Deployment: Autonomous Ocean Forecasting Agent (Bedrock Agents + Lambda + S3)

Region note: Use a Bedrock Agents supported region (recommended: us-east-1). Ensure the chosen model (Nova Pro / Claude Sonnet) is available in your region.

0) Prereqs

- AWS CLI v2 configured with admin/deployer credentials
- AWS SAM CLI
- Python 3.12
- Bedrock access enabled for your account (console: Bedrock → Model access)

1) Create S3 bucket (unique)

Choose a unique bucket name (e.g. `ocean-forecast-data-<username>`). This stack expects a name you pass in at deploy.

2) Deploy core stack (S3, Ingest Lambda, Agent Gateway Lambda + API)

PowerShell (Windows):

```
$REGION = "us-east-1"
$BUCKET = "ocean-agent-data-<your-unique-suffix>"
cd "c:\Users\mubva\Downloads\AI agent aws"
sam build --template-file .\infra\template-agentcore.yaml
sam deploy --stack-name ocean-agentcore --resolve-s3 --capabilities
CAPABILITY_IAM CAPABILITY_NAMED_IAM `
  --parameter-overrides ProjectName=ocean-agent BucketName=$BUCKET
Region=$REGION
```

Outputs include:

- DataBucketName
- IngestFunctionName
- ApiInvokeUrl (append `/query`)

3) (Optional) Create Copernicus secret

If you have Copernicus Marine credentials, store them in Secrets Manager as JSON.

```
aws secretsmanager create-secret `
  --name copernicus/credentials `
```

```
  --secret-string '{"username":"<user>","password":"<pass>"}' `
  --region $REGION
```

The ingest Lambda automatically detects this secret and will mark `copernicus: configured` (extend handler to fetch datasets).

4) Test ingest Lambda

```
$INGEST=$(aws cloudformation describe-stacks --stack-name ocean-agentcore --
query "Stacks[0].Outputs[?OutputKey=='IngestFunctionName'].OutputValue" --
output text)
# If your AWS CLI is older and doesn't support --cli-binary-format, write a
file:
Set-Content -Path payload.json -Value '{"latitude": -33.9249, "longitude":
18.4241, "forecast_hours": 48}'
aws lambda invoke --function-name $INGEST --payload fileb://payload.json
out.json --region $REGION
Get-Content out.json
```

You should see `{ bucket, key, copernicus }`. Verify S3 object exists.

5) Create Bedrock Agent (console or CLI)

Console path: Amazon Bedrock → Agents → Create agent.

- Agent name: OceanForecastAgent
- Foundation model: Choose a reasoning-capable model (e.g., Amazon Nova Pro or Anthropic Claude Sonnet) available in your region
- Instructions: paste your system prompt (see everything.md)
- Memory: enabled (optional)
- Create Agent → then Create Alias (e.g., `prod`)

CLI (reference):

```
# Create agent (minimal example; adjust model + instructions)
$AGENT_JSON = @'{
  "agentName": "OceanForecastAgent",
  "description": "Autonomous ocean forecasting",
  "instruction": "You are a maritime safety expert...",
  "foundationModel": "amazon.nova-pro-v1:0"
}'@
aws bedrock-agent create-agent --agent $AGENT_JSON --region $REGION | Set-
Content agent-create.json

# Create alias
$AGENT_ID = (Get-Content agent-create.json | ConvertFrom-Json).agentId
aws bedrock-agent create-agent-alias --agent-id $AGENT_ID --agent-alias-name
```

```
prod --region $REGION | Set-Content agent-alias.json
$ALIAS_ID = (Get-Content agent-alias.json | ConvertFrom-Json).agentAliasId
```

> If your region doesn't offer the model, pick an available one from Bedrock Model Access.

Helper script (boto3):

```
$ROLE_ARN = (aws cloudformation describe-stacks --stack-name ocean-agentcore --
query "Stacks[0].Outputs[?OutputKey=='AgentExecutionRoleArn'].OutputValue" --
output text)
python .\scripts\bedrock_agent_setup.py --role-arn $ROLE_ARN --instructions-
file .\everything.md --region $REGION `
  --name OceanForecastAgent --alias prod --model amazon.nova-pro-v1:0 | Tee-
Object agent-out.json
$AGENT_ID = (Get-Content agent-out.json | ConvertFrom-Json).agentId
$ALIAS_ID = (Get-Content agent-out.json | ConvertFrom-Json).agentAliasId
```

6) Create Action Group for `fetch_ocean_data`

In the Bedrock Agent console, add an Action group:

- Name: `oceanTools`
- Lambda: select the deployed `ocean-agent-ingest` function
- API Schema: upload `schemas/action-groups/fetch_ocean_data.json`
- Save and apply

7) Wire Agent IDs into API Gateway Lambda

Update the stack with AgentId and AgentAliasId so the `/query` endpoint can invoke your agent.

```
$AGENT_ID = "<your-agent-id>"
$ALIAS_ID = "<your-alias-id>"
cd "c:\Users\mubva\Downloads\AI agent aws"
sam deploy --stack-name ocean-agentcore --resolve-s3 --capabilities
CAPABILITY_IAM CAPABILITY_NAMED_IAM `
  --parameter-overrides ProjectName=ocean-agent BucketName=$BUCKET
Region=$REGION AgentId=$AGENT_ID AgentAliasId=$ALIAS_ID
```

8) Call the public API

```
$API=$(aws cloudformation describe-stacks --stack-name ocean-agentcore --query
"Stacks[0].Outputs[?OutputKey=='ApiInvokeUrl'].OutputValue" --output text)
Invoke-WebRequest -Method POST -Uri "$API/query" -ContentType application/json
-Body '{"query":"Is it safe to sail from Cape Town to Mossel Bay
tomorrow?","session_id":"demo-001"}' | Select-Object -ExpandProperty Content
```

9) Optional: Local Web UI (Streamlit)

Run a simple UI to query the API without writing code.

```
cd "c:\Users\mubva\Downloads\AI agent aws\streamlit_app"
# (Optional) activate venv, then install deps
pip install -r requirements.txt

# Set your API base URL for the app (no trailing slash)
$env:OCEAN_API = (aws cloudformation describe-stacks --stack-name ocean-
agentcore --query "Stacks[0].Outputs[?OutputKey=='ApiInvokeUrl'].OutputValue" --
--output text)

python -m streamlit run app.py
```

The app sends server-side requests (no browser CORS hassles) and shows the raw JSON plus a friendly response view.

10) Observability

- CloudWatch Logs: `/aws/lambda/ocean-agent-ingest`, `/aws/lambda/ocean-agent-agent-gateway`
- X-Ray (enabled via Tracing: Active) — view traces if X-Ray is enabled in your account

11) Troubleshooting

- 403 or model not allowed: enable model access in Bedrock → Model access
- InvokeAgent fails: ensure Agent Alias exists and Lambda env has AGENT_ID and AGENT_ALIAS_ID
- CORS: For a browser-based client, front the API with CloudFront or add an OPTIONS method + CORS headers
- Copernicus: If secret missing, handler still writes Open-Meteo data and sets `copernicus: not_configured`