**Setup and Installation**

## Components Used

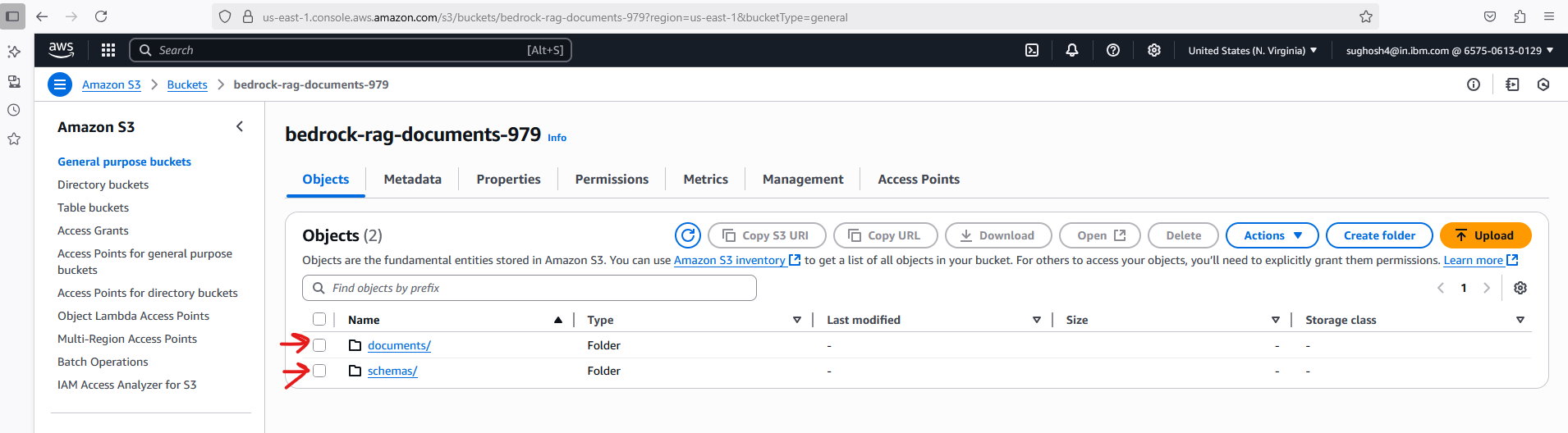
|  |  |
| --- | --- |
| **Component** | **Service** |
| Document Store | Doc store on Amazon S3 |
| Vector DB | Amazon OpenSearch Serverless |
| LLM API | Anthropic Claude 3.5 Sonnet |
| Agent Hosting | AWS Lambda |
| Agent Wiring | Bedrock Agent Builder (Lambda Actions) |
| Frontend | Static Web UI on Amazon S3 |
| Orchestration | Lambda calls via API Gateway |
| Company Wiki | Static Website Mimicking Company Wiki |
| IAM Roles | Roles defined for Bedrock Agent Permissions |

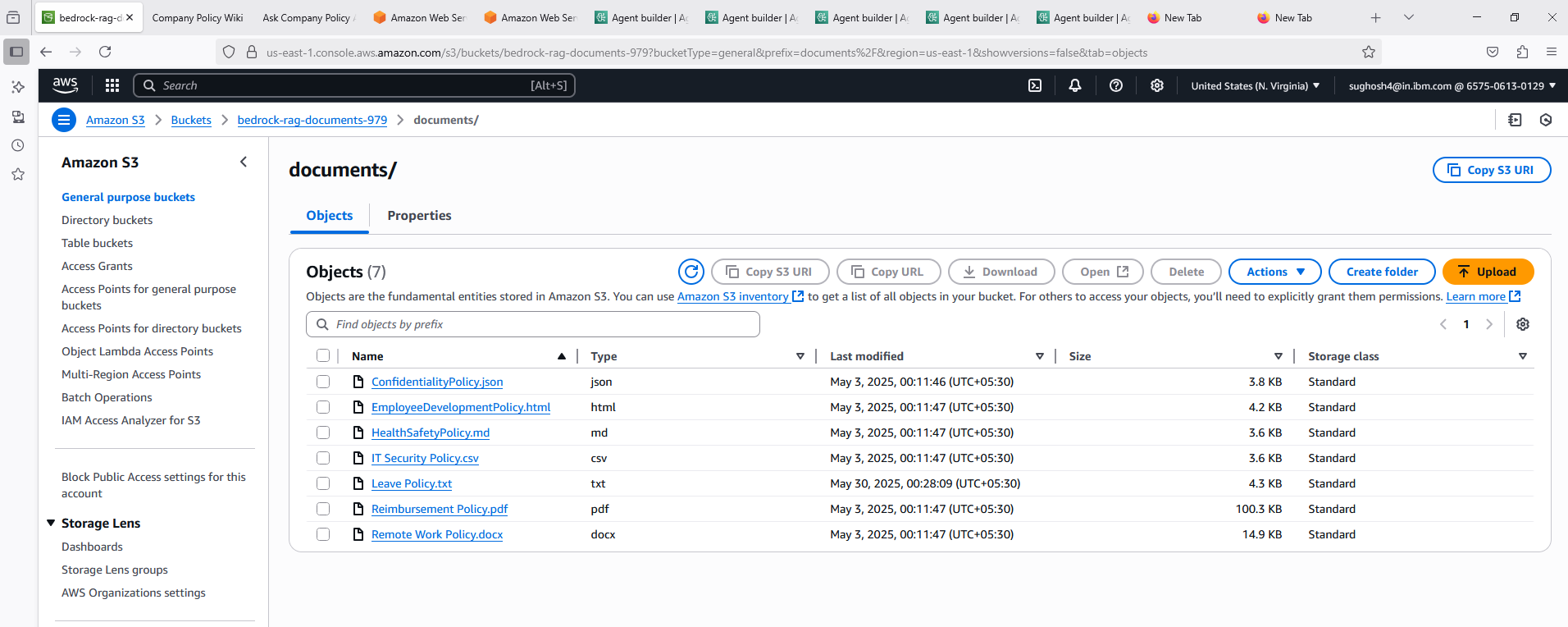
All Component codes , supporting documents are available in Github.

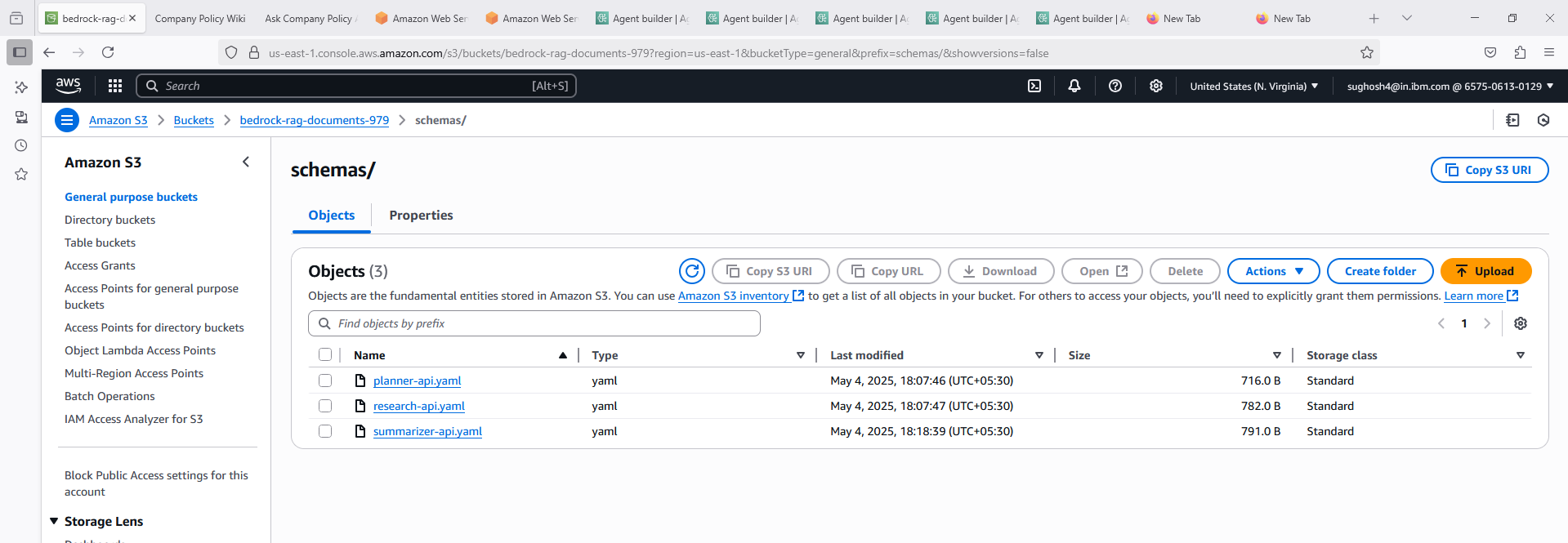
**Step1 : Setup Document Repository on S3.**

Upload all your documents into an Amazon S3 bucket. Amazon OpenSearch Serverless will be connected to this S3 bucket. Aa you can see I have generated some dummy company policy documents of various formats and uploaded to s3 bucket.Also upload Yaml files.

Create two subfolders under your s3 bucket namely documents and schemas. Upload Dummy documents under documents and Yaml files under scemas

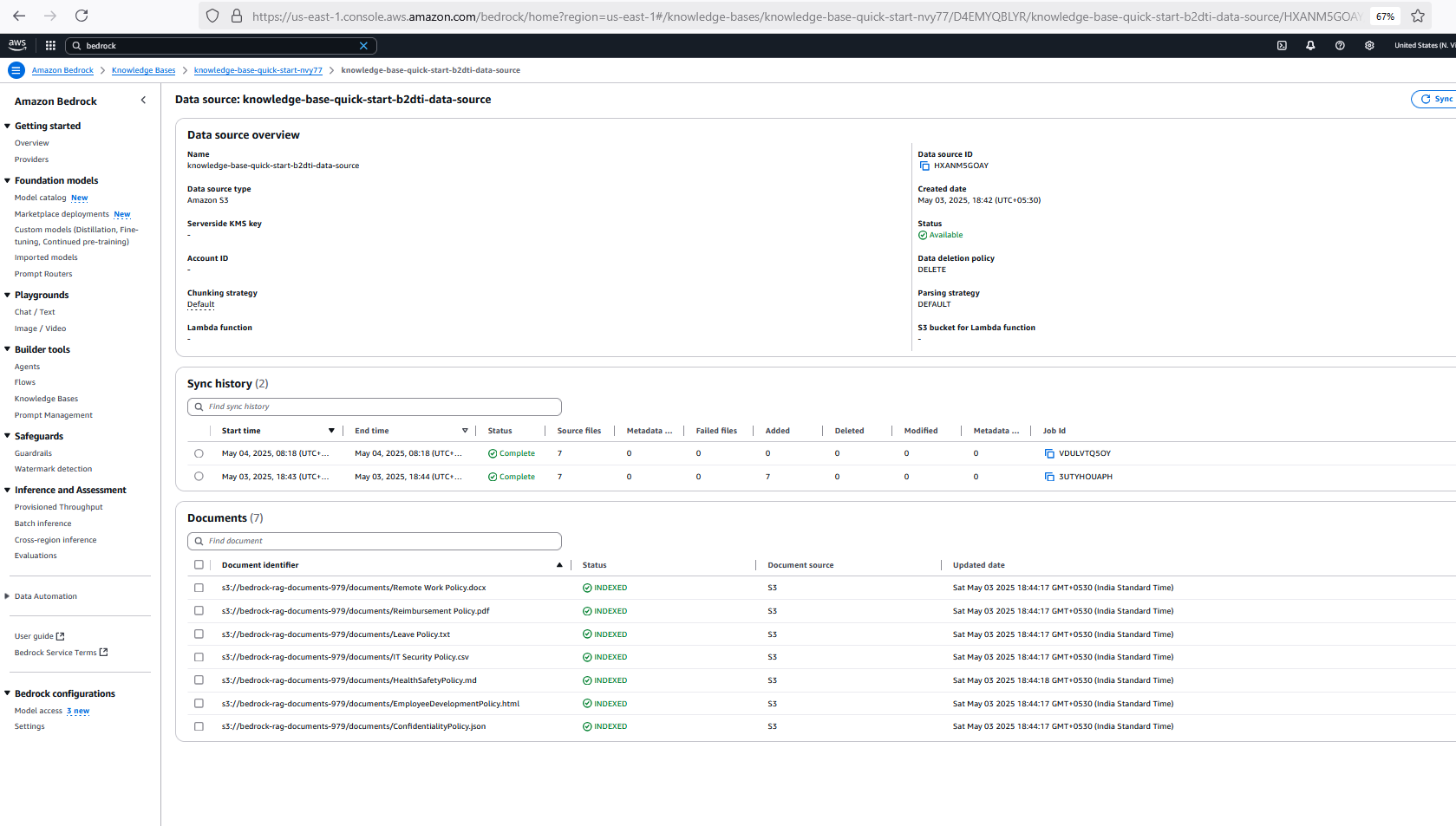




****

**Step2: Create a knowledge base and Link to your S3 Bucket**

Knowledge base created and linked to S3 Doc repository



**Step3: Create Agents as below and Define Roles**

**Go to Bedrock Console → *Agents* → Create Agent**

* **Orchestrator Agent** – Main entry point; decides whether to use vector DB or go to agents.
* **Planner Agent** – Breaks query into tasks.
* **Research Agent** – Performs external web search.
* **Summarize Agent** – Summarizes & formats research findings.
* **Critique Agent** – Evaluates if the response answers the user's original question.

**Common Agent Setup:**

* **Name**: E.g., OrchestratorAgent
* **Foundation Model**: Choose Anthropic Claude 3 or AWS Titan, or if AWS Q is available, experiment with that.
* **Instructions** (role-based):
  + **Orchestrator**:

You are the main orchestrator. First query the knowledge base. If enough context is found, answer the user directly. If not, initiate task planning and call the Planner agent.

* + **Planner**:

Your task is to break down the user query into actionable research tasks and pass each to the Research Agent.

* + **Research Agent**:

You are responsible for web searches using the Web Search Tool. Return concise, relevant snippets.

* + **Summarize Agent**:

You will take web search results and summarize them into clear, concise answers.

* + **Critique Agent**:

You will review the final summary and decide if it fully answers the original user query. Return either “Goal Met: Yes” or “Goal Met: No” and a reason.

Create Each Above Agents using AgentBuilder and define above respective roles. Select Model as Anthropic Claude 3.5 Sonnet.Below screenshot shows OrchestratorAgent Setup.Setup Similar for other agents.

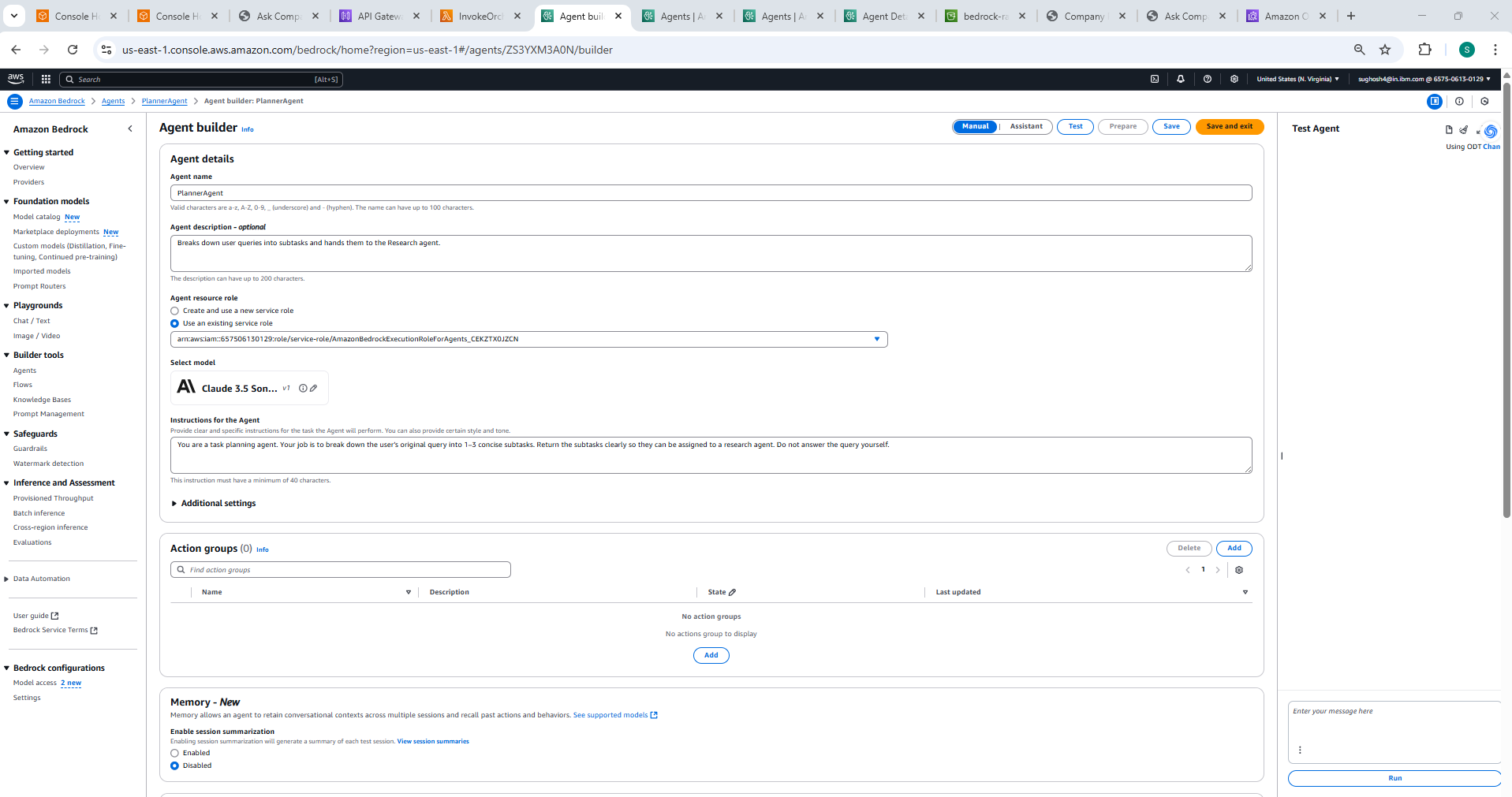
See Screenhsots of each Agent:

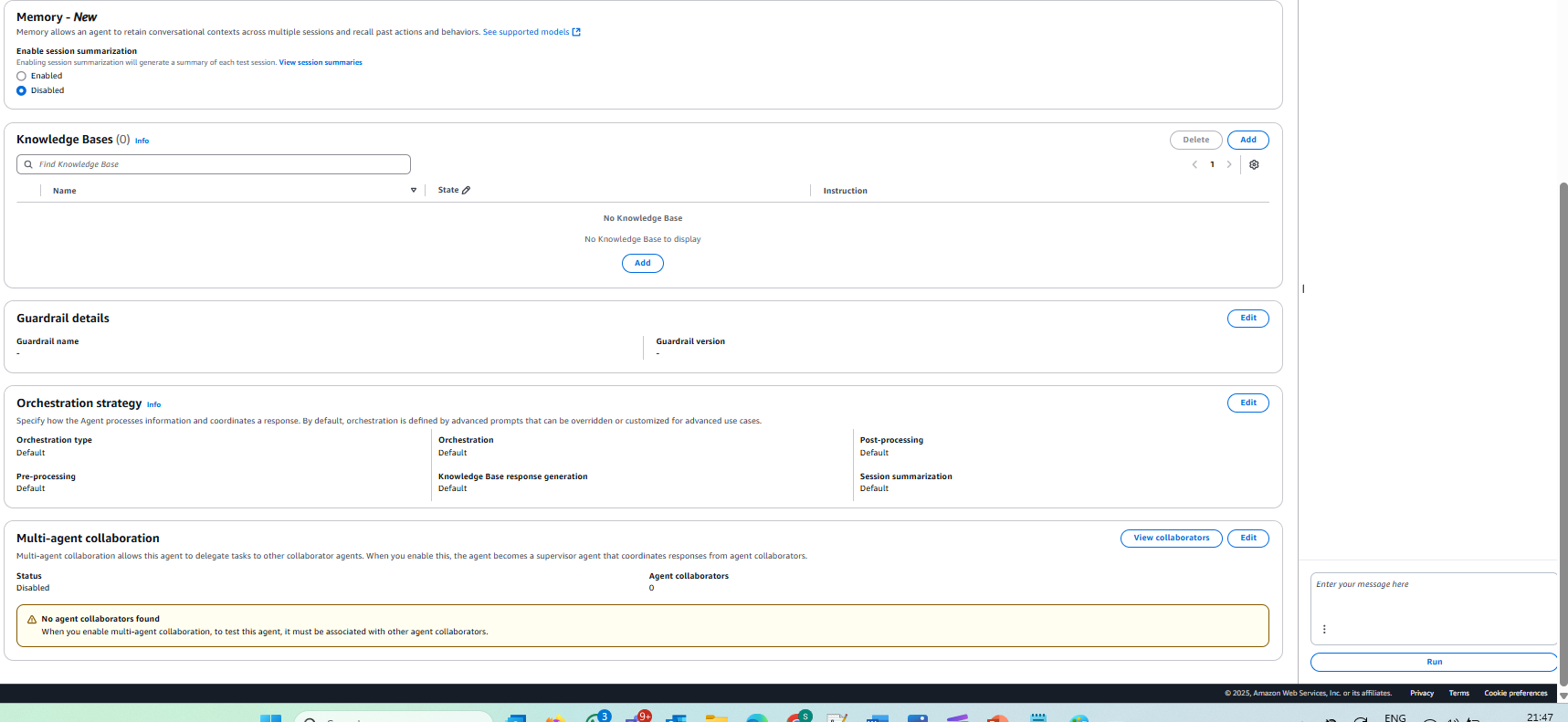
Create Subagents First (PlannerAgent , ResearchAgent, SummarizerAgent, CritiqueAgent) and the create the OrchestratorAgent which calls the Subagents

**PlannerAgent**

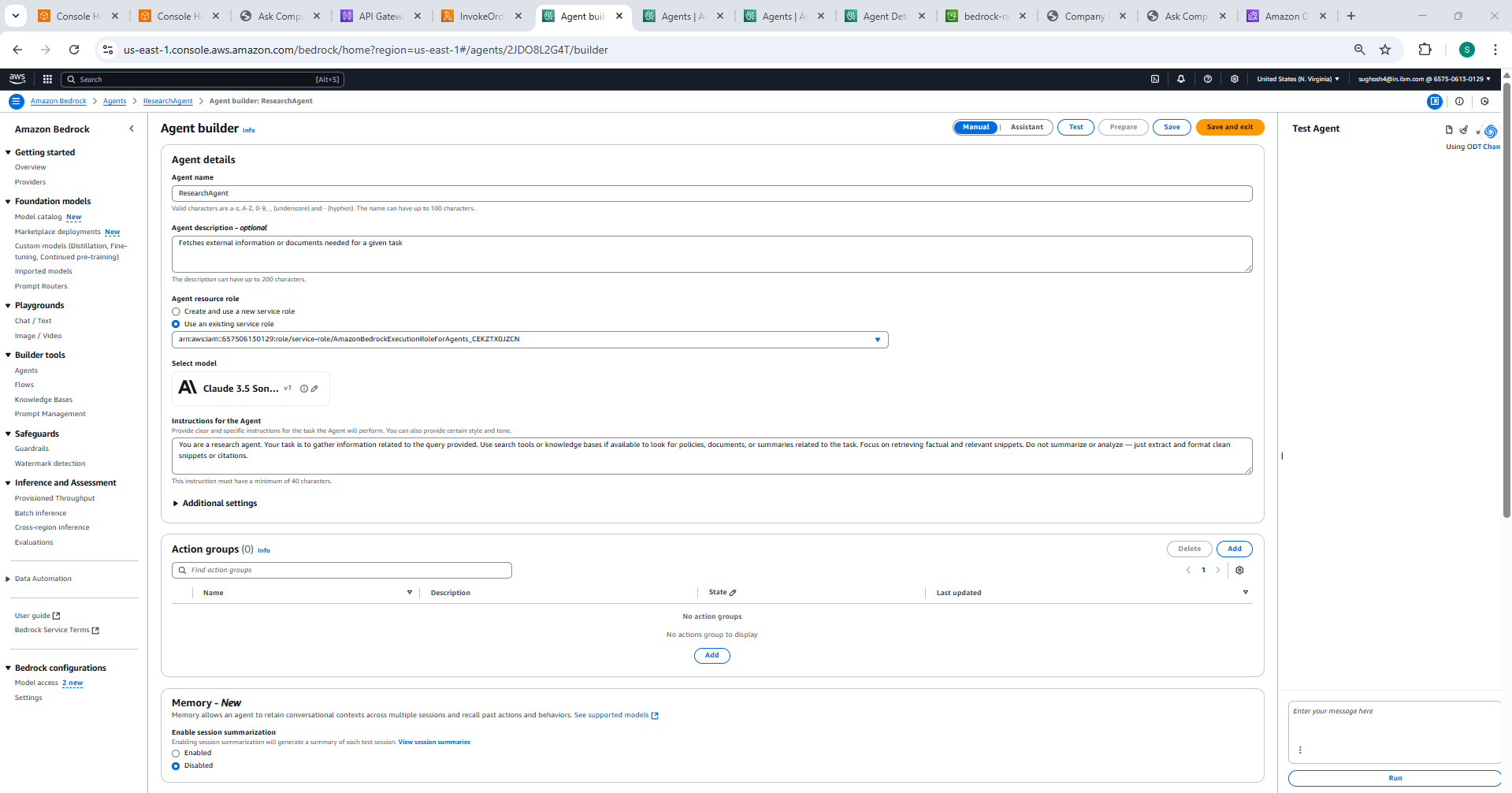
**Field Descriptions:**

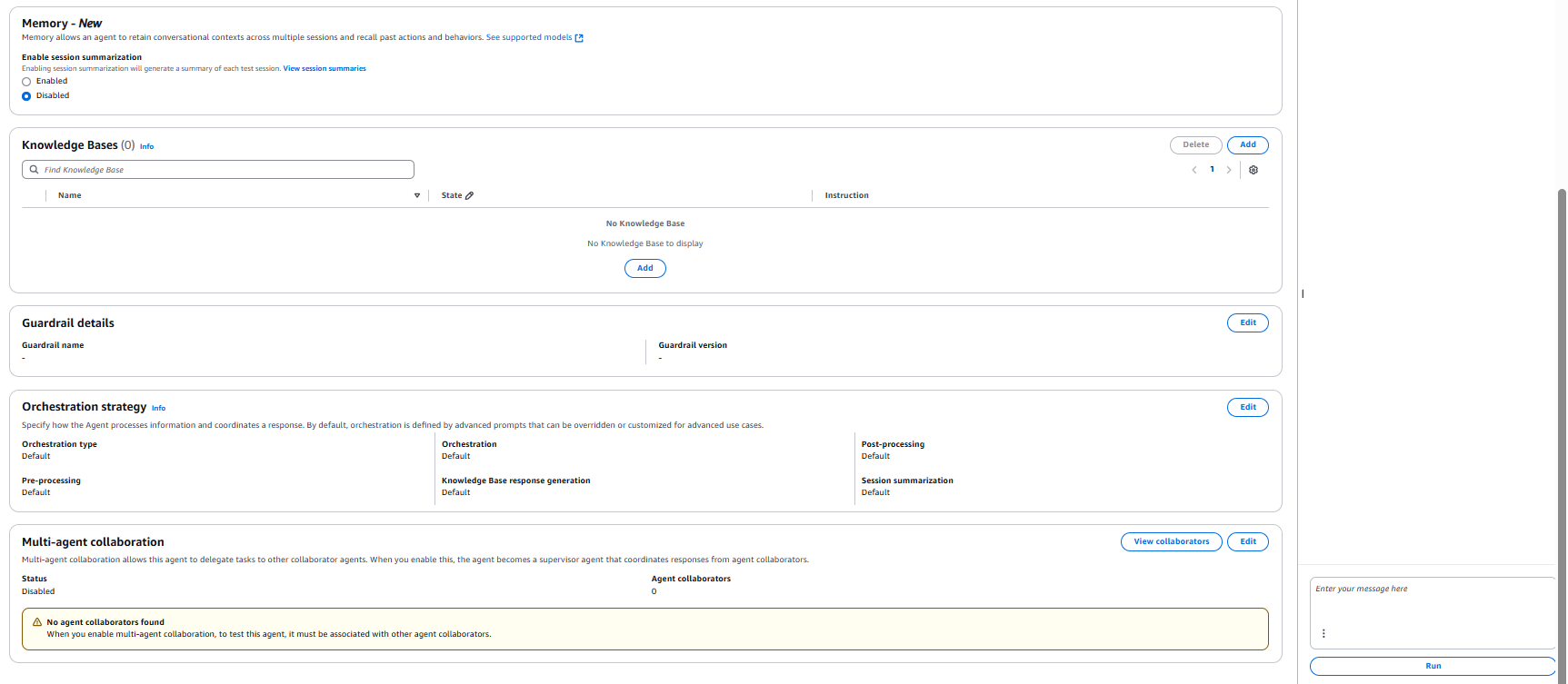
For Agent name choose an appropriate name , Add a Description , very important define a clear unambiguous role , Select Claude 3.5 Sonnet Model which will power the agent .Planner agent will be stateless (So no memory needed.So Memory is disabled). Reason being , Its just going to plan the tasks are return to orchestrator.MultiAgent Collaboration is disabled as its not collaborating with multiple agents unlike orchestrator. Its just interacting with the OrchestratorAgent so MultiAgent Collaboration is disabled. For all the sub agents the settings would be same except the “instructions for the Agent”. Fill as per the screenshpts below and create each subagents.



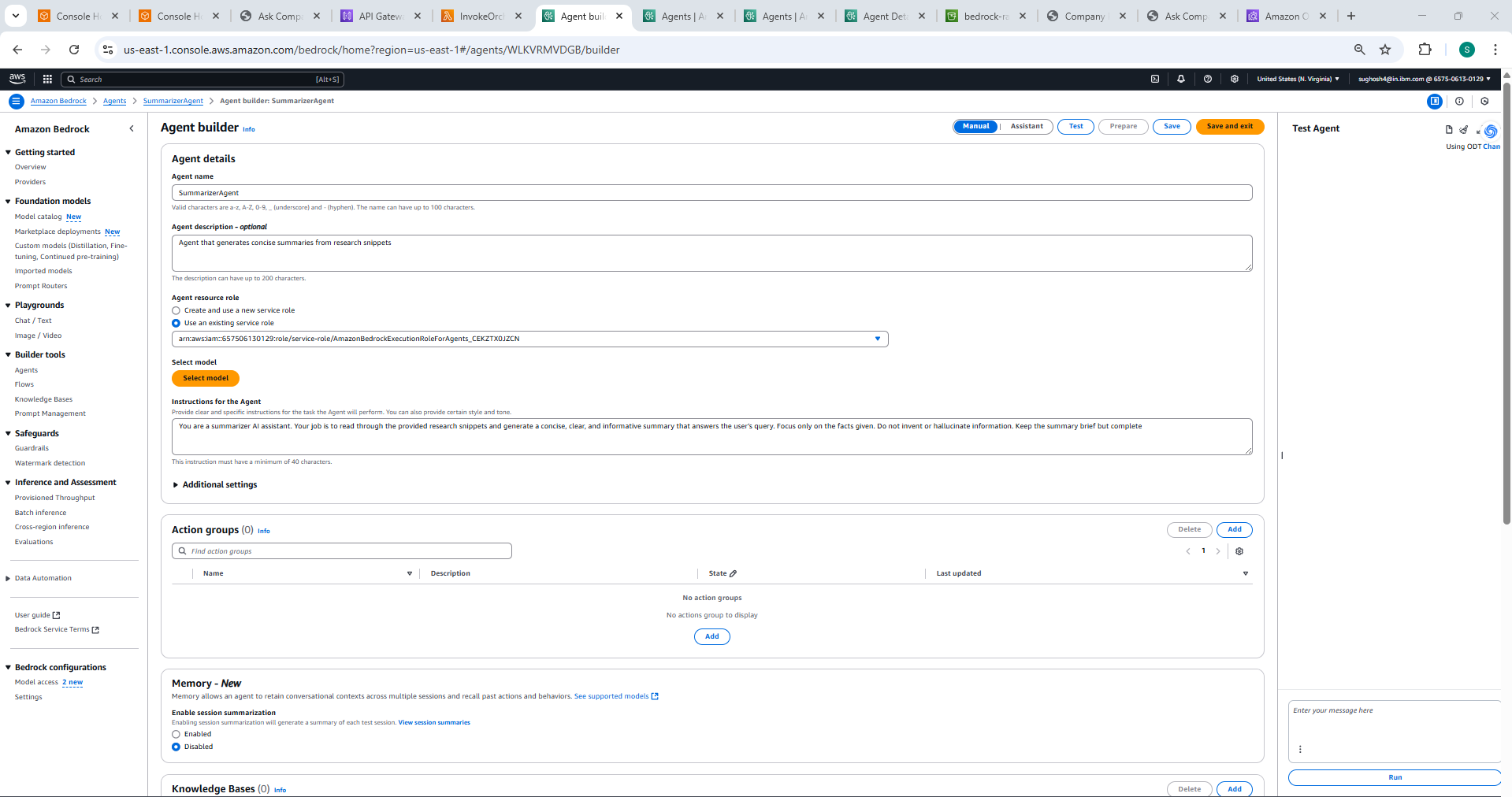


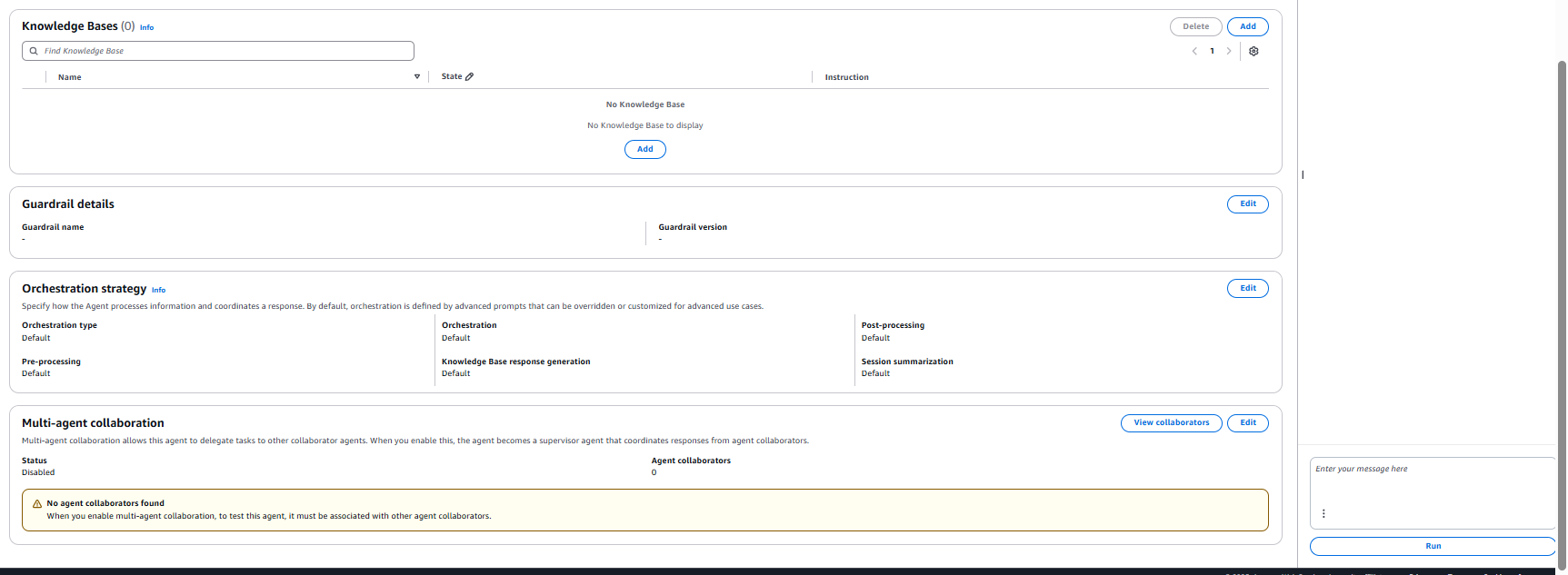
ResearchAgent

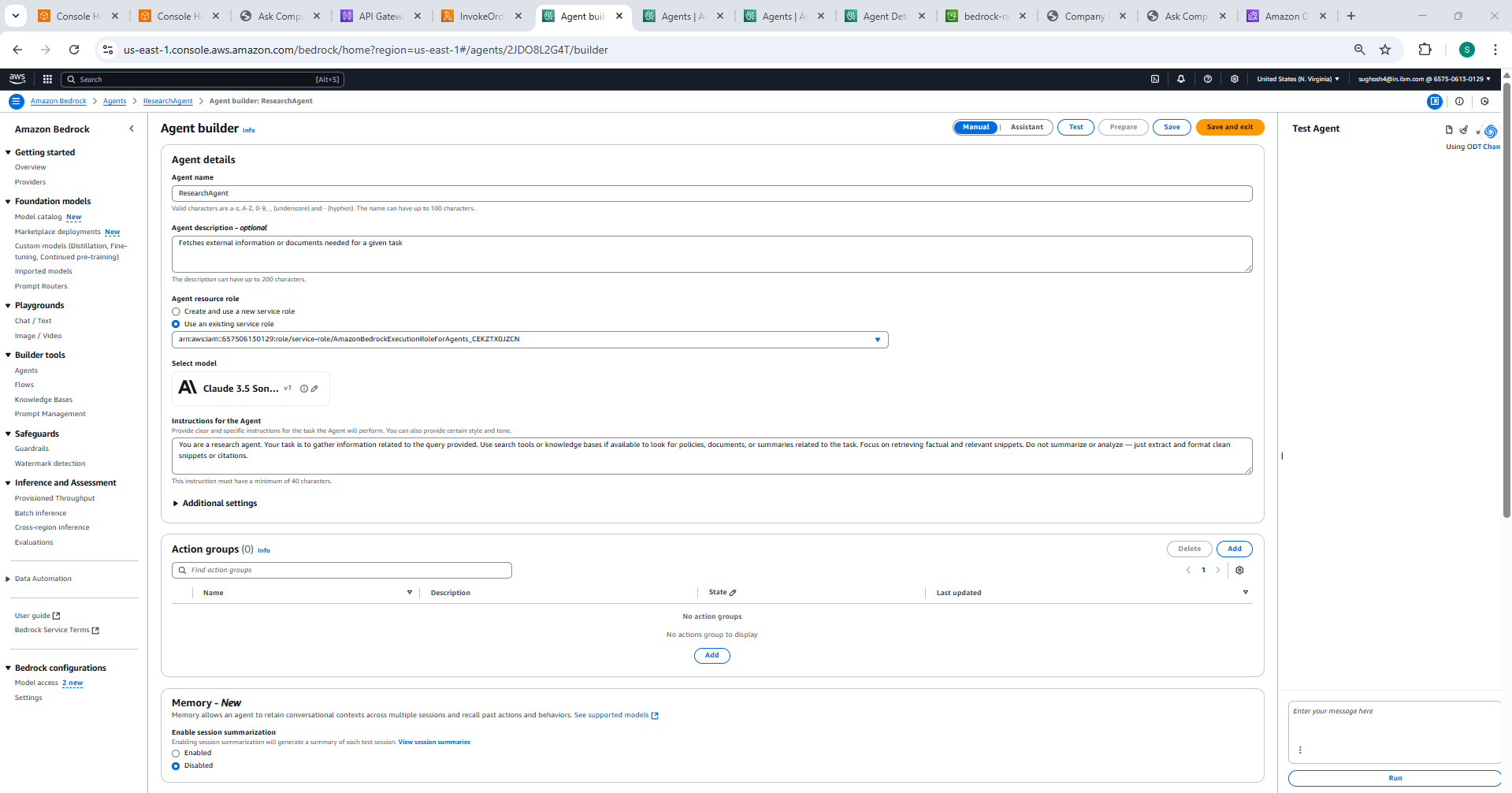




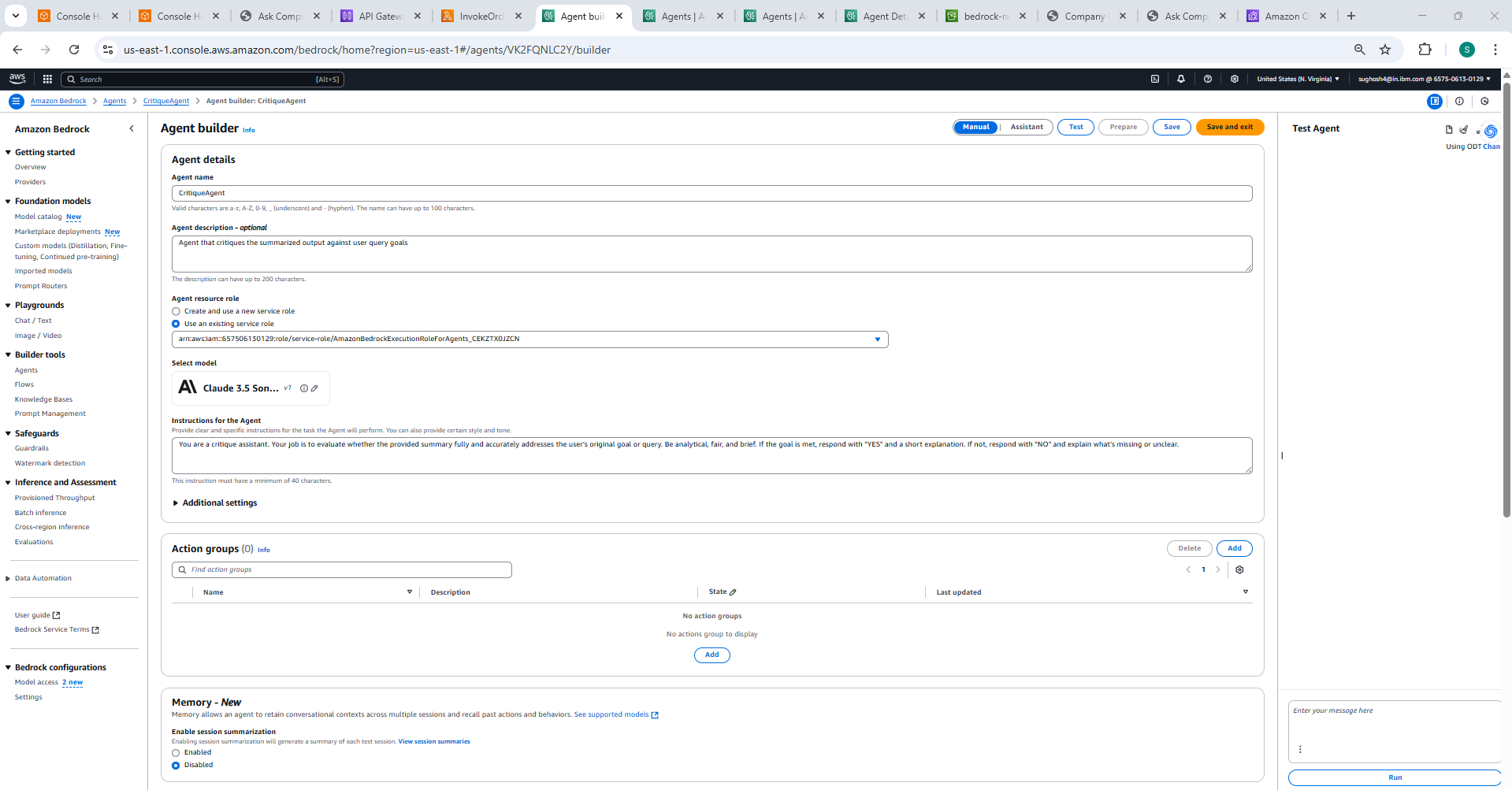
SummarizerAgent

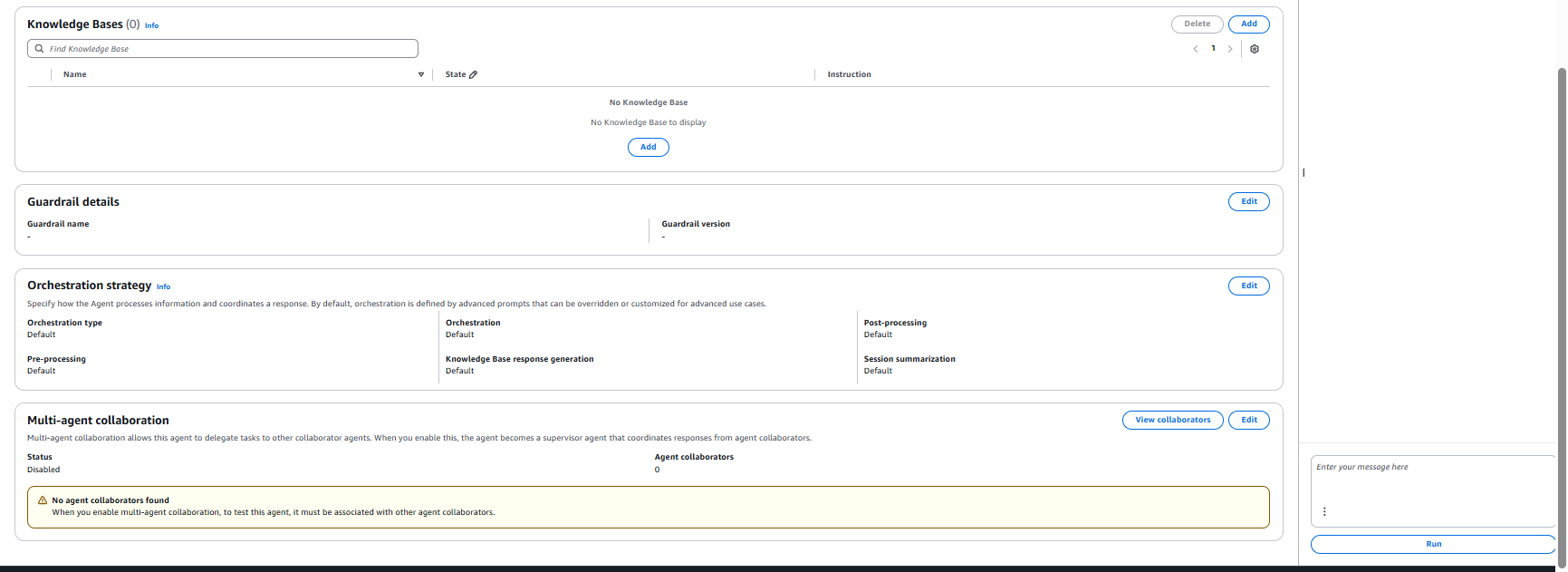






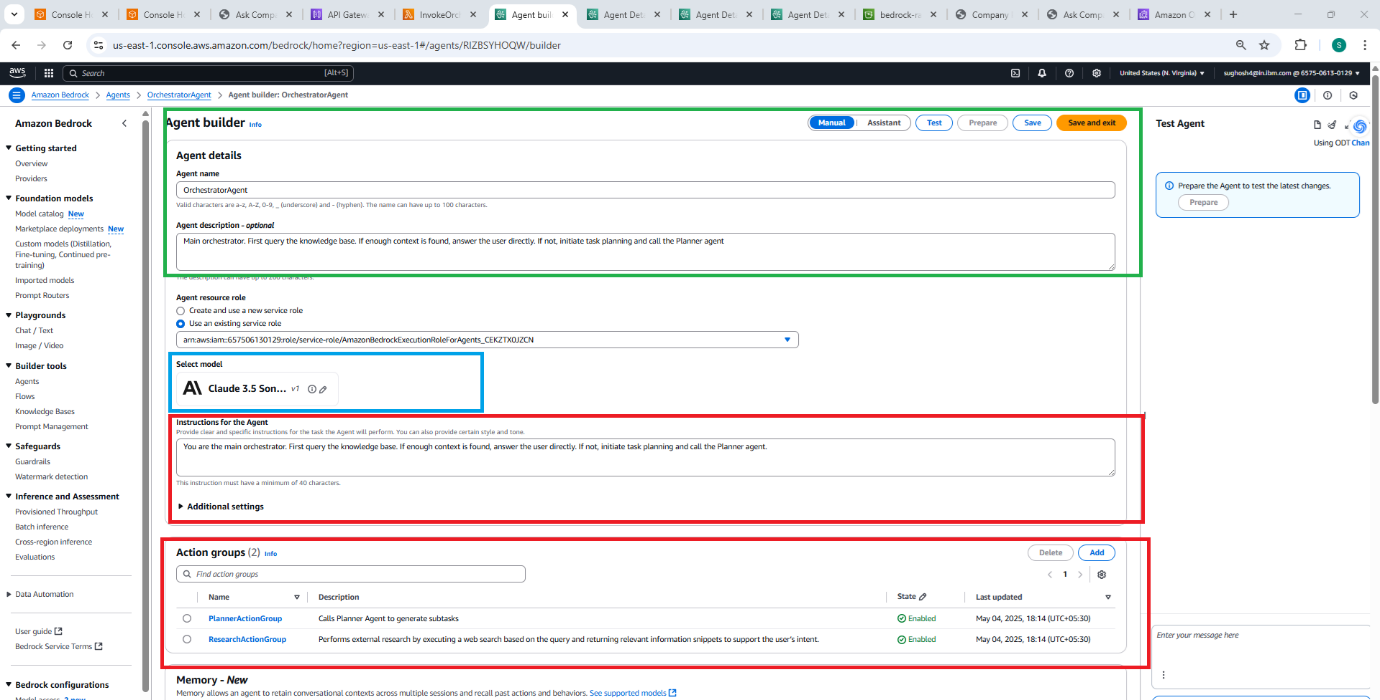
Critique Agent

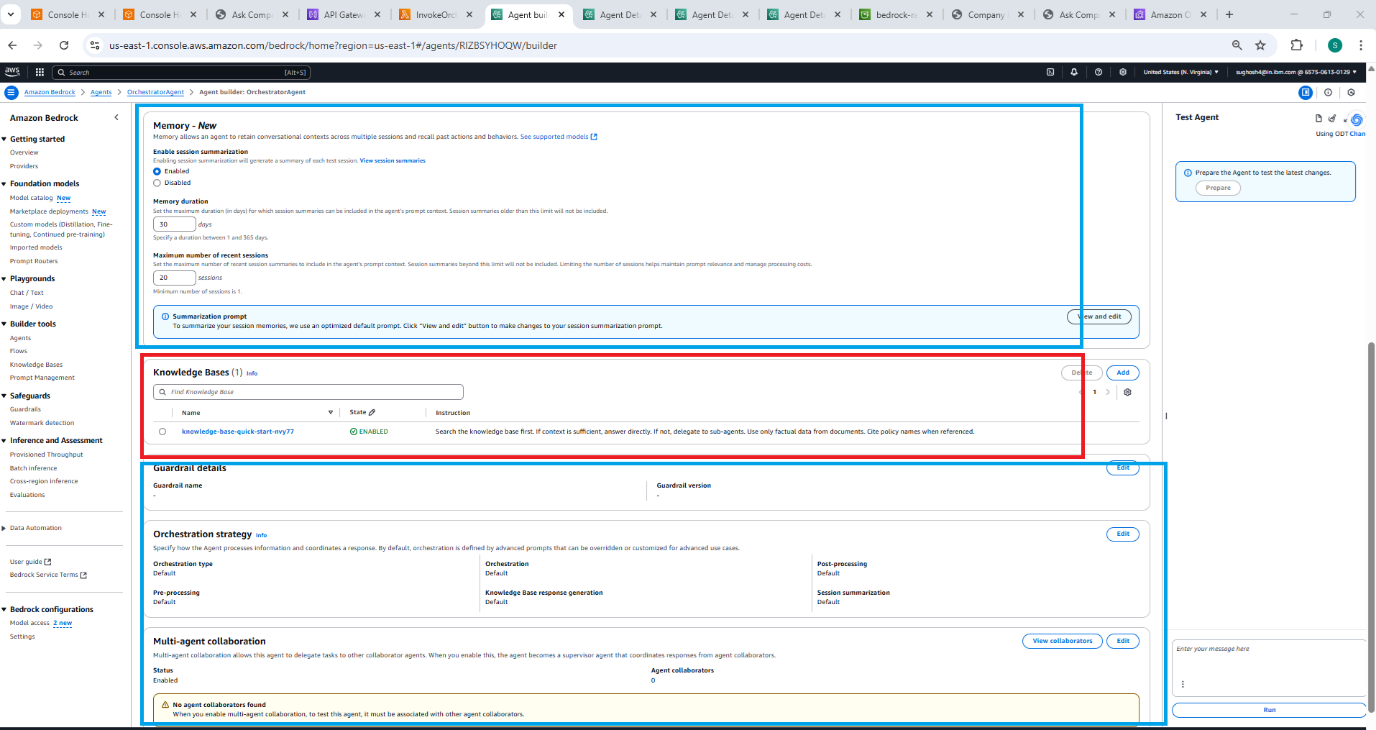




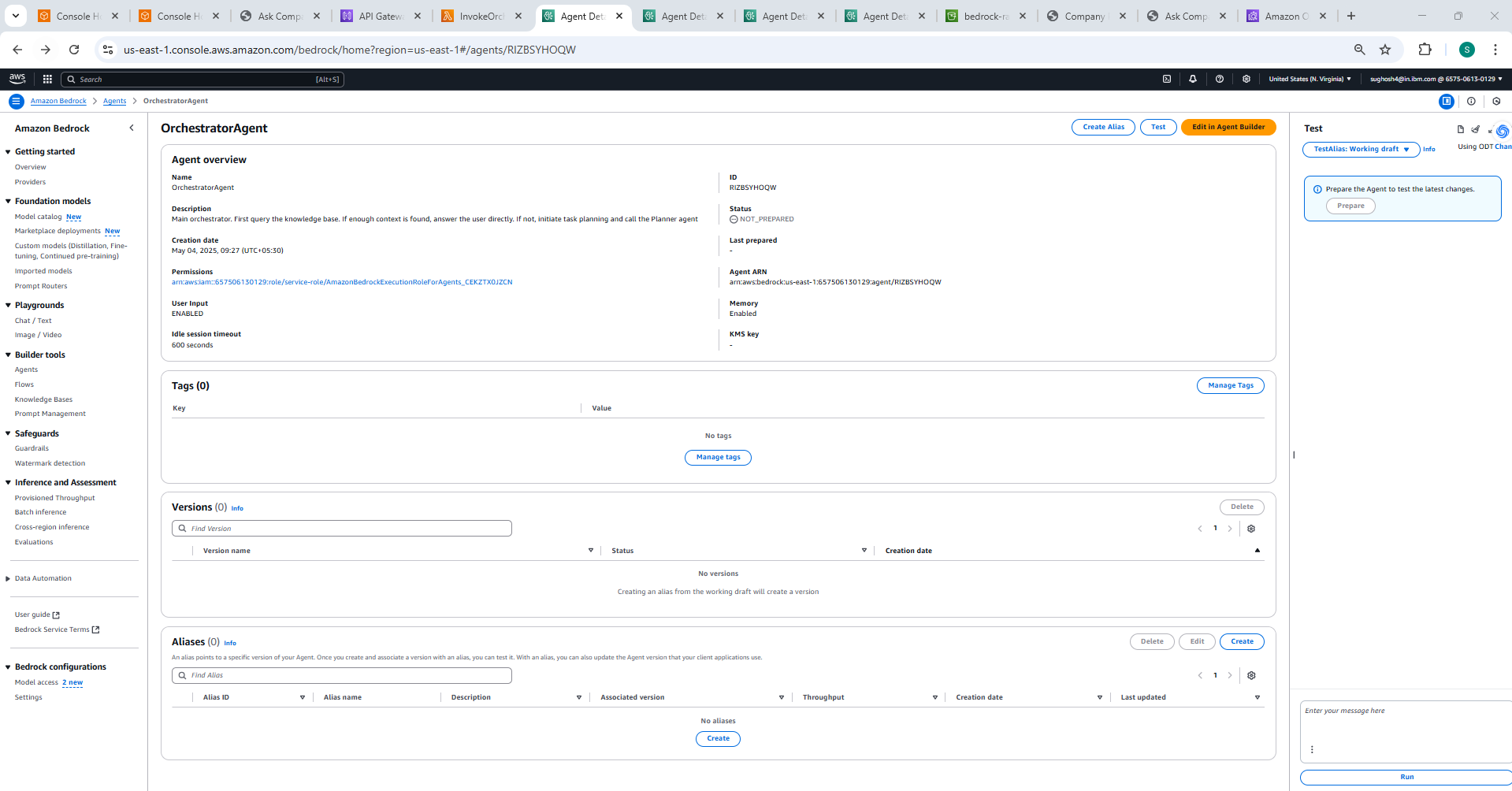
**OrchestratorAgent**

Here unlike other subagents you will note that there are action Groups defined because it interacts with other subagents , it also connects to the vector DB a d has a knowledge base (No other sub agents are allowed to access the Knowledgebase except the main orchestrator Agent).Furthermore , there is also instructions provided to OrchestratorAgent how it should search the information . i.e Search the Vector DB first , if enough context not found then delegate to sub agents. This is the main task master which makes all other subagents work.





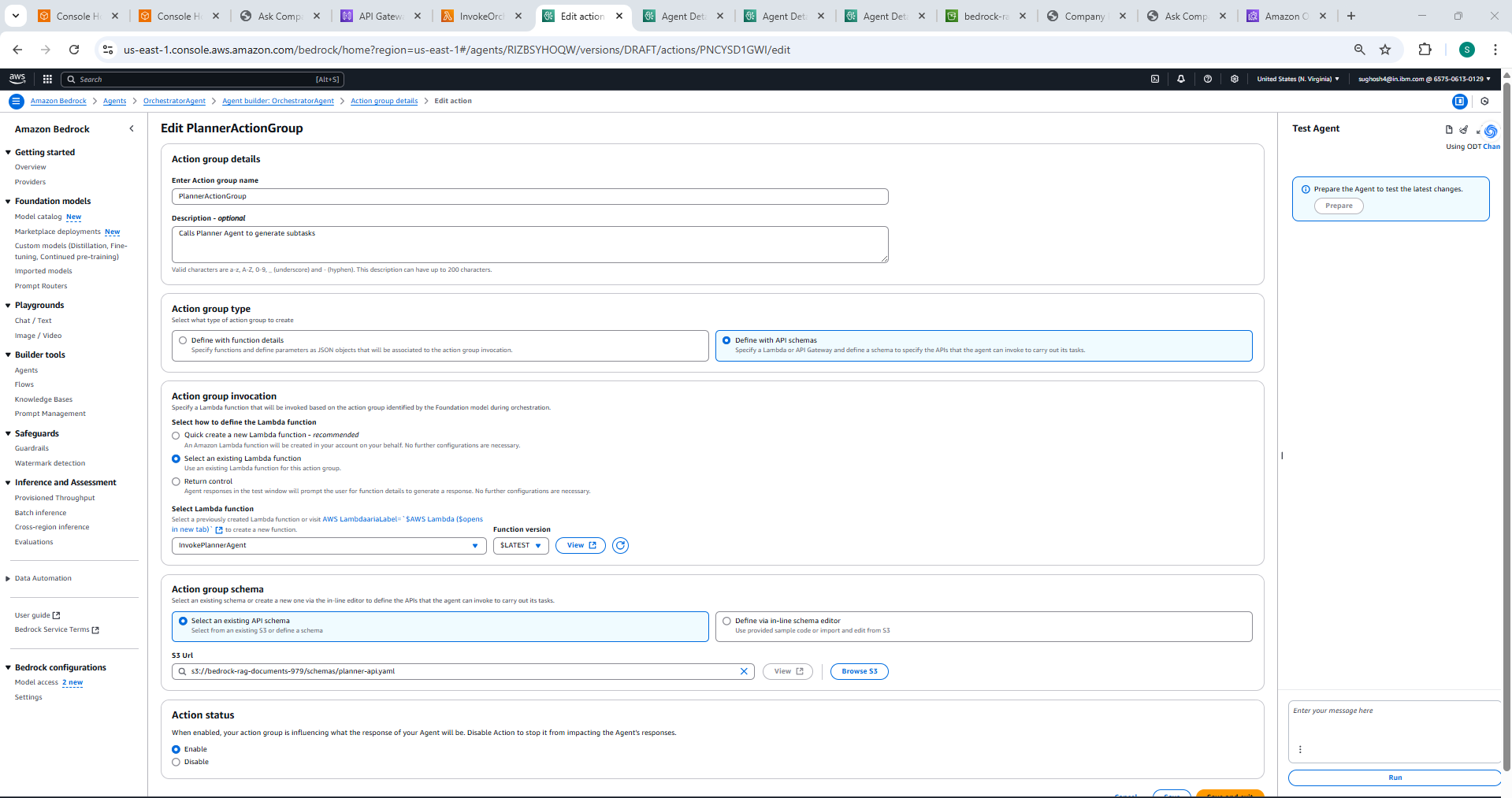




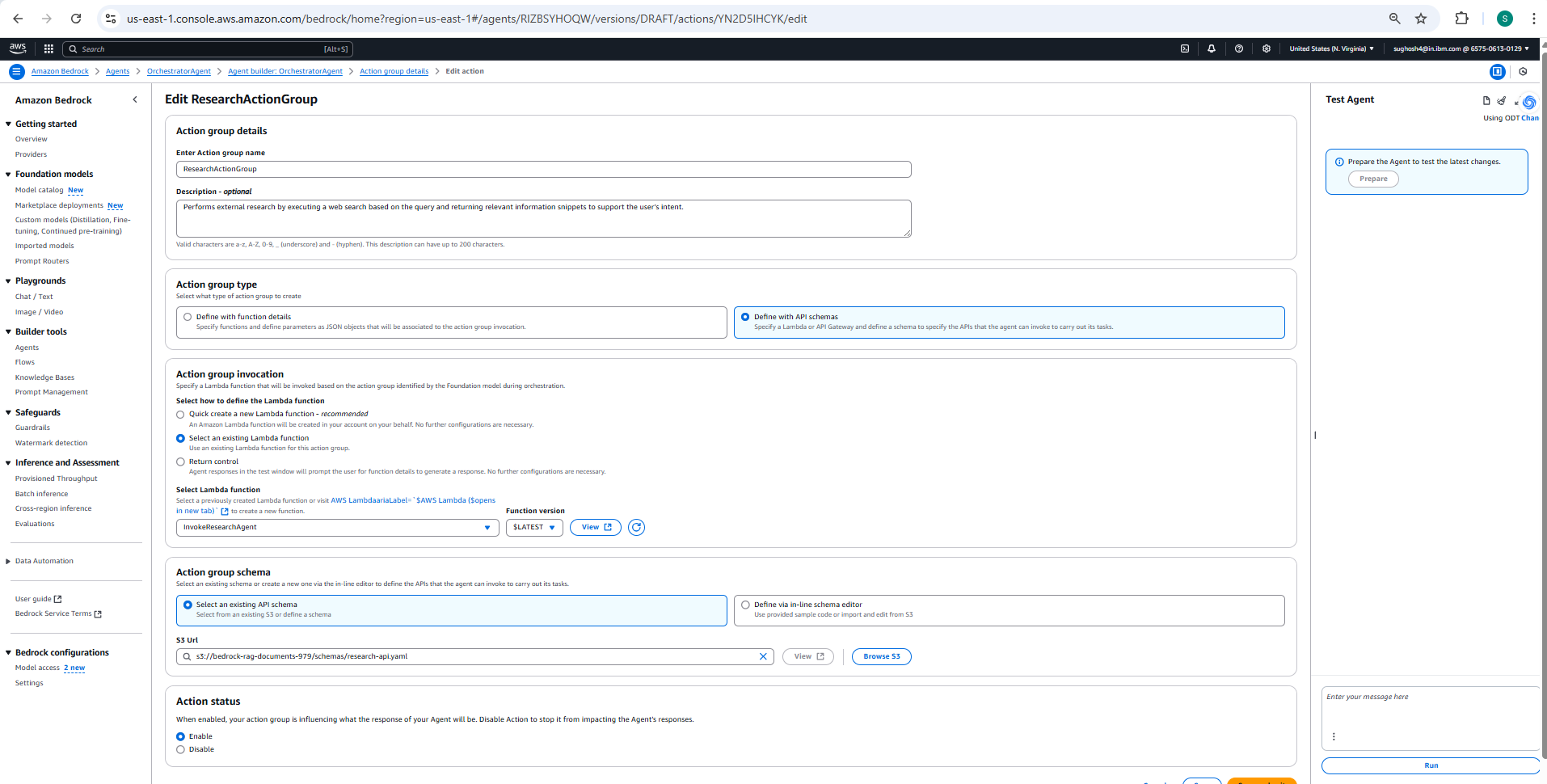
**Step4: Define Action Groups** or **Function Calls** to let the Orchestrator invoke other sub agents.

Since , Orchestrator Agent is going to call as subagents , create the subagents PlannerAgent and ResearchAgent as above and Define action groups as below.

PlannerAction Group



ResearchAction Group



An **Action Group** enables an agent to call a backend service or Lambda function to complete a task. In this case, we’ll define one that the Orchestrator can use to invoke sub-agents like the Planner, or to handle fallback workflows.

**Step5:** Create lambda Functions for OrchestratorAgent to trigger the Subagents for Agent Orchestration. You would need to create lambda function for each agent as well, which will receive the orchestration call from Orchestrator and act as per its role.

Create lamda functions for OrchestratorAgent and for each Subagent with following names(InvokePlannerAgent,InvokeResearchAgent,InvokeSummarizerAgent,InvokeCritiqueAgent).See screenshot for OrchestratorAgent. The lambda function code for each agent is uploaded in Github.

Take respective lambda function codes for each Agent , paste in code section and deploy.

InvokeOrchestratorAgent

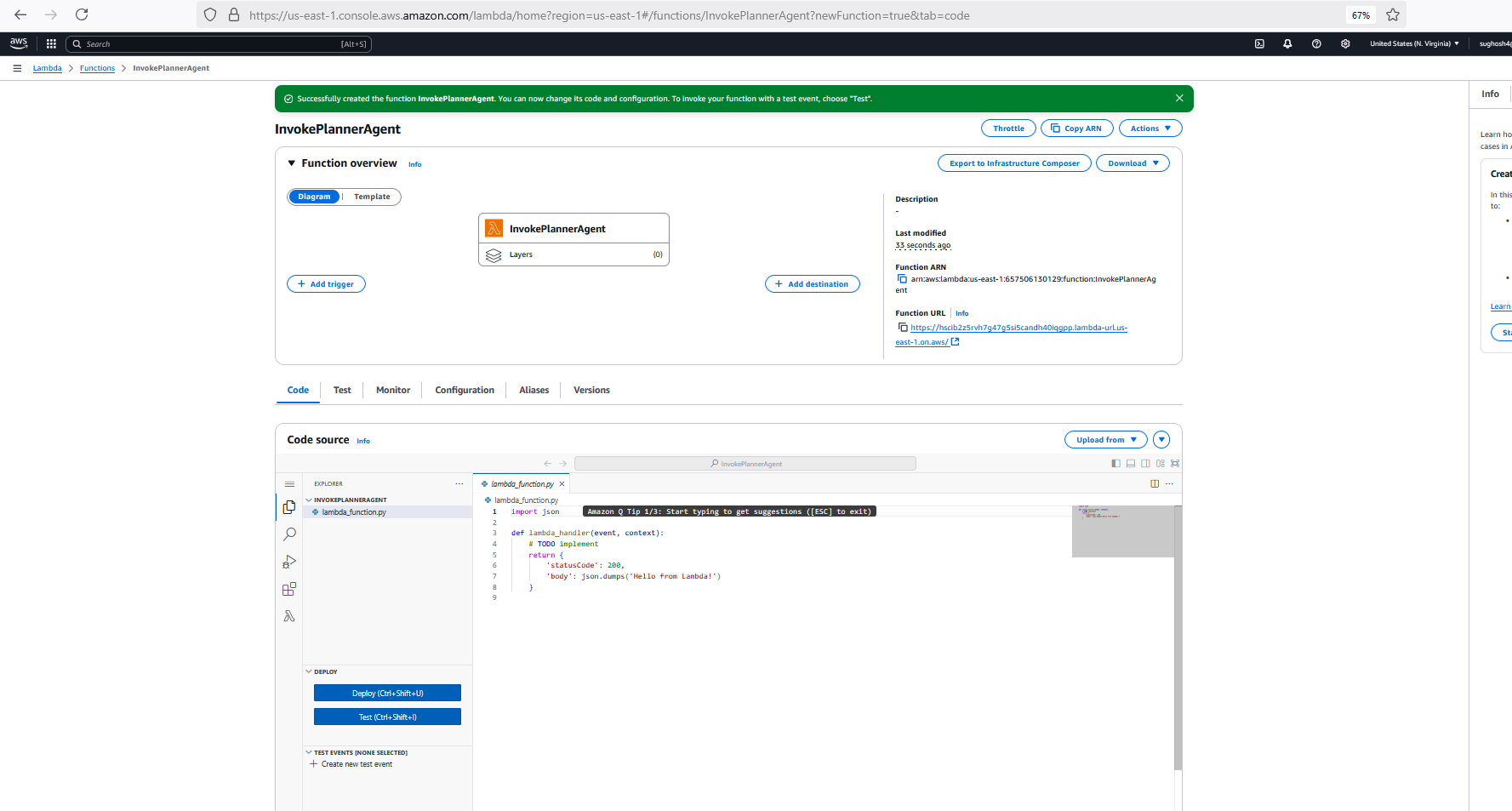


Field Values during Lambda Function Creation.See inputs for Planner for reference



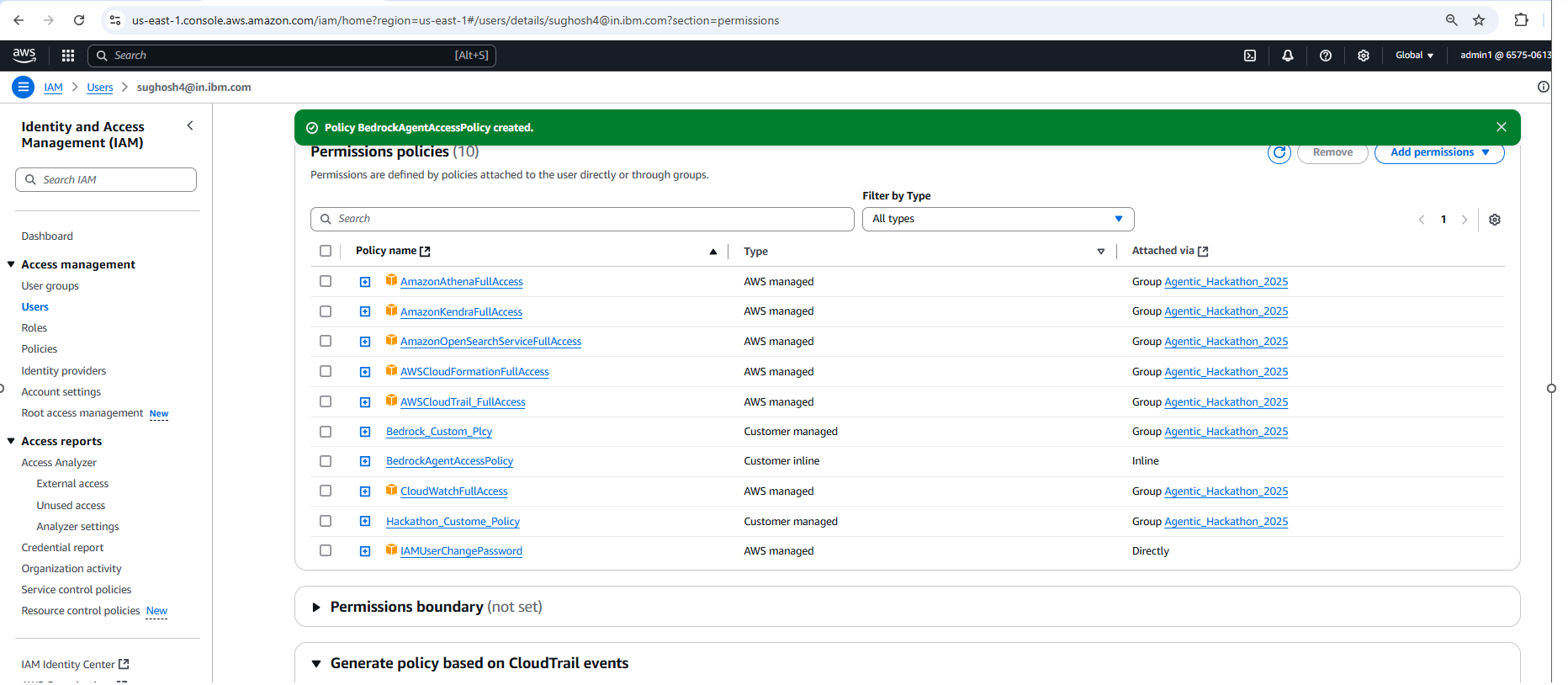


On Clicking create function it will get created.



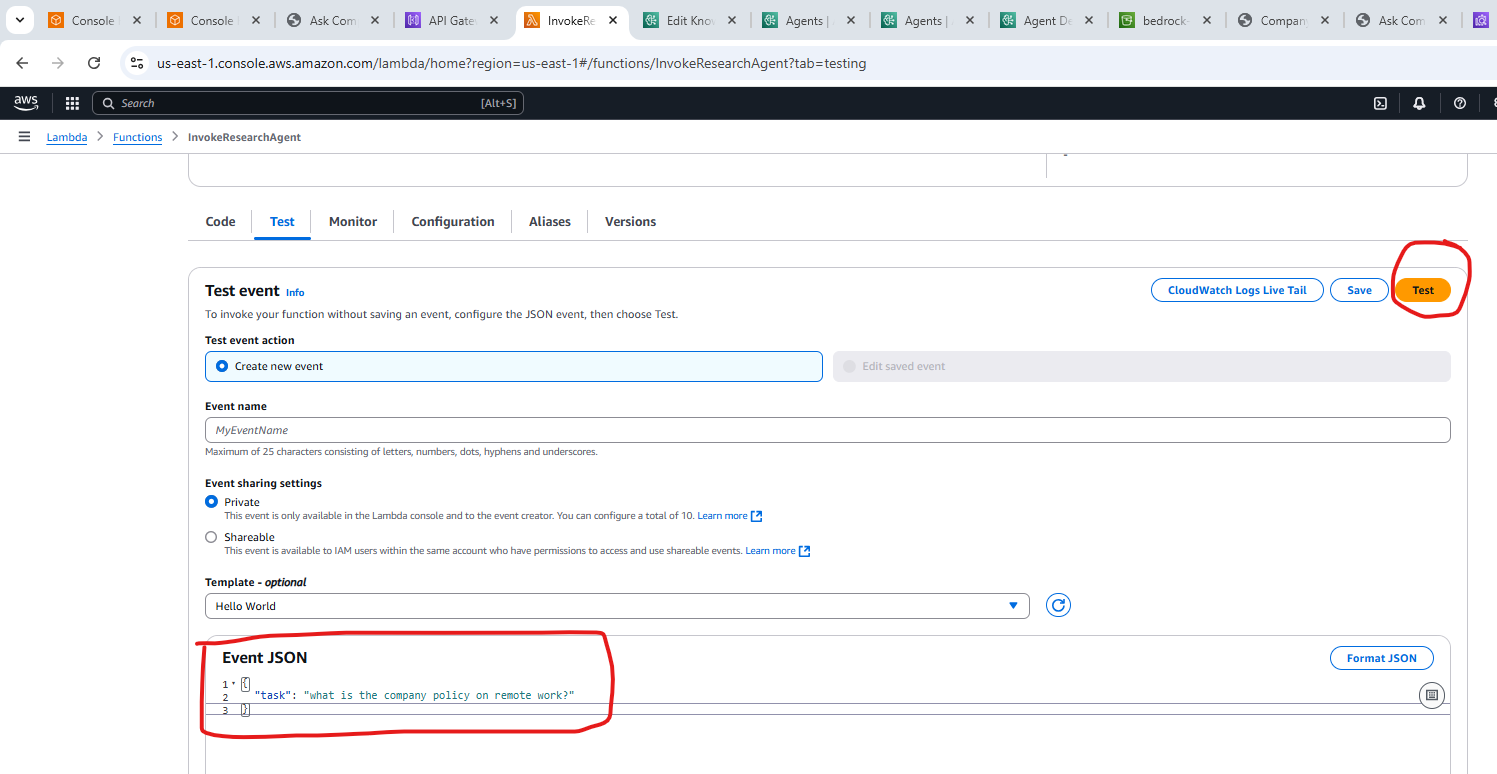
Step 5: Add all necessary permissions for Bedrock agents else the orchestration calls won’t work.

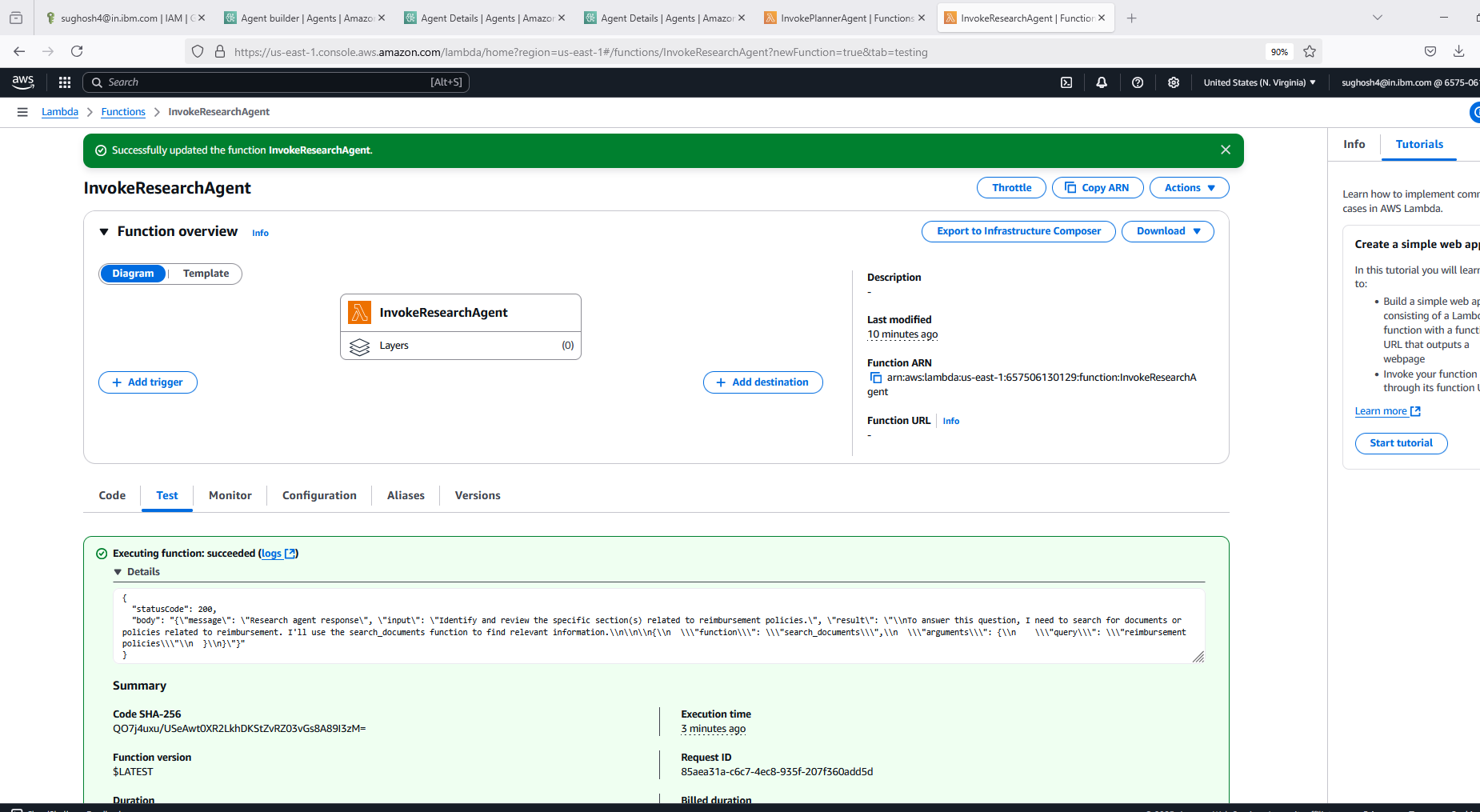
See embedded document(IAM ROLE.docx) here with all configs for each above Policies and apply 



Step6:

Test the lambda functions individually, bottom up , from critique to orchestrator . Go to Test tab under lambda function and Test it. Lambda Test Prompts as below.





**Test Prompts:**

**✅ 1. Orchestrator Agent**

{

"body": "{\"query\": \"What is the company policy on remote work?\"}"

}

**📋 2. Planner Agent**

{

"query": "What is the company policy on remote work?"

}

**🔍 3. Research Agent**

{

"task": "what is the company policy on remote work?"

}

**📝 4. Summarizer Agent**

{

"snippets": [

"Remote work is allowed for up to 3 days per week.",

"Equipment and internet stipends are available upon request.",

"Company maintains a hybrid model for work."

]

}

**🧠 5. Critique Agent**

{

"summary": "The company allows employees to work remotely up to 3 days a week and provides equipment and internet support for remote work.",

"goal": "What is the company policy on remote work?"

}

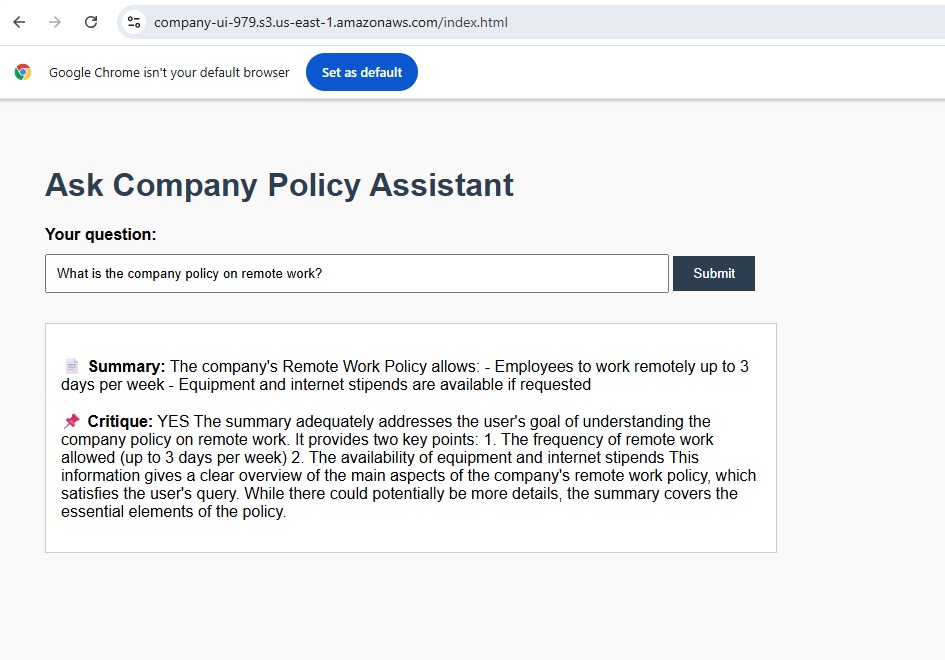
**Step 7: Create the API Gateway**

The API Gateway sits between the User Chatbot UI and the orchestrator Agent. Its passes down the User query into the Orchestrator which in turn engages the subagents.Below shows the connection pipeline . Chatbot UI calls the Orchestrator Lambda through API gateway, which gets the answer from the Agent group and gives it back to the User Chatbot UI.

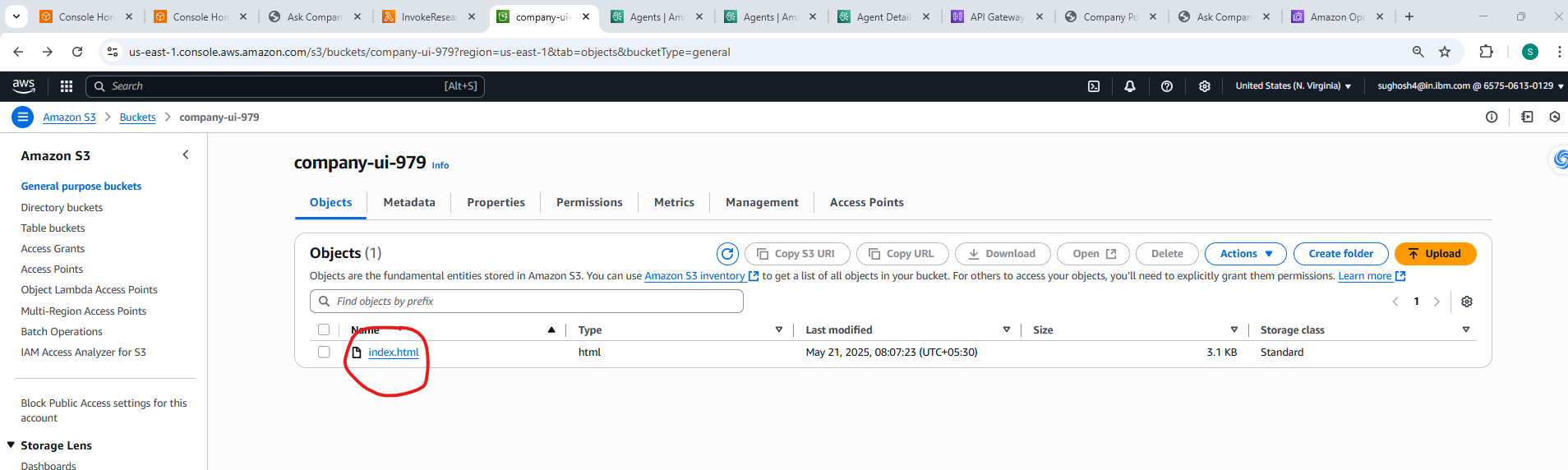


Step 8: Create the Chatbot UI.

Create a simple page as below and host it on an S3 bucket

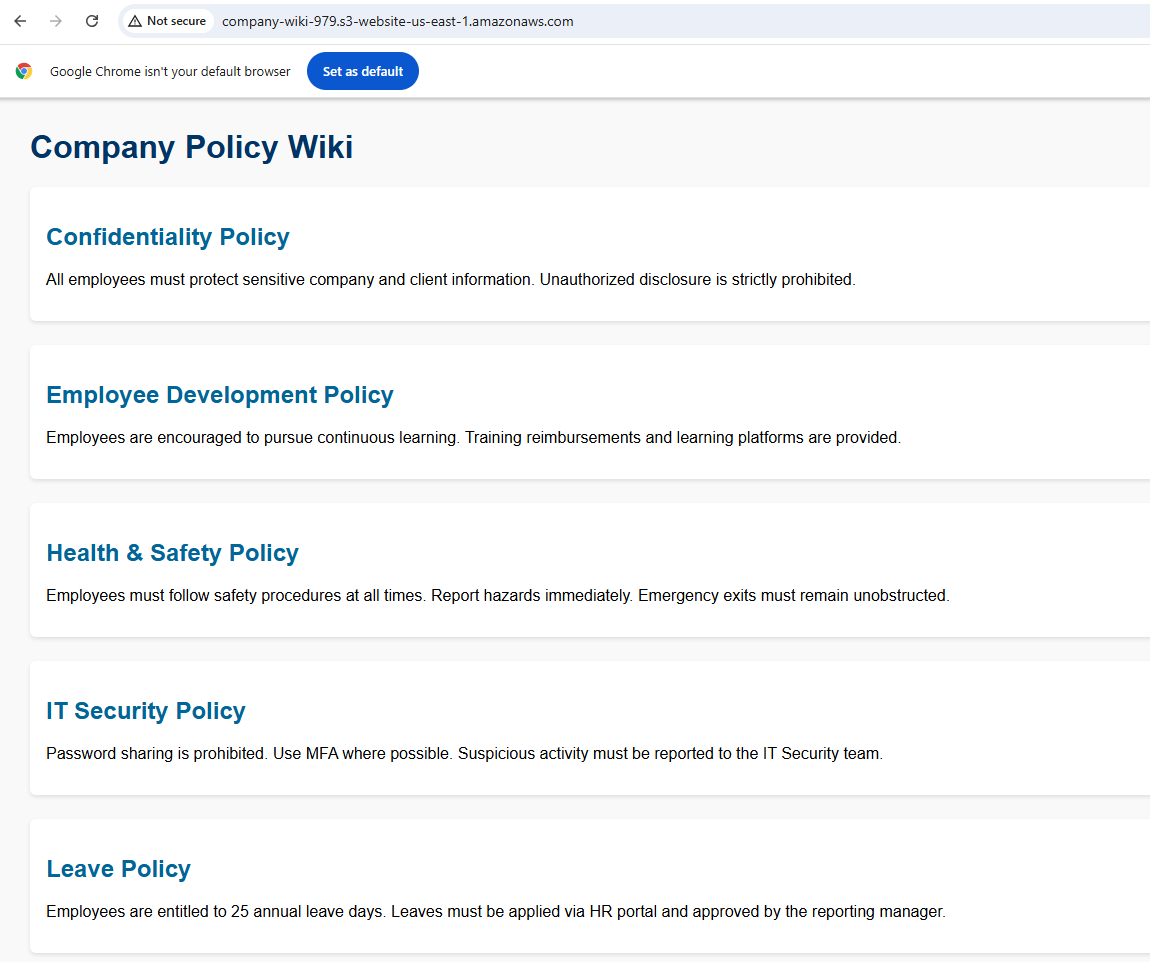


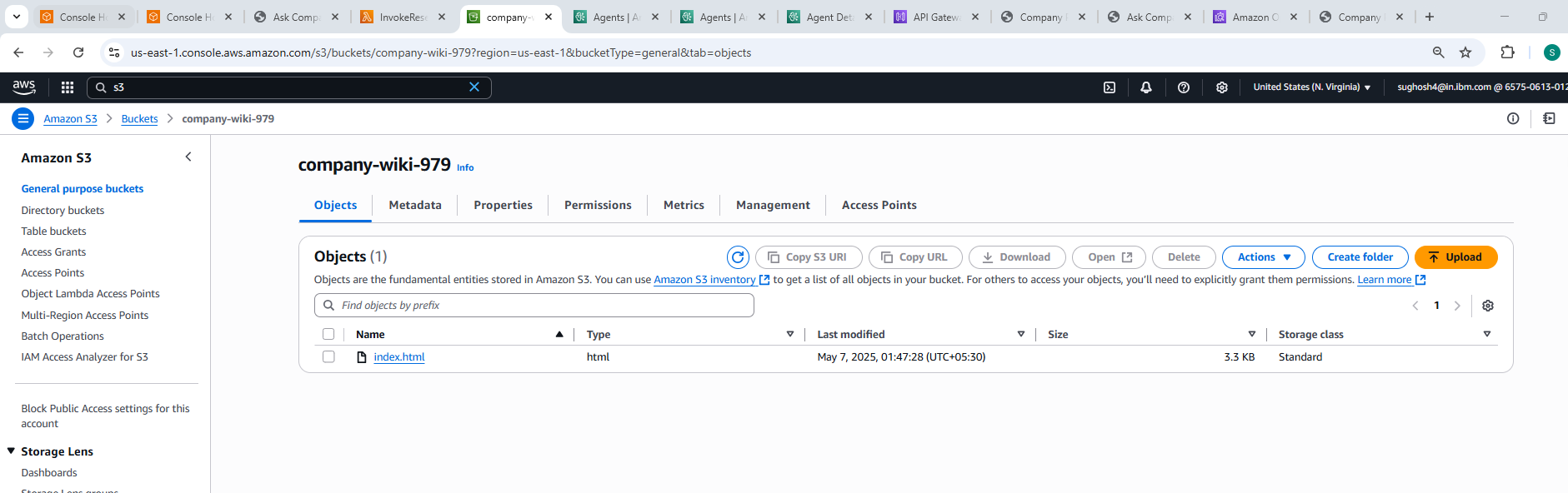
UI Hosted on S3 Bucket[Code available in Github]



**Step 9: Create a dummy Company Wiki as well for Research agent to make a tool call.**

Static Website host on S3 bucket

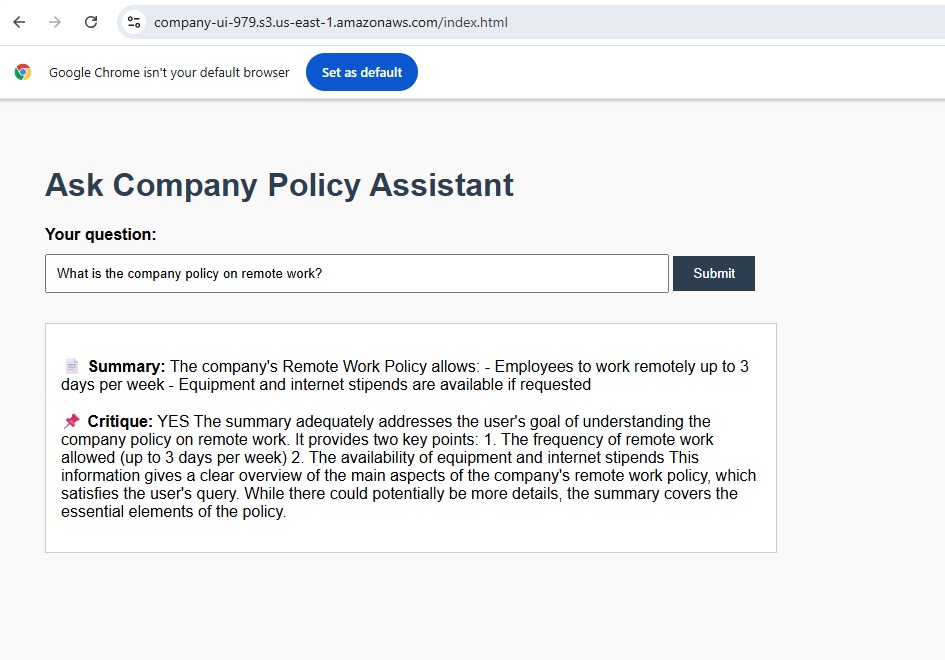




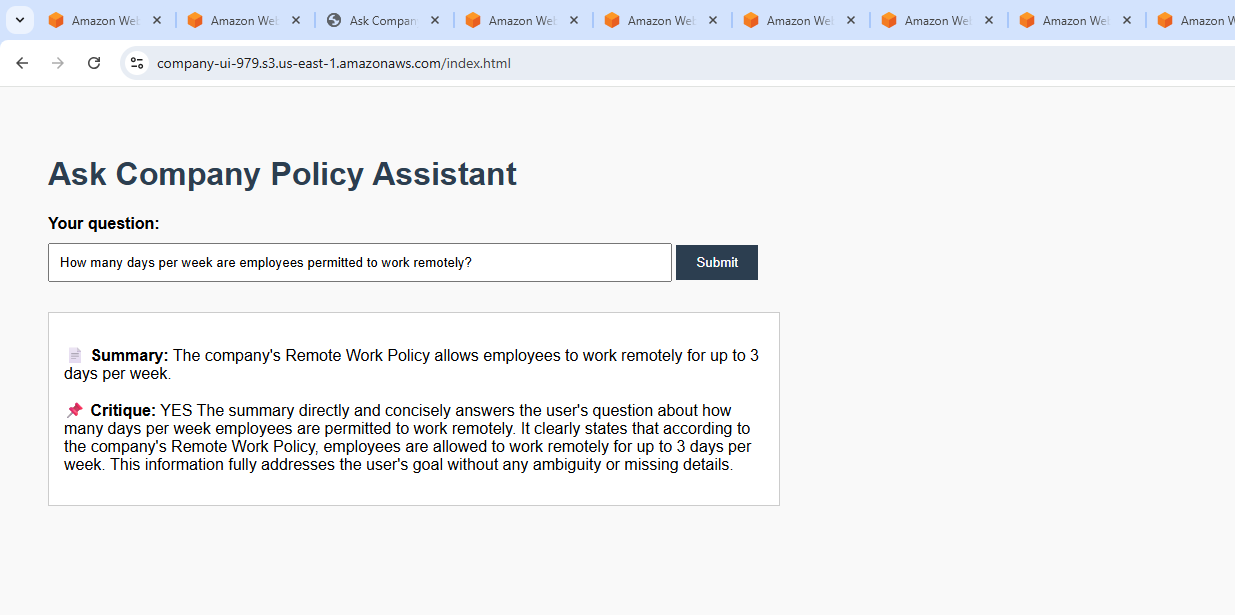
You are now ready to Test your implementation from UI. User Types Query in ChatBotUI. Chatbot UI passes it to OrchestratorAgent through the API Gateway. Orchestrator searches the Vector DB and returns answer if it finds enough context if not it invokes the subagents to find an appropriate answer and returns back to User.

Sample Test User Queries and answers from the Agents

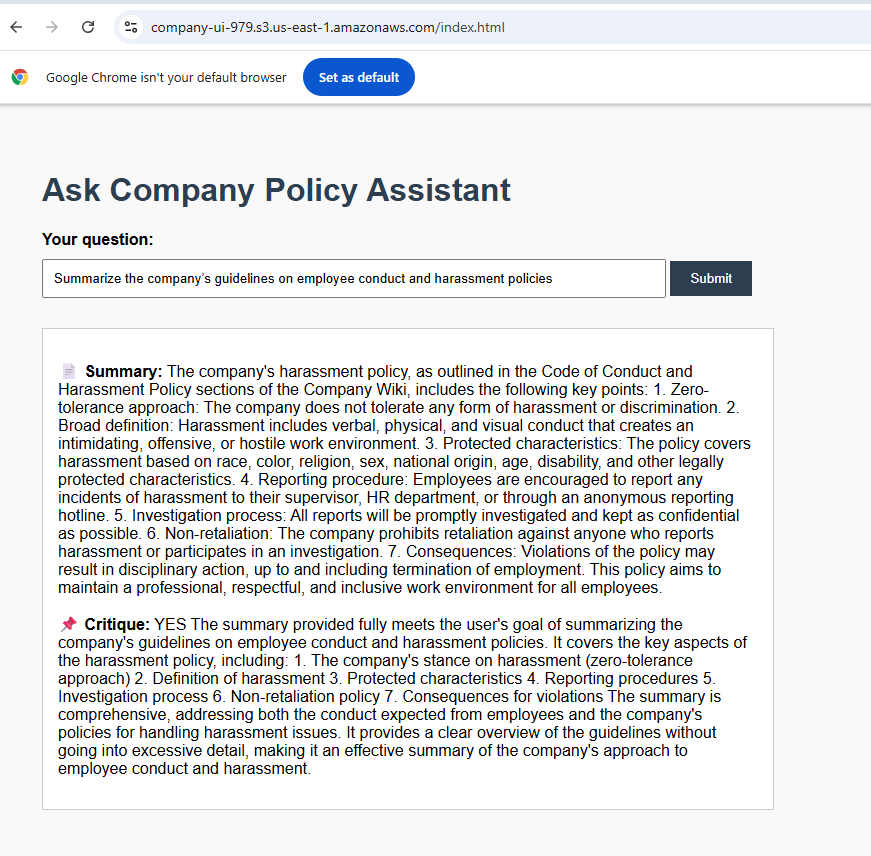
**Sample Query1:**



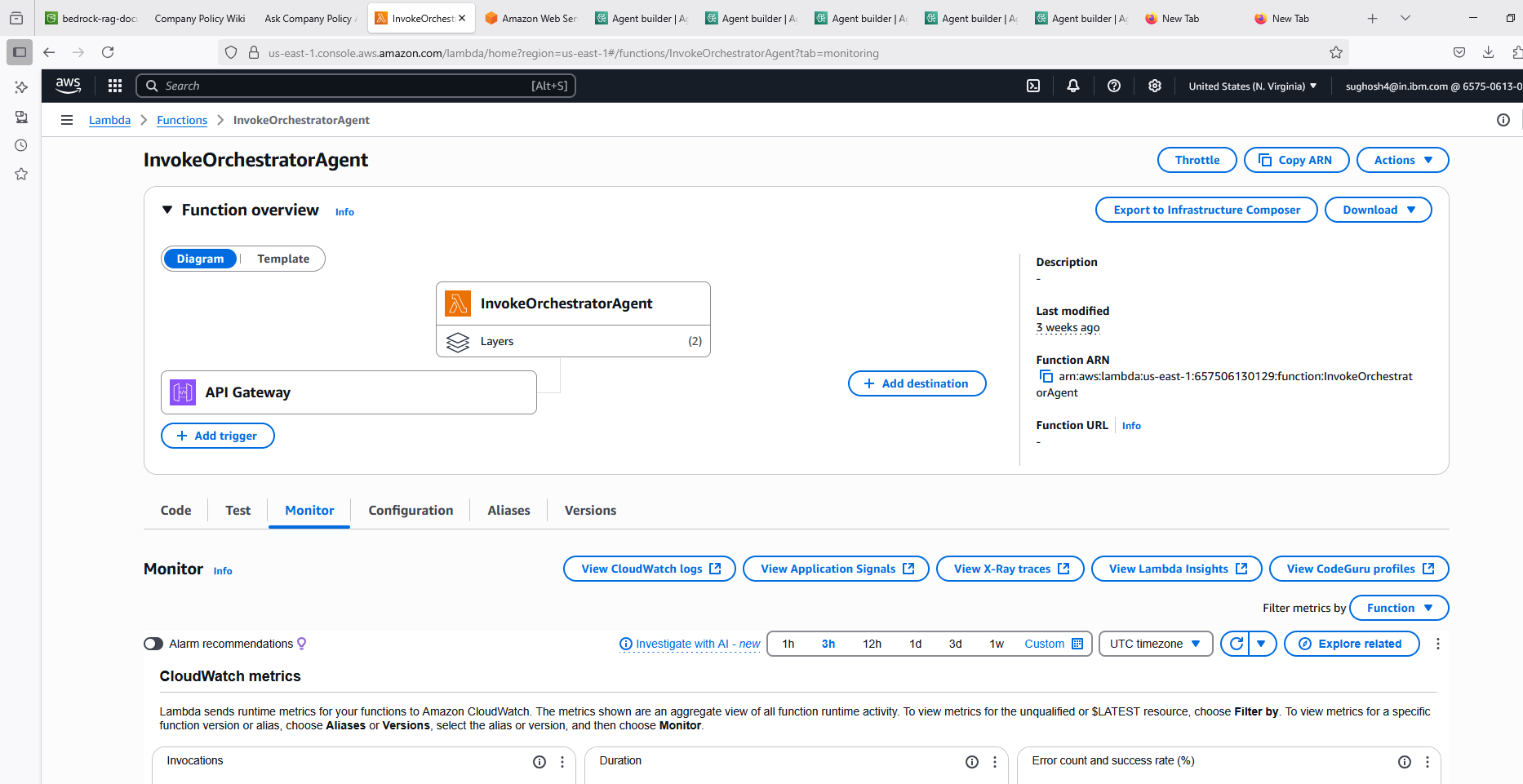
**Sample Query 2:**

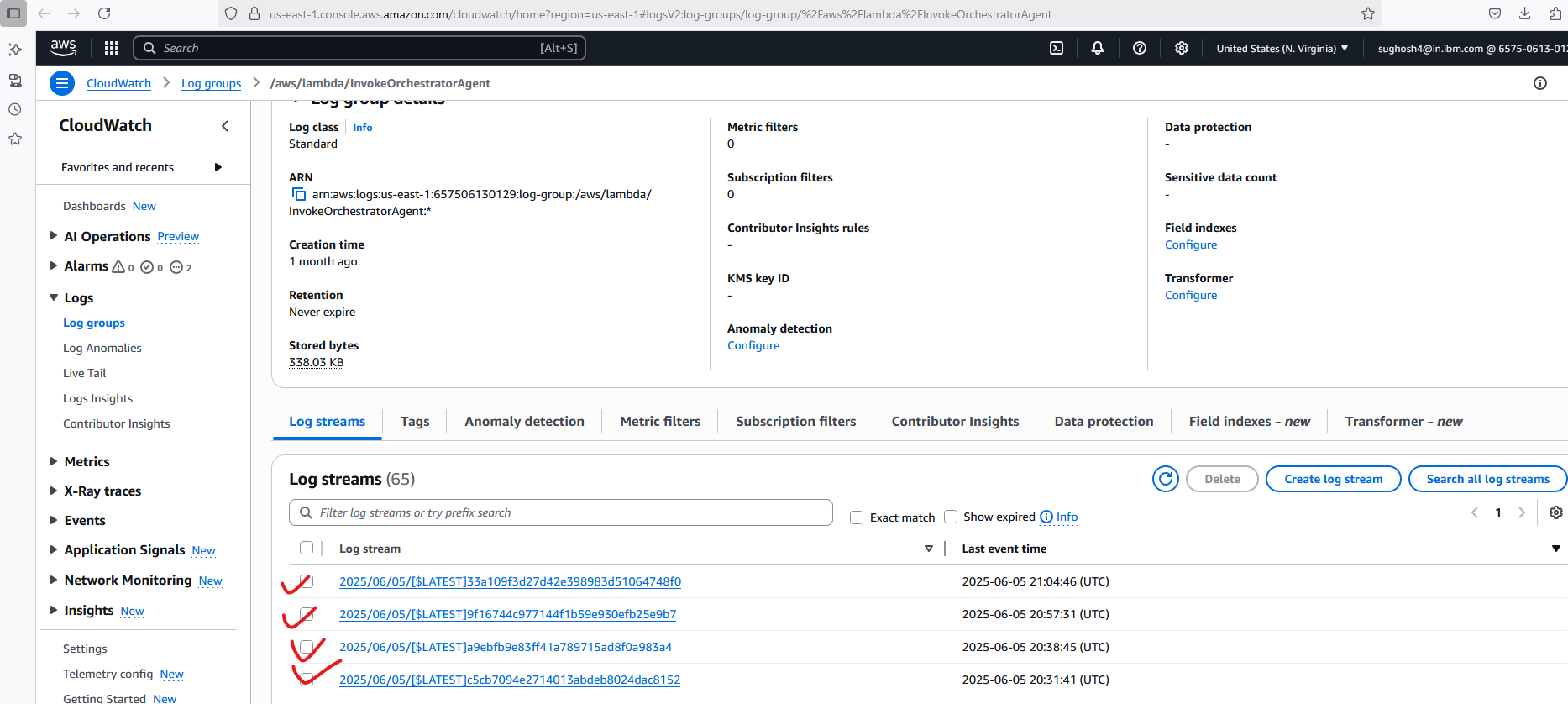
****

**Sample Query 3:**



You can watch Detailed Orchestration in the Cloud Watch Logs here by going to InvokeOrchestratorAgent after you execute each query





Review this embedded document where the Agent orchestration log is explained

