Xebia

# Building an Agentic Chat solution using Amazon Bedrock AgentCore

Joris Conijn
CTO for AWS @ Xebia

# Amazon Bedrock AgentCore (Preview)

Deploy and operate AI agents securely at scale - using any framework and model

Get started with the AgentCore toolkit | Access the AgentCore Console

## What is Amazon Bedrock AgentCore?

Amazon Bedrock AgentCore enables you to deploy and operate highly capable AI agents securely, at scale. It offers infrastructure purpose-built for dynamic agent workloads, powerful tools to


ONE WAY

# Bedrock AgentCore Starter Toolkit

## Deploy your local AI agent to Bedrock AgentCore with zero infrastructure

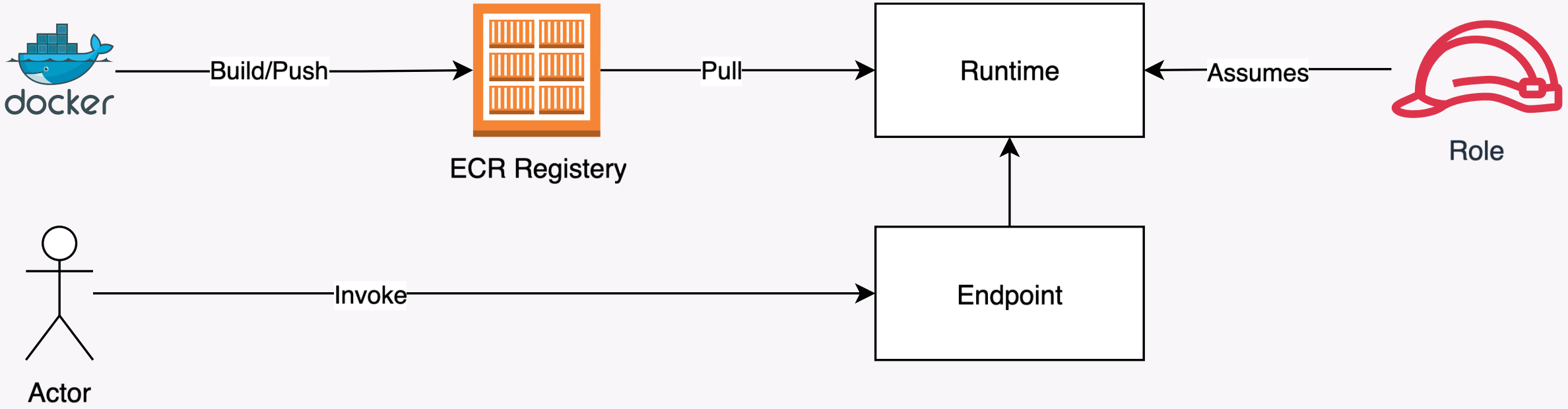

Documentation ◆ Samples ◆ Discord ◆ Boto3 Python SDK ◆ Runtime Python SDK ◆ Starter Toolkit

## Overview

Amazon Bedrock AgentCore enables you to deploy and operate highly effective agents securely, at scale using any framework and model. With Amazon Bedrock AgentCore, developers can accelerate AI agents into production with the scale, reliability, and security, critical to real-world deployment. AgentCore provides tools and capabilities to make agents more effective and capable, purpose-built infrastructure to securely scale agents, and controls to operate trustworthy agents. Amazon Bedrock AgentCore services are composable and work with popular open-source frameworks and any model, so you don't have to choose between open-source flexibility and enterprise-


Runtime
Memory
Gateway
Identity
Observability

# AgentCore Runtime

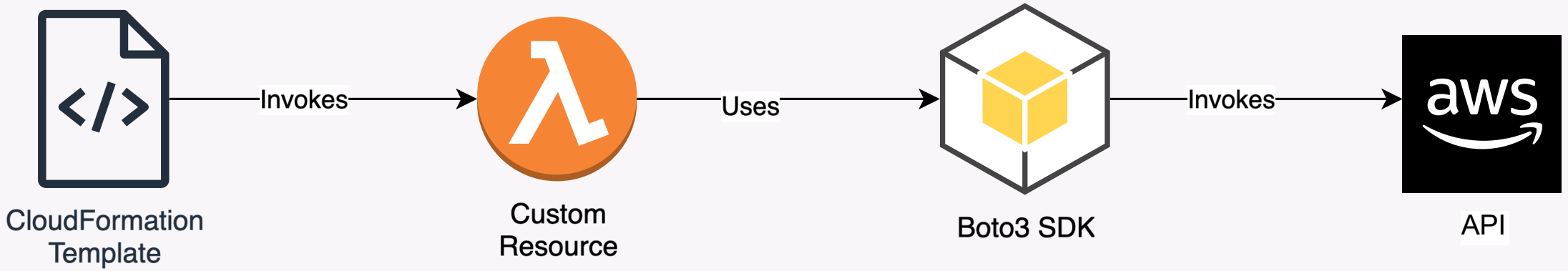


```
aws-opentelemetry-distro-genai-beta==0.1.6
aws-opentelemetry-distro==0.12.0
strands-agents==1.7.0
strands-agents-tools==0.2.6
bedrock-agentcore==0.1.2
pydantic>=2.11.7
```

```
@app.entrypoint
async def handler(payload, context):
    agent = Agent()
    user_input = payload.get("prompt")
    stream = agent.stream_async(user_input)
    async for event in stream:
        yield event
```

```
FROM public.ecr.aws/docker/library/python:3.12-slim
WORKDIR /app

COPY . .

# Install from requirements file
RUN python -m pip install --no-cache-dir -r requirements.txt

# Set AWS region environment variable
ENV AWS_REGION=eu-central-1
ENV AWS_DEFAULT_REGION=eu-central-1
ENV DOCKER_CONTAINER=1

# Create non-root user
RUN useradd -m -u 1000 bedrock_agentcore
USER bedrock_agentcore
EXPOSE 8080

CMD ["opentelemetry-instrument", "python", "-m", ”agent.main"]
```

# Custom Resources - CloudFormation

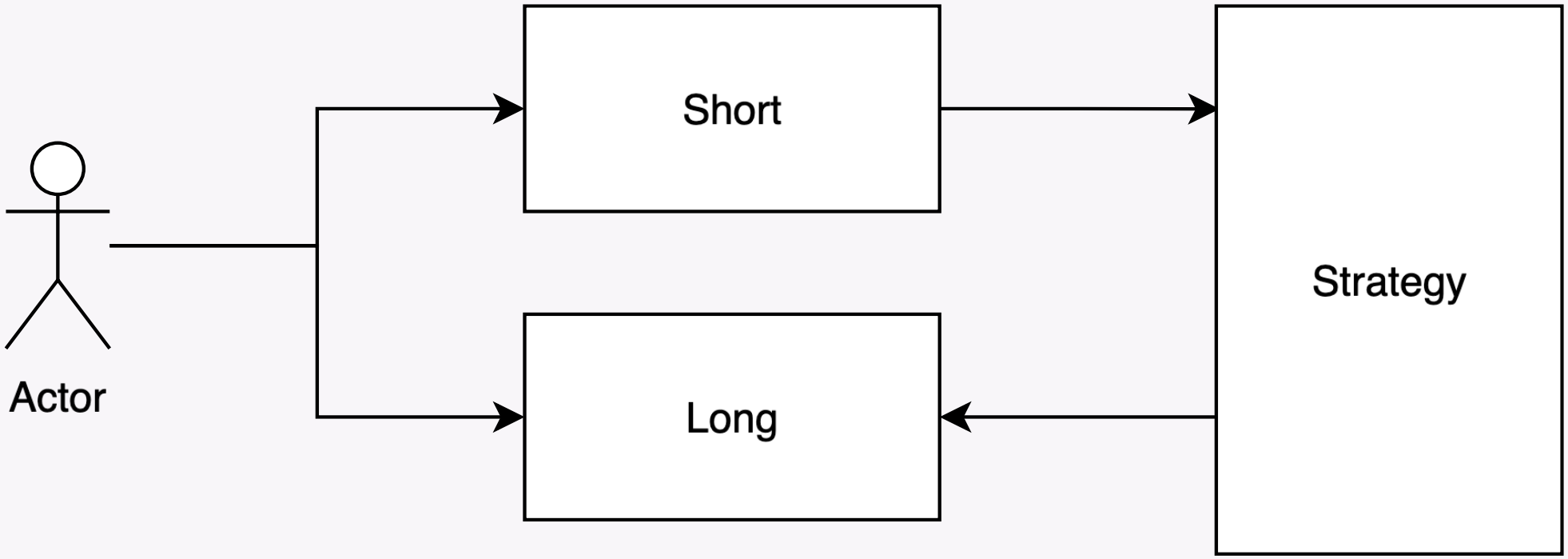


```
Build:
  Type: Custom::AgentCoreBuild
  Properties:
    ServiceToken: !Sub ${AgentCoreProvisioner.Arn}
    Repository: !GetAtt Repository.RepositoryUri
    ImageTag: latest
    Asset: s3://my-bucket/my-agent.zip
```

```
Runtime:
  Type: Custom::AgentCoreRuntime
  Properties:
    ServiceToken: !Sub ${AgentCoreProvisioner.Arn}
    Name: MyAgent
    ImageURI: ImageURI: !GetAtt Build.ImageUri
    RoleArn: !GetAtt Role.Arn
```

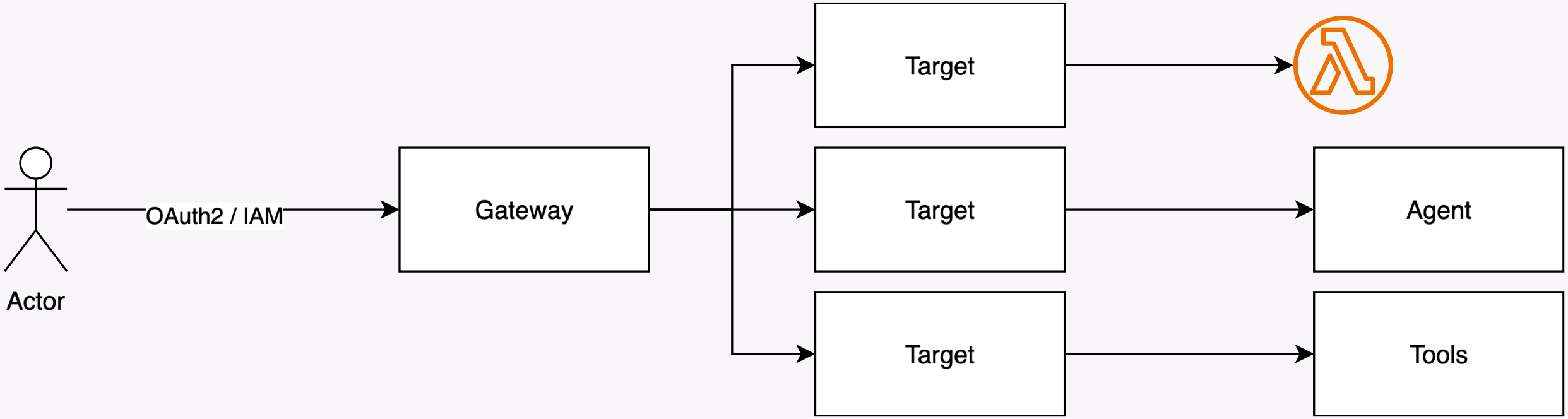
```
RuntimeLogGroup:
  Type: AWS::Logs::LogGroup
  Properties:
    LogGroupName: !Sub /aws/bedrock-agentcore/runtimes/${Runtime.Id}-DEFAULT
    RetentionInDays: !Ref RetentionInDays
```

# AgentCore Memory



```
Memory:
  Type: Custom::AgentCoreMemory
  Properties:
    ServiceToken: !Ref AgentCoreProvisioner
    Name: MyAgent
    Description: Memory for the main chat session.
    Expiration: 90
```

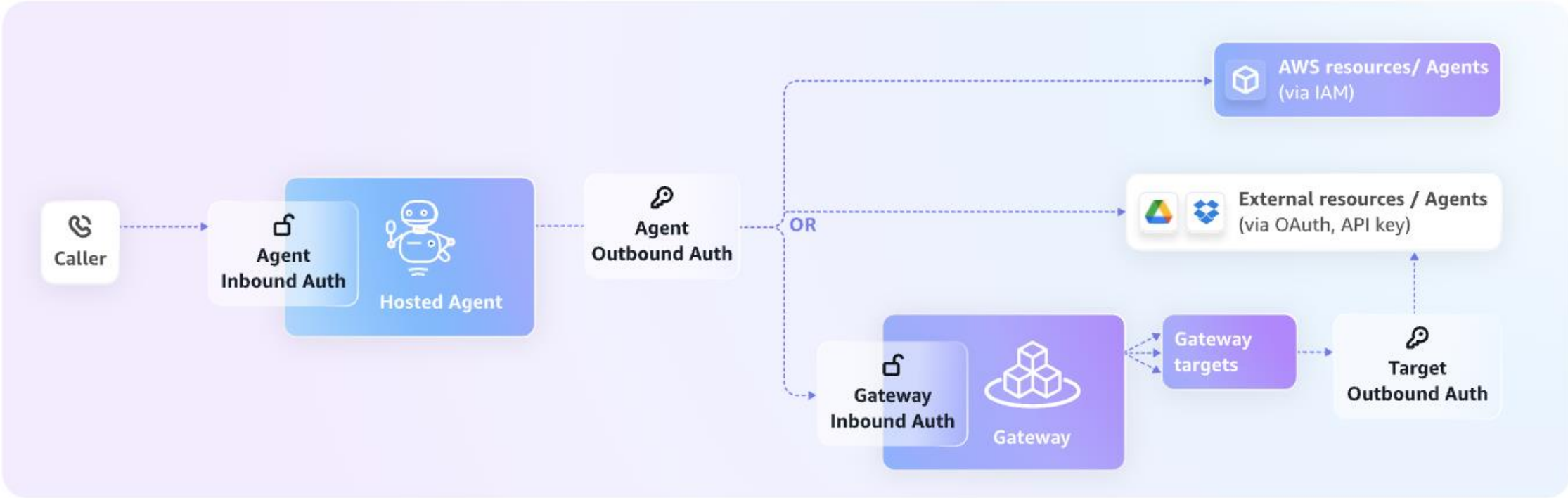
# AgentCore Gateway



```yaml
Gateway:
  Type: Custom::AgentCoreGateway
  Properties:
    ServiceToken: !Ref AgentCoreProvisioner
    Name: Gateway
    Description: Gateway to access other agents and MCP servers.
    RoleArn: !GetAtt GatewayRole.Arn
    ProtocolType: MCP
    ProtocolConfiguration:
      mcp:
        SearchType: SEMANTIC
    AuthorizerType: CUSTOM_JWT
    AuthorizerConfiguration:
      customJWTAuthorizer:
        DiscoveryUrl: !Sub https://cognito-idp.${AWS::Region}.amazonaws.com/${UserPoolId}/.well-known/openid-config
        AllowedClients:
          - !Ref GatewayClient
```

# AgentCore Identity

▼ How it works




### Inbound Auth

**Create and Integrate:** Create an Inbound Auth that authenticates and authorizes the caller to get access to an agent, tool, runtime, or Gateway. You can configure using either IAM or JSON web tokens (for example, OAuth tokens) from your identity provider.

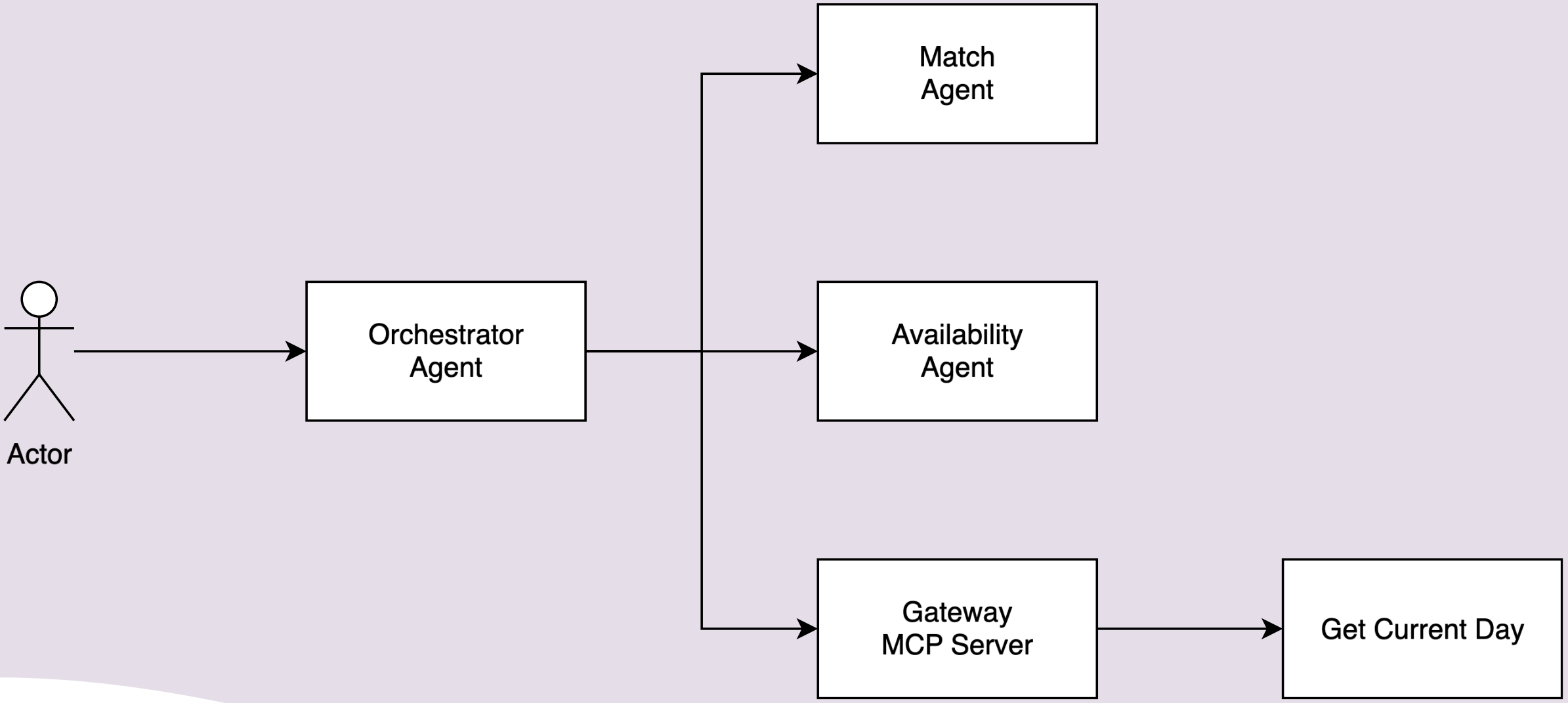
**Invoke:** Provide the OAuth2 JWT access token associated with the user you're representing.

### Outbound Auth

**Create and Integrate:** Create an Outbound Auth using either an API key or an OAuth client that allows an agent, tool or Gateway access to downstream resources. You can use the Outbound Auth ARN in your agent or tool code.

**Invoke:** No action required.

```python
from bedrock_agentcore.identity.auth import requires_access_token


@requires_access_token(
    provider_name=os.environ["IDENTITY_PROVIDER_NAME"],
    auth_flow="M2M",
    scopes=[f"https://{DOMAIN_NAME}/read"],
)
async def need_access_token(*, access_token: str):
    global ACCESS_TOKEN
    logger.info(
        f"Retrieved the access token from {os.environ['IDENTITY_PROVIDER_NAME']}"
    )
    ACCESS_TOKEN = access_token
```


Match
Agent
Actor
Orchestrator
Agent
Availability
Agent
Gateway
MCP Server
Get Current Day

Agentic AI Demo
by Xebia

Joris Conijn

No chat history yet

Type your message...

Agentic AI Demo
by Xebia

Joris Conijn

No chat history yet

Type your message...

# Lessons Learned

- AgentCore is only available in a few regions
- No CloudFormation support
- Logs are very verbose, and tailing log groups has become nearly impossible
- The service is in preview; as a result, I ran into bugs and frustration
  - Enable CloudTrail data events for AgentCore
  - Enable log delivery for all services within AgentCore
- Unit testing an agent becomes somewhat impossible (due to the streamed responses and their “structure”)

- For me, the best way to learn is to build

# Thank You

;( 404