specified input server; however, this scope may be narrowed by specifying a **[Root-]** on that server as described below.

[Field-]: specifies any parameters necessary to set up the connection with the input server, according to Apache OpenDAL standards:

- ***S3***: a **[Bucket-]** must be specified. A **[Root-]** within the filesystem on the bucket may also be specified to narrow the scope.
- ***HTTP***: the **[Endpoint-]** base URL of the website must be specified. Because HTTP servers do not offer a way to automatically list all the files on the server, in this case, a **[List-]** of space-separated files to be processed must also be specified. A **[Root-]** within the filesystem on that website may also be specified to narrow the scope.
- ***FS***: a **[Root-]** within the local filesystem in the container may be specified to narrow the scope. For security, this **[Root-]** is restricted to be relative to the part of the local filesystem accessible to the **[Agent]**, not to the entire local filesystem in the container.

[Div-]: if provided, specifies the method to split up the documents retrieved from the input server into self-contained parts to which the workflow will be applied. Currently supported splitting methods are ***h1***, ***h2***, …, etc., for splitting up documents at header levels 1, 2, …, etc., for a variety of document formats (HTML, Markdown, DOCX, PDF/UA); and ***mailbox,*** for documents containing a sequence of email messages.

When documents are split, an implicit **[Part-]** field will be supplied to the PRD to designate the specific title of the part of the document that is being processed. In the case of mailbox splitting, an implicit **[From-]** field will also be supplied.

[Output]: specifies the data output server according to Apache OpenDAL standards. Currently, the same server options are whitelisted as for the input servers.

[-Field]: specifies any parameters necessary to set up the connection with the output server, according to Apache OpenDAL standards. These parameters are the same as noted above for the input servers, except that the qualifying dash must be placed at the beginning instead of end of the field name.

[Q1]:, [Q2]:, [Q2-1]:, … specify the ***non-generative questions*** that the agent will answer about each input document (or part of a document), whose answers are to be reassembled into the output. These questions are hierarchically numbered according to the dash-separated numerical parts of their labels and will be asked in the corresponding order.

Due to the “eager” dereferencing of reserved links, chained questions are automatically supported: each question may contain reserved link references to prior questions in the hierarchical ordering, which will be “eagerly” substituted by their answers, before the question containing the link is asked of the Agent. If any such linked question is unanswerable, the question linking to it will also be deemed unanswerable. For example,

```
[Q1]: What is the name of the company this Document is about?
[Q2]: What was the annual revenue of [Q1] in 2025?
```

Moreover, unbounded chains of questions may be created using link definitions that include wildcards in the label. For example,

```
[Q3-1]: What is the name of a customer of [Q1]?
[Q3-?]: Excluding [Q3-?], what is the name of another
customer of [Q1]?
```

enables an arbitrarily long list of customer names to be extract from the Document, producing corresponding link labels `[Q2-1]`, `[Q3-2]`, … Because questions are always asked in the hierarchically numbered order, when each wildcard question is being asked, only the prior matches to the wildcard `[Q3-?]` exist to be substituted into the current wildcard question.

To allow parallel chains of questions, a wildcard question `[Q4-?]` that needs to refer to just the specific corresponding answer in a parallel series of questions `[Q3-?]`, can do this by replacing the `?` open wildcard with the `!` exact wildcard:

```
[Q4-?]: What product does [Q1] purchase from [Q3-!]?
```

[Infinity]: specifies the maximum number of wildcard iterations when generating questions. The default value is 6.

[My1]:, [User2]:, [User2-1]:, … specify ***user-defined eager links***, which are hierarchically named and numbered according to the dash-separated word and numerical parts of their labels, and will be substituted by their values in the corresponding order. (In particular, names beginning with letters before ‘Q’ will be substituted before the questions, and those beginning with letters after ‘Q’ will be substituted after the questions.)

Markdown text that will form the ***output document*** corresponding to each input document.

The Markdown may contain shortcut link references like `[Q1]`, or regular link references, like `[link text][Q1]`, to the above reserved link definitions. As mentioned, reserved link references are “eagerly” dereferenced, and immediately substituted by their link texts, prior to any further processing. The default link text of a reserved link reference is its value, or in the case of a question, the answer to that question, if answerable.

If needed, to distinguish two adjacent reserved shortcut link references `[Q1][Q2]` (a

Markdown syntax error) from a regular link reference `[link text][Q2]`, an "eager" zero-width space `&;` Markdown entity can be inserted between them, as in `[Q1]&;[Q2]`, which will not appear in the final rendered output.

For reserved link references including explicit link text, instead of relying on the default, their link text may consist of general Markdown, including further nested reserved link references within link text, thereby supporting ***recursive logic***, in fully declarative form. This ability to nest reserved links is an extension to the Markdown syntax, supported by the ***Orthix*** Markdown engine, but is fully compatible with all standard downstream Markdown engines, because all such nested reserved links will be "eagerly" substituted into flat Markdown prior to the final rendering of output.

As a specific example of "eager" link nesting, the link value of a reserved link reference may be explicitly substituted into its link text, via the special reserved link reference **[-]**. As noted, the default link text of a reserved link reference is its link value (not its link label). Thus, the two reserved link references below are equivalent, and either one will be substituted with the link value:

```
[label]
[[-]][label]
```

If the value of a reserved link is undefined (for example an unanswerable question), then the entire link text will be omitted during the "eager" substitution, whether or not the link text includes the (in this case, undefined) link value. Thus, if question `[Q1]` is either answerable with answer "A1", or unanswerable, then the reserved link:

```
Checking: [the answer is [-]][Q1]
```

will be substituted with the first or second text below, respectively:

```
Checking: the answer is A1
Checking:
```

To complete the support for ***if-then-else logic***, in fully declarative form, every reserved link `[X]` has a corresponding anti-link **[~X]** that has no value, but whose link text is only substituted exactly when `[X]` does ***not*** have a value. Thus, the above example could implement the opposite logic:

```
Checking: [there was no answer][~Q1]
```

which will be substituted with the first or second text below, respectively, depending on whether `[Q1]` has an answer:

```
Checking:
Checking: there was no answer
```

If a reserved link reference label includes a wildcard, then the values of all the reserved links whose labels match that wildcard, and have defined values, will be substituted, in

the hierarchically numbered order, into copies of the link text, thereby supporting ***loop logic***, in fully declarative form. Separator link text, that should not be included in the final copy of the link text, can be placed with the special "eager" link reference **[,]**, which has default separator text ", ", or **[separator text][,]** if a non-default separator is desired. Thus, if there are questions `[Q3-1]` and `[Q3-2]`, with answers "Answer 3(a)" and "Answer 3(b)", then the two reserved links below are equivalent:

```
[Q3-?]
[[-]&;[,]][Q3-?]
```

and either will be substituted with:

```
Answer 3(a), Answer 3(b)
```

Similarly to the notation for parallel question definitions, within the link text for a parallel wildcard question `[Q4-?]`, the specific corresponding value of a parallel chain of questions `[Q3-?]` can be substituted using the exact wildcard notation **[Q3-!].** In addition, the numerical value of the exact wildcard may be substituted into the link text with the special "eager" link reference **[!].** Thus, the reserved link:

```
[
[!]. [-].][Q3-?]
```

will be substituted with the Markdown numbered list:

```
1. Answer 3(a).
2. Answer 3(b).
```

Running a Workflow

The ***Orthix*** container is designed to be provisioned as an AWS ECS service, typically by a central IT group. Once provisioned, this section describes how each department can run its own workflows on demand, without any further need for centralized resources.

Users provided with the ECS service can run any number of workflows by invoking the standard container endpoint for ***Bedrock AgentCore*** agents, `/invocations` on port `8080`. The POST input for this endpoint is a JSON object of the format `{"prompt"="…"}`, where the value of the "`prompt`" field contains the PRD in Markdown format. The POST output is also a JSON object of the format `{"response"="…", …}`, where the "`response`" field contains a log of successes and failures of the workflow items, also in Markdown format.

The actual data inputs and outputs of the workflow items specified in the PRD are handled by the **[Input]** and **[Output]** servers as specified in the PRD. In particular, the results of running the workflow on each input item are written to corresponding files/entries on the **[Output]** server.

Configuration and Security

The ***Orthix*** platform is delivered as an AWS Marketplace container product, which therefore runs entirely in your customer AWS account, with no dependencies on Artificial Genius. This container meets all of the stringent enterprise security requirements for the AWS Marketplace.

A typical deployment of the container in your customer account would be as a Service in the ***Elastic Container Service (ECS).*** The container has been built for cross-platform deployment, and can therefore be run on either ***x86_64*** or ***arm64*** instances on ECS. However, arm64 instances are recommended for compatibility with Bedrock AgentCore.

Import the Non-Hallucinatory Models

The ***Orthix*** non-hallucinatory language models are delivered via ***AWS Resource Access Manager (RAM)***. This is a secure service that provides inference endpoints within your customer AWS account, by hosting and running the ***Orthix*** models in a secure, AWS-managed escrow account that:

1. Does not allow Artificial Genius any access to the prompts, data, or responses that you send to the models;
2. Does not allow you any access to the proprietary weights, either of the Artificial Genius Orthix models, nor of the underlying Amazon Nova models.

The models shared with you will appear in your ***RAM*** console for acceptance. Once the model share is accepted, you will need to deploy them via ***inference endpoints*** for them in either ***Bedrock or SageMaker,*** depending on the type of model. Please do not use **":"** characters in SageMaker inference endpoint names, as ***Orthix*** uses them to distinguish Bedrock ARNs from SageMaker endpoint names.

Once deployment is complete, note the ***model deployment ARN(s)*** for Bedrock models, or the ***inference endpoint name(s)*** for SageMaker models. You will then need to provide these identifier(s) to the teams using the ***Orthix*** platform, which may be entered into their PRD prompts as the **[Agent]** field.

Configure the Network

In the VPC console, if you don't already have one, create a VPC consistent with the network, Regions, and Availability Zones in which your resources are located, and in which you will deploy the ***Orthix*** containers.

In the VPC console, create a Security Group that is attached to the VPC above, and configure its Inbound and Outbound Rules to enable network access for the following resources and ports:

- AWS Marketplace Product Registration: TCP port 443
- Bedrock Runtime Agent: TCP port 8080
- S3: TCP ports 80, 443
- HTTP: TCP ports 80, 443
- All ICMP – IPv4

In the VPC console, to enable the container to access the required AWS services, the most secure method is to create Endpoints corresponding to the selected *region*, with Private DNS Names, using the above Security Group:

- com.amazonaws.*region*.sts (Interface);
- com.amazonaws.*region*.metering-marketplace (Interface);
- com.amazonaws.*region*.bedrock-runtime (Interface);
- com.amazonaws.*region*.s3 (Gateway, if access to S3 desired).

Configure Permissions

In the IAM console, create an IAM Role for the ***Orthix*** container, for example, "*OrthixRole*".

The Permission Policies attached to *OrthixRole* should include those necessary to give the host and container access to the resources (such as S3 buckets or HTTP sites) that it is intended to be enabled to read and/or write. In addition, to allow the host to run the ***Orthix*** container and its Bedrock AgentCore runtime, it should also attach the following permission policies:

```
AmazonEC2ContainerServiceforEC2Role

AWSMarketplaceMeteringRegisterUsage

AWSMarketplaceMeteringFullAccess

AmazonECSTaskExecutionRolePolicy

AmazonBedrockReadOnly

BedrockAgentCoreFullAccess (*)
```

(*) Note that the policy `BedrockAgentCoreFullAccess` is slightly over-permissive. IAM does not have a ready-made policy with the optimal permissions for the AgentCore Runtime execution role. See https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/runtime-permissions.html#runtime-permissions-execution for information on how to create one.

The Trust Relationships of the *OrthixRole* should follow the template: https://agipub.s3.us-east-1.amazonaws.com/artificial-genius-orthix-trust-relationships.json. Modify the template so that the Statement Id "AllowRoleToAssumeItself" allows *OrthixRole* to assume itself. This is necessary because various derived versions of the role may be automatically generated that need to be normalized back to the original role.

In addition, if you plan to use managed instances in your ECS cluster, create a second role called *ecsInfrastructureRoleForManagedInstances*. This role should have the permissions `AmazonECSInfrastructureRolePolicyForManagedInstances` and a Trust Relationship so that `ecs.amazonaws.com` can assume the role.

Create the ECS Cluster

In ECS console, create a new cluster, give it a name, say "*orthix-cluster*".

- The ***Orthix*** containers may be orchestrated in ECS using a variety of deployment options. If you select the Infrastructure Capacity type to be "Fargate and Managed instances", then select the Instance Profile to be the ARN for *OrthixRole*, and the Infrastructure Role to be the ARN for *ecsInfrastructureRoleForManagedInstances*.
- Select Network Settings to point to your above VPC and Security Group.

Create the ECS Service

In ECS console, create a Task according to the template Task Definition `https://agipub.s3.us-east-1.amazonaws.com/artificial-genius-orthix-ecs-task.json`. Replace the wildcards "*" in this template with your specifics, including the container URI provided to you when you subscribed to the ***Orthix*** product, and the *OrthixRole* above. Select the `requiresCompatibilities` field to conform to the Infrastructure Capacity type of your cluster. Then, register it as a Task Definition under a name, say "*orthix-task*".

Then Deploy the Task Definition as a Service.

- Select Cluster as *orthix-cluster*.
- Select the Networking to be consistent with the VPC and Security Group above.

The ***Orthix*** container does not accumulate any usage charges for the product itself, except when it is actually running a workflow, in which case usage is billed in hourly units. Therefore, container lifetimes and orchestration can be optimized as with any other ECS container. The orchestration grants each container a temporary access token for AWS services at startup, with a lifetime of 6 hours, after which the token is automatically refreshed.