**Technical Guide**

# Understanding AWS Bedrock AgentCore

AWS Bedrock AgentCore is a serverless framework for deploying and running agentic AI applications at scale. Unlike traditional Lambda functions or EC2 instances, AgentCore provides:

**Managed Agent Execution**: AgentCore automatically handles agent orchestration, retry logic, timeout management, and scaling without requiring manual infrastructure configuration.

**Built-in Memory Management**: AgentCore includes a distributed memory system that persists agent state, conversation history, and session data across invocations, eliminating the need for external state management.

**Integrated Tool Calling**: The framework natively supports function calling and tool integration, automatically routing tool invocations and managing execution flow.

**Bedrock Model Access**: AgentCore applications have optimized access to all Bedrock models including Nova Pro, Nova Sonic, Claude, and others, with automatic credential management and request optimization.

**Event-Driven Architecture**: AgentCore supports both synchronous API invocations and asynchronous event-driven processing, enabling complex workflows with minimal code.

# Pre-Deployment Requirements

**AWS Account Setup**: You need an AWS account with Bedrock access enabled in your target region (us-east-1 recommended for full Nova model availability). Ensure your account has sufficient service quotas for Bedrock model invocations.

**IAM Permissions**: Your deployment user or role requires:

* Bedrock full access for model invocation
* S3 full access for document storage and status tracking
* CloudWatch Logs for monitoring and debugging
* AgentCore deployment permissions
* STS assume role capabilities for credential management

**Local Development Environment**: Install Python 3.9 or higher, pip for package management, and the AgentCore CLI tool. Ensure you have AWS CLI configured with appropriate credentials.

**S3 Bucket Preparation**: Create a dedicated S3 bucket for your deployment (e.g., nyl-underwriting-documents-121409194654). Configure appropriate lifecycle policies for automated cleanup of old sessions.

# Application Structure and Configuration

The SDIS application follows AgentCore's required structure with specific entry points and dependencies:

**Primary Entry Point**: The agentcore\_main.py file serves as the application entry point, containing the invoke() function decorated with @app.entrypoint. This function receives all incoming requests and routes them appropriately.

**Agent Definitions**: All specialized agents (data intake, medical assessment, financial analysis, etc.) are defined in underwriting\_agents.py using the Strands framework, which provides the Agent abstraction layer for Bedrock models.

**Configuration Management**: The config.py file contains all business rules, underwriting guidelines, risk scoring parameters, and system configuration. This separation allows updating business logic without code changes.

**Model Definitions**: The models.py file initializes Bedrock model clients for Nova Pro, Nova Lite, Nova Premier, Nova Micro, and Nova Sonic, with appropriate temperature settings for each use case.

**Dependencies**: The requirements.txt specifies all required packages including bedrock-agentcore, strands-agents, boto3, flask-socketio, and document processing libraries. AgentCore automatically installs these during deployment.

# Deployment Process

**Package Preparation**: AgentCore requires all application code, dependencies, and configuration to be packaged together. Ensure your directory contains all Python files, requirements.txt, and any additional resources like templates or static files.

**AgentCore CLI Deployment**: Use the AgentCore command-line interface to deploy your application:

agentcore deploy --name nyl-underwriting-system --region us-east-1

```

The CLI tool automatically packages your application, uploads it to AWS, creates necessary IAM roles, configures CloudWatch logging, and deploys the application to AgentCore infrastructure.

\*\*Environment Variables\*\*: AgentCore supports environment variable injection during deployment. Critical variables include AWS\_REGION, S3 bucket names, and any API keys for external integrations. These are configured through the AgentCore console or CLI.

\*\*Version Management\*\*: Each deployment creates a new version with automatic rollback capabilities. You can maintain multiple versions simultaneously for A/B testing or gradual rollout strategies.

# Running the Frontend Application

The Flask-based frontend serves as the user interface and coordinates with the AgentCore backend through S3 and direct AgentCore invocations.

\*\*Local Frontend Execution\*\*: Start the Flask application using `python run.py`. This launches the web server on port 8080, initializes SocketIO for real-time communication, and establishes boto3 clients for S3 and AgentCore interaction.

\*\*Frontend-Backend Communication\*\*: When users upload documents through the web interface, the frontend:

1. Validates the ZIP file and uploads it to S3 in a session-specific folder

2. Constructs an AgentCore invocation payload with session\_id and S3 location

3. Triggers the AgentCore backend using subprocess calls to the AgentCore CLI

4. Monitors the S3 agent\_status.json file for processing updates

5. Displays real-time status to users through SocketIO events

\*\*Nova Sonic Voice Integration\*\*: The voice advisory feature uses WebSocket connections to stream audio bidirectionally between the browser and Nova Sonic. The Flask backend manages Nova Sonic sessions, handling audio encoding, conversation state, and response streaming.

# Monitoring and Logging

\*\*CloudWatch Logs Integration\*\*: AgentCore automatically sends all application logs to CloudWatch. Each agent execution, tool invocation, and error generates structured log entries with timestamps, session IDs, and contextual information.

\*\*Log Group Structure\*\*: Logs are organized by AgentCore application name and version, with separate log streams for different execution contexts. Search and filter logs using CloudWatch Insights with queries like:

- Filter by session\_id to trace a specific user journey

- Search for ERROR or WARNING to identify issues

- Analyze agent execution times and performance metrics

\*\*S3 Monitoring\*\*: Monitor your S3 bucket for session folders, agent status files, and generated policy documents. Each session creates a structured folder hierarchy:

```

session\_2025-01-20\_14-30-45\_abc123def/

├── agent\_status.json (real-time processing status)

├── session\_2025-01-20\_14-30-45\_abc123def\_upload.zip (original documents)

├── conversation\_context.json (Nova Sonic conversation data)

└── policy\_generated\_20250120\_143500.pdf (final policy document)

**AgentCore Metrics**: Access AgentCore-specific metrics through the AWS console, including invocation counts, execution duration, error rates, and throttling events. Set up CloudWatch alarms for error rate spikes or extended execution times.

**Custom Logging**: The application implements structured logging throughout, with specific markers for each processing stage:

* [NOVA] for Nova Sonic voice interactions
* [AGENT] for individual agent execution
* [S3] for storage operations
* [SOCKET] for WebSocket events
* [ERROR] for all error conditions

# Testing and Validation

**Session Creation Test**: Verify the system by creating a new session through the frontend. Check CloudWatch logs for the session creation event and confirm the S3 folder structure is created correctly.

**Document Upload Validation**: Upload a test ZIP file containing sample health insurance PDFs. Monitor the S3 bucket to ensure the file appears in the session folder, then watch CloudWatch logs for AgentCore invocation and agent execution logs.

**Agent Processing Verification**: During document processing, regularly check the agent\_status.json file in S3. It should update after each agent completes, showing status changes from "pending" to "in\_progress" to "completed" for each of the eight agents.

**Voice Interaction Testing**: Use the Nova Sonic interface to have a conversation with Alan. Speak naturally about insurance needs and verify that:

* The assistant responds appropriately to questions
* Conversation context is maintained across turns
* Upload links are only provided when explicitly requested
* The conversation\_context.json file in S3 captures all interactions

**Policy Generation Verification**: After successful underwriting completion, verify that:

* A PDF policy document is created in the session folder
* The agent\_status.json includes policy\_generated information
* The PDF can be viewed and downloaded through the frontend
* The document contains properly formatted tables and all required sections

**Error Handling Tests**: Intentionally trigger error conditions to validate resilience:

* Upload corrupted ZIP files
* Provide documents with encoding issues
* Let AWS credentials expire mid-processing
* Test with extremely large document sets

# Troubleshooting Common Issues

**Credential Expiration**: If you see "AWS credentials not available or expired" errors, refresh your temporary credentials. The system automatically detects expiration and provides clear error messages with remediation steps.

**Model Throttling**: During high-volume processing, Bedrock may throttle requests. The application implements automatic retry logic with exponential backoff. Monitor CloudWatch for throttling events and request quota increases if needed.

**S3 Consistency**: Occasionally, S3 eventual consistency may cause status updates to lag. The frontend polling mechanism accounts for this with appropriate retry intervals.

**Agent Failures**: If an individual agent fails, the orchestrator continues processing and flags the issue in the final summary. Check CloudWatch logs for the specific agent that failed and review the error message for root cause analysis.

**Memory Issues**: Large document sets may exceed AgentCore memory limits. The system includes validation for file size and document count, rejecting uploads over 50MB or with excessive page counts.

# Production Considerations

**Scaling**: AgentCore automatically scales based on invocation volume. Monitor your CloudWatch metrics to understand peak usage patterns and adjust concurrency limits if needed.

**Cost Optimization**: Bedrock model invocations are the primary cost driver. Consider implementing caching for frequently accessed policy information and using Nova Lite for simpler tasks to reduce costs.

**Security**: Enable encryption at rest for S3 buckets, use VPC endpoints for AWS service access, implement IP whitelisting for the frontend if needed, and ensure all PHI is properly masked in logs.

**Backup and Recovery**: Implement S3 versioning and cross-region replication for critical session data. Set up automated backups of the AgentCore application configuration and code.

**Performance Tuning**: Optimize agent prompts to reduce token usage, implement parallel agent execution where possible, and cache intermediate results to avoid redundant processing.

This deployment guide provides the foundation for successfully running SDIS in production environments, ensuring reliability, observability, and maintainability of your agentic AI insurance advisory system.

# Deployment Commands

**AWS Bedrock AgentCore - Command Reference**

**Initial Setup Commands**

**Install AWS CLI (if not already installed)**

bash

*# For Linux/macOS*

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

*# For Windows*

*# Download and run the MSI installer from:*

*# https://awscli.amazonaws.com/AWSCLIV2.msi*

**Configure AWS Credentials**

bash

*# Configure default profile*

aws configure

*# You'll be prompted for:*

*# AWS Access Key ID: [Enter your access key]*

*# AWS Secret Access Key: [Enter your secret key]*

*# Default region name: us-east-1*

*# Default output format: json*

*# Verify configuration*

aws sts get-caller-identity

*# Expected output shows your Account ID, UserId, and ARN*

**Install Python and Dependencies**

bash

*# Check Python version (requires 3.9+)*

python --version

*# or*

python3 --version

*# Create virtual environment (recommended)*

python3 -m venv venv

*# Activate virtual environment*

*# On Linux/macOS:*

source venv/bin/activate

*# On Windows:*

venv\Scripts\activate

*# Install application dependencies*

pip install -r requirements.txt

*# Verify key packages installed*

pip list | grep bedrock-agentcore

pip list | grep strands-agents

pip list | grep boto3

**Install AgentCore CLI**

bash

*# Install AgentCore command-line interface*

pip install bedrock-agentcore-cli

*# Verify installation*

agentcore --version

*# Check available commands*

agentcore --help

**S3 Bucket Setup Commands**

**Create S3 Bucket**

bash

*# Create bucket in us-east-1 region*

aws s3 mb s3://nyl-underwriting-documents-121409194654 --region us-east-1

*# Verify bucket created*

aws s3 ls | grep nyl-underwriting

*# Enable versioning (recommended for production)*

aws s3api put-bucket-versioning \

--bucket nyl-underwriting-documents-121409194654 \

--versioning-configuration Status=Enabled

*# Configure lifecycle policy for automatic cleanup (optional)*

cat > lifecycle-policy.json <<EOF

{

"Rules": [

{

"Id": "DeleteOldSessions",

"Status": "Enabled",

"Prefix": "session\_",

"Expiration": {

"Days": 30

}

}

]

}

EOF

aws s3api put-bucket-lifecycle-configuration \

--bucket nyl-underwriting-documents-121409194654 \

--lifecycle-configuration file://lifecycle-policy.json

**Test S3 Access**

bash

*# Upload test file*

echo "Test file" > test.txt

aws s3 cp test.txt s3://nyl-underwriting-documents-121409194654/

*# List bucket contents*

aws s3 ls s3://nyl-underwriting-documents-121409194654/

*# Download test file*

aws s3 cp s3://nyl-underwriting-documents-121409194654/test.txt ./test-downloaded.txt

*# Clean up test file*

aws s3 rm s3://nyl-underwriting-documents-121409194654/test.txt

rm test.txt test-downloaded.txt

**AgentCore Deployment Commands**

**Prepare Application for Deployment**

bash

*# Navigate to application directory*

cd /path/to/your/sdis-application

*# Verify all required files exist*

ls -la agentcore\_main.py underwriting\_agents.py config.py models.py requirements.txt

*# Run pre-deployment validation (if available)*

python -m py\_compile agentcore\_main.py

python -m py\_compile underwriting\_agents.py

*# Check for syntax errors*

python agentcore\_main.py --help 2>&1 | head -n 5

**Deploy to AgentCore**

bash

*# Basic deployment command*

agentcore deploy \

--name nyl-underwriting-system \

--region us-east-1 \

--entry-point agentcore\_main.py

*# Deployment with environment variables*

agentcore deploy \

--name nyl-underwriting-system \

--region us-east-1 \

--entry-point agentcore\_main.py \

--env AWS\_REGION=us-east-1 \

--env S3\_BUCKET=nyl-underwriting-documents-121409194654

*# Deployment with custom timeout (in seconds)*

agentcore deploy \

--name nyl-underwriting-system \

--region us-east-1 \

--entry-point agentcore\_main.py \

--timeout 1800 \

--memory 2048

*# View deployment progress*

*# Output shows:*

*# - Packaging application*

*# - Uploading to AWS*

*# - Creating IAM roles*

*# - Configuring CloudWatch*

*# - Deployment URL and invocation details*

**Verify Deployment**

bash

*# List all deployed AgentCore applications*

agentcore list --region us-east-1

*# Get specific application details*

agentcore describe \

--name nyl-underwriting-system \

--region us-east-1

*# Check application status*

agentcore status \

--name nyl-underwriting-system \

--region us-east-1

**Test AgentCore Deployment**

bash

*# Test with simple health check payload*

agentcore invoke '{"request\_type": "get\_status", "session\_id": "test-session"}' \

--name nyl-underwriting-system \

--region us-east-1

*# Test session creation*

agentcore invoke '{"request\_type": "create\_session"}' \

--name nyl-underwriting-system \

--region us-east-1

*# Expected output shows session\_id and s3\_bucket information*

**Running the Frontend Demo**

**Start Flask Application**

bash

*# Ensure you're in the application directory*

cd /path/to/your/sdis-application

*# Activate virtual environment (if not already active)*

source venv/bin/activate *# Linux/macOS*

*# or*

venv\Scripts\activate *# Windows*

*# Set environment variables (if not in .env file)*

export AWS\_REGION=us-east-1

export AWS\_DEFAULT\_REGION=us-east-1

export S3\_BUCKET=nyl-underwriting-documents-121409194654

*# Start the Flask application*

python run.py

*# Expected output:*

*# ============================================================*

*# NOVA SONIC VOICE INTEGRATION policy generation*

*# ============================================================*

*# [INFO] AWS Region configured: us-east-1*

*# [INFO] S3 Bucket: nyl-underwriting-documents-121409194654*

*# [SUCCESS] S3 bucket accessible: nyl-underwriting-documents-121409194654*

*# ============================================================*

*# [INFO] Starting Flask-SocketIO with Nova Sonic integration*

*# [INFO] Access: http://127.0.0.1:8080*

*# [INFO] Health check: http://127.0.0.1:8080/health*

*# ============================================================*

**Access the Application**

bash

*# Open in default browser (Linux/macOS)*

open http://127.0.0.1:8080

*# Open in default browser (Windows)*

start http://127.0.0.1:8080

*# Or manually navigate to:*

*# http://127.0.0.1:8080*

**Test Health Endpoint**

bash

*# Check application health*

curl http://127.0.0.1:8080/health | jq

*# Expected output:*

*# {*

*# "status": "healthy",*

*# "timestamp": "2025-01-20T14:30:00",*

*# "service": "NYL Underwriting System - S3 Frontend with Nova Sonic",*

*# "version": "3.0.0",*

*# "s3\_bucket": "nyl-underwriting-documents-121409194654",*

*# "s3\_status": "connected",*

*# "nova\_sonic": "enabled"*

*# }*

**Testing Commands**

**Create Test Session**

bash

*# Using curl to create session*

curl -X POST http://127.0.0.1:8080/api/create-session | jq

*# Using AgentCore CLI*

agentcore invoke '{"request\_type": "create\_session"}' \

--name nyl-underwriting-system \

--region us-east-1 | jq

*# Save session\_id from response for subsequent commands*

SESSION\_ID="session\_2025-01-20\_14-30-45\_abc12345"

**Upload Test Documents**

bash

*# Prepare test ZIP file with sample PDFs*

zip -r test-documents.zip sample\_policies/\*.pdf

*# Upload using curl*

curl -X POST http://127.0.0.1:8080/upload/$SESSION\_ID \

-F "zipFileInput=@test-documents.zip"

*# Verify upload in S3*

aws s3 ls s3://nyl-underwriting-documents-121409194654/$SESSION\_ID/

*# Expected output shows uploaded ZIP file*

**Trigger AgentCore Processing**

bash

*# Create processing payload*

cat > process-payload.json <<EOF

{

"request\_type": "s3\_process",

"s3\_bucket": "nyl-underwriting-documents-121409194654",

"s3\_key": "$SESSION\_ID/${SESSION\_ID}\_upload.zip",

"session\_id": "$SESSION\_ID"

}

EOF

*# Invoke AgentCore processing*

agentcore invoke "$(cat process-payload.json)" \

--name nyl-underwriting-system \

--region us-east-1

*# This triggers the 8-agent workflow*

**Monitor Processing Status**

bash

*# Check status via HTTP endpoint*

curl http://127.0.0.1:8080/status/$SESSION\_ID | jq

*# Check agent status directly from S3*

aws s3 cp s3://nyl-underwriting-documents-121409194654/$SESSION\_ID/agent\_status.json - | jq

*# Watch for status updates (Linux/macOS)*

watch -n 5 "aws s3 cp s3://nyl-underwriting-documents-121409194654/$SESSION\_ID/agent\_status.json - | jq '.agents | to\_entries[] | {agent: .key, status: .value.status}'"

*# Manual polling (all platforms)*

while true; do

curl -s http://127.0.0.1:8080/status/$SESSION\_ID | jq '.agents | to\_entries[] | {agent: .key, status: .value.status}'

sleep 5

done

**Retrieve Processing Results**

bash

*# Get comprehensive summary*

curl http://127.0.0.1:8080/status/$SESSION\_ID | jq '.agents.summary\_generation.analysis'

*# Download generated policy PDF*

curl http://127.0.0.1:8080/download\_policy/$SESSION\_ID \

--output policy\_$SESSION\_ID.pdf

*# View policy in browser*

curl http://127.0.0.1:8080/view\_policy/$SESSION\_ID

*# Get policy status*

curl http://127.0.0.1:8080/policy\_status/$SESSION\_ID | jq

**Monitoring and Logging Commands**

**View CloudWatch Logs**

bash

*# List log groups for AgentCore application*

aws logs describe-log-groups \

--log-group-name-prefix /aws/agentcore/nyl-underwriting-system

*# Get latest log stream*

LOG\_STREAM=$(aws logs describe-log-streams \

--log-group-name /aws/agentcore/nyl-underwriting-system \

--order-by LastEventTime \

--descending \

--max-items 1 \

--query 'logStreams[0].logStreamName' \

--output text)

echo "Latest log stream: $LOG\_STREAM"

*# View recent logs*

aws logs tail /aws/agentcore/nyl-underwriting-system \

--follow \

--format short

*# Search logs for specific session*

aws logs filter-log-events \

--log-group-name /aws/agentcore/nyl-underwriting-system \

--filter-pattern "$SESSION\_ID" \

--start-time $(date -u -d '1 hour ago' +%s)000

*# Search for errors*

aws logs filter-log-events \

--log-group-name /aws/agentcore/nyl-underwriting-system \

--filter-pattern "ERROR" \

--start-time $(date -u -d '1 hour ago' +%s)000

**Monitor S3 Session Activity**

bash

*# List all active sessions*

aws s3 ls s3://nyl-underwriting-documents-121409194654/ | grep session\_

*# Count total sessions*

aws s3 ls s3://nyl-underwriting-documents-121409194654/ | grep session\_ | wc -l

*# Get session details*

aws s3 ls s3://nyl-underwriting-documents-121409194654/$SESSION\_ID/ --recursive --human-readable

*# Monitor session folder size*

aws s3 ls s3://nyl-underwriting-documents-121409194654/$SESSION\_ID/ --recursive --summarize

*# Download all session files*

aws s3 sync s3://nyl-underwriting-documents-121409194654/$SESSION\_ID/ ./session\_backup/$SESSION\_ID/

**Application Performance Monitoring**

bash

*# Check AgentCore metrics*

aws cloudwatch get-metric-statistics \

--namespace AWS/AgentCore \

--metric-name Invocations \

--dimensions Name=ApplicationName,Value=nyl-underwriting-system \

--start-time $(date -u -d '1 hour ago' --iso-8601) \

--end-time $(date -u --iso-8601) \

--period 300 \

--statistics Sum

*# Check error rate*

aws cloudwatch get-metric-statistics \

--namespace AWS/AgentCore \

--metric-name Errors \

--dimensions Name=ApplicationName,Value=nyl-underwriting-system \

--start-time $(date -u -d '1 hour ago' --iso-8601) \

--end-time $(date -u --iso-8601) \

--period 300 \

--statistics Sum

*# Check execution duration*

aws cloudwatch get-metric-statistics \

--namespace AWS/AgentCore \

--metric-name Duration \

--dimensions Name=ApplicationName,Value=nyl-underwriting-system \

--start-time $(date -u -d '1 hour ago' --iso-8601) \

--end-time $(date -u --iso-8601) \

--period 300 \

--statistics Average,Maximum

**Update and Maintenance Commands**

**Update AgentCore Application**

bash

*# Make code changes, then redeploy*

agentcore deploy \

--name nyl-underwriting-system \

--region us-east-1 \

--entry-point agentcore\_main.py \

--update

*# Deploy new version without replacing current*

agentcore deploy \

--name nyl-underwriting-system \

--region us-east-1 \

--entry-point agentcore\_main.py \

--version v2

*# Switch traffic to new version*

agentcore update-alias \

--name nyl-underwriting-system \

--alias production \

--version v2 \

--region us-east-1

**Rollback Commands**

bash

*# List all versions*

agentcore list-versions \

--name nyl-underwriting-system \

--region us-east-1

*# Rollback to previous version*

agentcore update-alias \

--name nyl-underwriting-system \

--alias production \

--version v1 \

--region us-east-1

*# Delete specific version*

agentcore delete-version \

--name nyl-underwriting-system \

--version v2 \

--region us-east-1