Artificial Genius *Orthix* Cyber Agent:

Non-Hallucinatory Agentic Coding Workflow Agent

Customer Manual

Version AWS 2.60 (November 26, 2026)

Mission: Unlocking Commercial Returns from Code

Enterprises have a lot of ***mission-critical source code and system configurations*** which, if ***the accurate and relevant information in them*** could be delivered to the ***right person at the right time for action*** – could create significant, ***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 understand 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 Coding Foundation Models

Artificial Genius' third-generation, non-hallucinatory coding foundation models are post-trained from ***Qwen 3.6 27B***. These models have multi-modal capabilities beyond natural language text, such understanding computer source code and system configuration files. These ***SageMaker*** models are sold separately on AWS Marketplace. Once purchased and deployed, you will need to note the resulting SageMaker ***inference endpoint name(s)*** as the Agent identifier for the agent platform. These models are:

Orthix-50000: this is a ***pure non-hallucinatory, frontier coding*** 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 code and other modalities.

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 Cyber Workflow Agent

Enterprise cyber workflows ***process source code and system configurations***, to produce and ***disseminate other code, structured data, or unstructured data*** as entries or reports in output databases or feeds.

With the ***Orthix*** cyber agent, Artificial Genius’ third-generation language non-hallucinatory foundation coding 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 source code or configurations, followed by ***reassembly*** of these extracted items into the final report. When the number of items of input code is large and stored in a code repository, the extracted items may be precomputed and stored in an ordinary relational database.

Orthix Cyber Agent 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’ cyber agent, 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:

- The non-hallucinatory coding foundation model;
- The input file, and how to either interpret it as source code or configuration data, or subdivide it into such code or configuration blocks.
- The non-generative questions to extract relevant information from the input code or configuration blocks;
- The Markdown template to reassemble the extracted information into the output.

Each domain-specialized department can be supplied with suitable PRD templates by the AI leadership that enable them to use the ***Orthix*** cyber agent in a self-service manner to create custom, domain-specific cyber 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"**), that are interpreted by the workflow cyber agent 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 non-hallucinatory Agent (without the enclosing dashes).
- All labels of the form **"-…"**, starting with a dash, that are used as parameters to interpret the Input as code, or subdivide into code blocks (without the 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 "!"***.

Example Product Requirements Document (PRD)

Before providing a detailed specification of the PRD fields, we'll run through an example PRD – for a mailbox of text files, instead of source code – 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]: artificial-genius-orthix-30303-endpoint
[Input]: enronsent20a.txt
[Div-]: mailbox
[Lang-]: txt
[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.

In this PRD, the input context is split up into individual email messages for processing. The output Markdown document contains one top-level section for each email message, with the contents of that section being templated from the Markdown in the PRD.

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 agent 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-agent-prd-example.md .

Now we'll run this PRD through the Orthix agent, on some actual entries in the database, using the ***Orthix-30303*** third-generation (non-coding) 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>
```

Markdown Specification

[Name]: specifies the name of the workflow.

[Agent]: specifies the non-hallucinatory language model, as the name of its SageMaker inference endpoint.

[-Field-]: specifies parameters for the Agent, surrounded by dashes. Currently available Agent fields include **[-Max-Tokens-]**, which is the maximum number of output tokens (by default, 128).

[Input]: optionally specifies the input filename, which may be used as a hint about the default coding or configuration language that it is written in. The input file may also be in Markdown format (the assumed default), and if so, the code blocks in the Markdown

can specify coding or configuration languages individually.

[Field-]: specifies any parameters necessary to interpret the input context. These are detailed below.

[Lang-]: optionally specifies the either the name of the coding or configuration language being analyzed, or the corresponding filename extension (either one case-insensitive), e.g., `"c"`, `"cs"`, `"py"`, `"java"`, `"js"`, `"json"`, `"jsonl"`, `"yaml"`, etc. This field overrides any language hints from the **[Input]** filename, but individual code or configuration blocks in the file can still override this default, by further explicitly specifying a different coding or configuration language on that block.

[Div-]: if provided, specifies the method to split up **[Input]** document 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). Each such section should contain a code block as supported by the corresponding format.

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. If the [Part-] includes a filename extension, it will be used as a default override hint for any [Input] filename extension hint for the coding or configuration language. Any explicit specification of the language via **[Lang-]** or the tag on a coding block takes precedence over such hints.

[Q1]:, [Q2]:, [Q2-1]:, … specify the ***non-generative questions*** that the agent will answer about the input context (or each part of the context), 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, contingent questions are automatically supported: each question may contain eager 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.

[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.

If the input context has been subdivided by [Div-], then the PRD will be applied to each subdivision, and the final output will be reassembled from the individual Markdown outputs by making them be top-level sections, with the subdivision names as titles. This kind of bundling of Cyber Agent input/output is encouraged for efficiency.

The Markdown may contain shortcut link references like `[Q1]`, or regular link references, like `[link text][Q1]`, to the above eager link definitions. As mentioned, eager link references are "eagerly" dereferenced, and immediately substituted by their link texts, prior to any further processing. The default link text of a eager 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 eager 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 eager link references including explicit link text, instead of relying on the default, their link text may consist of general Markdown, including further nested eager link references within link text, thereby supporting ***recursive logic***, in fully declarative form. This ability to nest eager links is an ***Orthix*** extension to the Markdown syntax, supported

by its Markdown engine, but is fully compatible with all standard downstream Markdown engines, because all such nested eager links will be “eagerly” substituted into regular, flat Markdown prior to the final rendering of output.

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

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

If the value of a eager 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 eager 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 eager 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
```

To correctly interpret answers which take the form of code instead of natural language, in addition to the special link reference **[-]** mentioned above, two alternatives are provided. While the default **[-]** interprets the answer as natural language text, the link reference **[']** interprets it as inline code, while **["]** interprets it as a block of code. For example, if `[Q1]` is a question that produces inline code, its answer should be referenced as `[[']][Q1]`. (Note that this is not the same as `[`[-]`][Q1]`, because in this case, the `[-]` is merely inline code, not a link reference.) Conversely, on the input side of a question – which may differ from the output side – code within Markdown may be distinguished from natural language by quoting it inline as `` `code` `` or as a block in ```` ```code``` ````.

If a eager link reference label includes a wildcard, then the values of all the eager 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. Moreover, parallel loops of related questions are also supported.

Separator link text, whose link text will be included in all except the final copy of the link text, can be placed with the special "eager" link reference **[,]**, which has default separator text `", "`, or as **[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 eager links below are equivalent:

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

and either will be substituted with:

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

Parallel loops of questions are supported. Within either the link definition or link text for a parallel wildcard question `[Q4-?]`, the specific corresponding answer of a parallel chain of questions `[Q3-?]` can be substituted using the "exact wildcard" notation **[Q3-!].** In addition, the pure numerical value of the "exact wildcard" may be substituted into the link text with the special "eager" link reference **[!].** Thus, the eager link:

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

will be substituted with the Markdown numbered list:

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

Running a Workflow

The ***Orthix*** cyber agent is designed to be provisioned as a Bedrock AgentCore Agent. Once initially provisioned, as explained in the next section below, this section describes how to run ***Orthix*** workflows from any computer via secure Web access.

These instructions include certain information that you will have saved during the one-time provisioning process. This information is labeled by ***CAPITALIZED_BOLD_ITALICS***. Where used, this italic name should be replaced by the actual information you saved earlier.

The ***Orthix*** Cyber Agent is optimized for ***bulk workflows***. Since the Cyber Agent accepts one input at a time, for best speed, latency, and cost, a sufficient number of source code or configuration files should be bundled into a single invocation of the Cyber Agent, and then subdivided internally using **[Div-]**. Since usage is priced per million tokens, in whole number units, each invocation is effectively priced at a minimum of one million tokens.

Login Command

The following command will obtain a temporary authentication token and save it to a local file named “token.json”. The format is compatible with either Linux or Windows.

```
curl -X POST https://cognito-idp.us-east-
1.amazonaws.com/USER_POOL_ID/.well-known/openid-configuration -H
"Content-Type: application/x-amz-json-1.1" -H "X-Amz-Target:
AWSCognitoIdentityProviderService.InitiateAuth" -d "{\"AuthFlow\":
\"USER_PASSWORD_AUTH\",\"ClientId\":\"CLIENT_ID\",\"AuthParameters\":{\
"USERNAME\":\"USERNAME\",\"PASSWORD\":\"PASSWORD\"}}" >token.json
```

From the resulting “token.json” file, extract the ***ACCESS_TOKEN***, which will be valid for 30-60 minutes, and will appear there in the form

```
"AccessToken":"ACCESS_TOKEN"
```

(Do not save the quotes around the ***ACCESS_TOKEN***.)

First-Time Login

The first time you run the above command, after provisioning the agent, the ***PASSWORD*** should be the ***TEMP_PASSWORD***. The returned “token.json” file will, in that case, not contain an “AccessToken”, but instead a message to change the ***TEMP_PASSWORD***, and a ***SESSION_ID*** in the following format:

```
"Session":"SESSION_ID"
```

Here is the command to change to your desired, permanent ***PASSWORD***, which you will only have to run once ever (or if you later decide to change your password). The format is compatible with Linux or Windows.

```
curl -X POST https://cognito-idp.us-east-1.amazonaws.com/ -H "Content-
Type: application/x-amz-json-1.1" -H "X-Amz-Target:
AWSCognitoIdentityProviderService.RespondToAuthChallenge" -d
"{\"ChallengeName\":\"NEW_PASSWORD_REQUIRED\",\"ClientId\":\"CLIENT_ID\
",\"Session\":\"SESSION_ID\",\"ChallengeResponses\":{\"USERNAME\":\"USE
RNAME\",\"NEW_PASSWORD\":\"PASSWORD\"}}" >password_change_token
```

After this, the first “curl” command can be re-run to get an ***ACCESS_TOKEN***.

Agent Invocation Command

Prepare your input, consisting of the “context” you want analyzed, and the “prompt” that will analyze it, as a JSON file “input.json”, in the following general format:

```
{
  "context": "Bob ran.",
  "prompt": "[Agent]: SAGEMAKER_ENDPOINT\n[Q1]: Who ran?\n[Q1]."
}
```

Then run this command to process the “input.json” file:

```
curl -X POST https://bedrock-agentcore.REGION.amazonaws.com/runtimes/URL_ENCODED_AGENT_ARN/invocations -H "Authorization: Bearer ACCESS_TOKEN" -H "X-Amzn-Bedrock-AgentCore-Runtime-User-Id: CLIENT_ID" -H "Content-Type: application/json" -d @- <input.json >output.json
```

The “output.json” file will include a “response” field with the result.

One-Time Instructions to Provision the *Orthix* Cyber Agent

Note: in these instructions, information you will need to save and use later is named in ***CAPITALIZED_BOLD_ITALICS***. Where used, this italic name should be replaced by the actual information you saved earlier.

These instructions are intended to work straight through a fresh, unmodified AWS account (running as root). There have been occasional reports of new accounts not being provisioned with the intended minimum capabilities. If this happens, you may have to open a Support case.

Deploy the Cyber Agent Container

On the AWS Marketplace product page for the Orthix Cyber Agent:

1. Click “View Subscription”
2. Select Artificial Genius ***Orthix*** Cyber Agent
3. Click Launch
4. Select “Amazon Bedrock AgentCore”
5. Select “Console” launch
6. Under Host, click “Set Up hosting on AgentCore”

7. Select Runtime “V2”
8. Select “Micro VMs”
9. Select “ECR” container deployment
10. Under Permissions, create “default role”
11. Under Inbound auth, select “HTTP” with “JSON web tokens” and “Quick create configurations with Cognito”
12. Finally, click “Create runtime”

During the creation of the runtime, you will be presented with an opportunity to edit the permissions of the default agent role specified in step (10) above.

Select the role’s “Permissions” tab and click on the name of the role’s single pre-filled permissions policy. Edit it in JSON format to insert the following additional permission statement block so that the product can run by registering with the AWS Marketplace:

```
{
    "Sid": "MarketplaceMetering",
    "Effect": "Allow",
    "Action": [
        "aws-marketplace:MeterUsage"
    ],
    "Resource": "*"
},
```

Ignore warning about “no resource specified”.

Finally, for future steps, obtain key information about the Agent you have just deployed:

1. Open the AWS Management Console and search for the “Amazon Bedrock AgentCore” service page.
2. In the left-hand navigation pane, go to the “Agents” Section.
3. Click on the deployed agent container product, where it says “Runtime”.
4. On the agent dashboard details page, select the “Configuration” tab, and then under “Runtime Details”, locate and copy the “Runtime ARN”. This name will include a ***REGION***, such as “us-east-1”, somewhere in the string.
5. The ***URL_ENCODED_AGENT_ARN*** is obtained by taking this ARN and applying the following character replacements:

- Replace every ":" with "%3A"
- Replace every "/" with "%2F"

Deploy the Non-Hallucinatory Language Model

Once you also purchase an Artificial Genius non-hallucinatory, coding foundation language model from the AWS Marketplace, suitable for the Cyber Agent, you may deploy it by creating a SageMaker inference endpoint for it. Be sure to record the ***SAGEMAKER_ENDPOINT*** name.

Configure Secure Web Authentication for the Agent

To set up secure Web authentication to the Cyber Agent using the simplest username/password method, follow the instructions below. If you are more familiar with Cognito, a variety of other authentication configurations are supported.

1. In the AWS Console, search for the "Amazon Cognito" Console.
2. Click on "User pools" in the left sidebar.
3. Look for the user pool created by the Cyber Agent deployment (it usually contains the name of your product or "AgentCore" in the title).
4. Click on the user pool name. The ***USER_POOL_ID*** (something like "us-east-1_aBc123XyZ") will be displayed prominently at the top under the pool name. Note the Region prefix (e.g., us-east-1).
5. Click on the "App Clients" tab inside your user pool dashboard, and then on the name of the corresponding Agent client. Record the ***CLIENT_ID***.
6. Click the "Edit" button, then on the new page, under "Authentication Flows", check the box by "Sign in with username and password: ALLOW_USER_PASSWORD_AUTH", and click "Save Changes".
7. Click on the "Users" tab inside your user pool dashboard, and then click the "Create User" button. Set a ***USERNAME***, then choose to set a ***TEMP_PASSWORD***. Finally, click "Create User".

This completes the installation and setup. Now you can login into, and then run, the ***Orthix*** Cyber Agent, using one-line "curl" commands on any computer, as explained in the previous section.