

# AWS MCP Server Deployment Guide

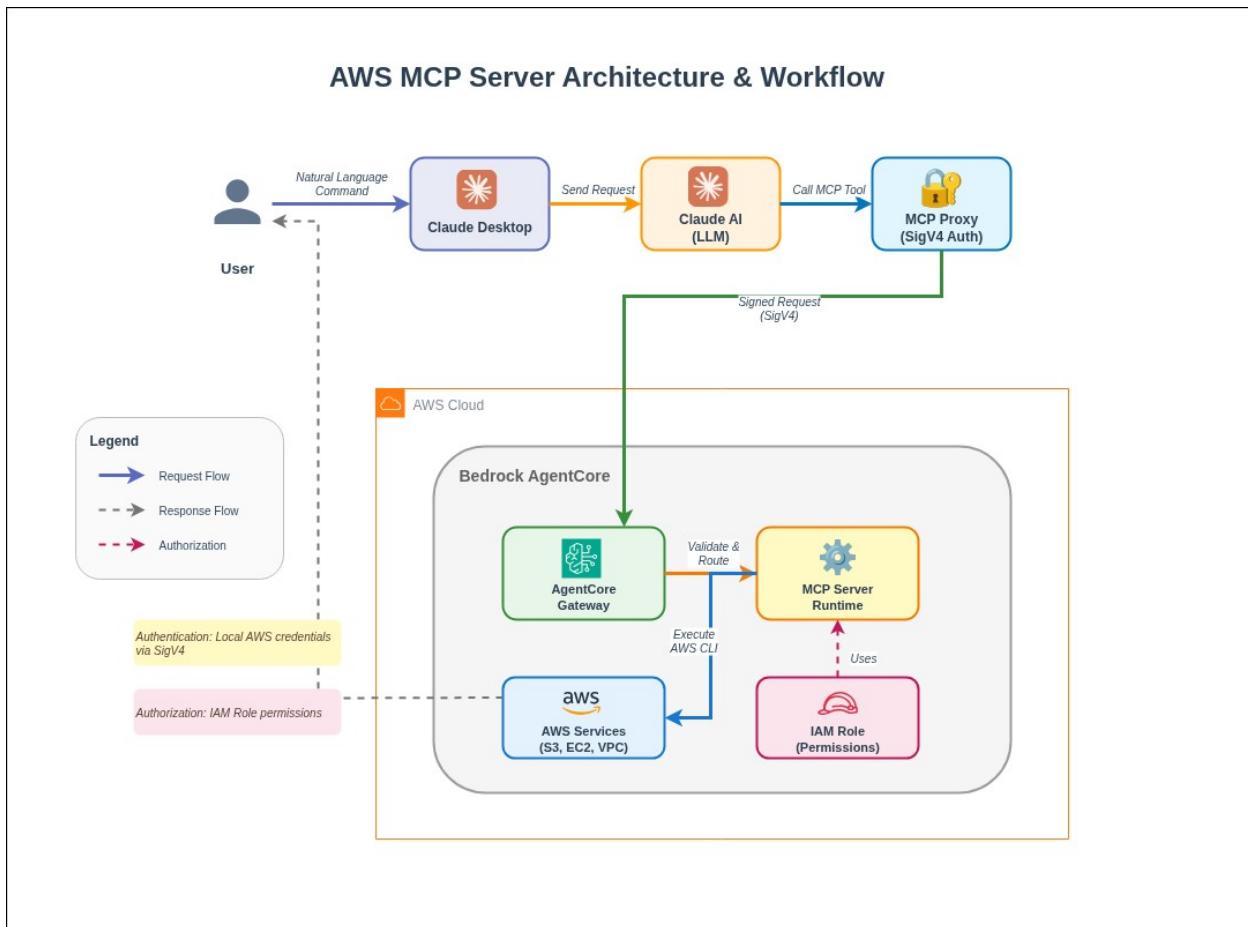
Complete Setup for AWS API MCP Server with Claude Desktop

**Author:** Wajahat Ullah

## Table of Contents

Architecture.....	2
1. Overview.....	2
2. Architecture & Communication Flow.....	3
3. Prerequisites.....	3
4. Step-by-Step Deployment.....	4
5. Testing the Setup.....	10
6. Cost Breakdown.....	15
7. Troubleshooting.....	15
Quick Reference.....	17

# Architecture



## 1. Overview

This guide provides complete instructions for deploying the AWS API MCP Server to Amazon Bedrock AgentCore. This setup enables you to use natural language commands in Claude Desktop to manage AWS resources through the Model Context Protocol (MCP).

### What You'll Build

- AWS MCP Server hosted on Bedrock AgentCore
- IAM role with custom permissions for AWS operations
- Claude Desktop integration using AWS credentials

- Natural language interface for AWS management

## 2. Architecture & Communication Flow

### How It Works

The system consists of four main components working together:

1. **Claude Desktop** - Your user interface where you type natural language commands
2. **Claude AI** - The language model that interprets commands and decides which tools to call
3. **AWS MCP Proxy** - Authenticates requests using your AWS credentials (SigV4)
4. **MCP Server (AgentCore)** - Executes AWS CLI commands using its IAM role

### Communication Flow

```
You: 'List my S3 buckets'
      ↓
Claude Desktop (sends to Claude AI)
      ↓
Claude AI (decides to call MCP tool)
      ↓
MCP Proxy (signs request with your AWS creds)
      ↓
AgentCore (validates signature)
      ↓
MCP Server (runs: aws s3 ls)
      ↓
Results return through same chain
```

### Security Model

- **Authentication:** Your local AWS credentials authenticate you to AgentCore
- **Authorization:** The MCP server's IAM role determines what AWS operations are allowed
- **Isolation:** Each AgentCore session is isolated; stateless execution

## 3. Prerequisites

### Required Tools

- **AWS CLI** - Install from: <https://aws.amazon.com/cli/>

- **Claude Desktop** - Download from: <https://claude.ai/download>
- **uv Package Manager** - Will be installed in Step 4
- **Claude Pro Subscription** - Required for using Claude Desktop

## AWS Account Setup

- AWS account with admin access (for initial setup)
- SSO or IAM credentials configured locally
- Region: us-east-1 (recommended, but others work)

## Verify Prerequisites

Run these commands to verify your setup:

```
# Check AWS CLI
aws --version

# Check AWS credentials
aws sts get-caller-identity
```

## 4. Step-by-Step Deployment

### Step 4.1: Create IAM Trust Policy

Create a trust policy that allows Bedrock AgentCore to assume the role:

```
cat > trust-policy.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRolePolicy",
            "Effect": "Allow",
            "Principal": {
                "Service": "bedrock-agentcore.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "<YOUR_ACCOUNT_ID>"
                },
            }
        }
    ]
}'
```

```

        "ArnLike": {
            "aws:SourceArn": "arn:aws:bedrock-
agentcore:*:<YOUR_ACCOUNT_ID>:*"
        }
    }
}
]

EOF

# Create the IAM role
aws iam create-role \
--role-name aws-api-mcp-execution-role \
--assume-role-policy-document file://trust-policy.json

```

## Step 4.2: Attach AgentCore Base Permissions

These permissions allow AgentCore to function properly (ECR access, CloudWatch logs, etc.):

```

cat > agentcore-base-permissions.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ECRImageAccess",
            "Effect": "Allow",
            "Action": ["ecr:BatchGetImage", "ecr:GetDownloadUrlForLayer"],
            "Resource": ["arn:aws:ecr:us-east-1:709825985650:repository/*"]
        },
        {
            "Sid": "ECRTokenAccess",
            "Effect": "Allow",
            "Action": ["ecr:GetAuthorizationToken"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["logs>CreateLogGroup", "logs>CreateLogStream",
                      "logs>PutLogEvents", "logs>DescribeLogStreams"],

```

```

        "Resource": "arn:aws:logs:us-east-1:<YOUR_ACCOUNT_ID>:log-group:/aws/bedrock-agentcore/runtimes/*"
    },
    {
        "Effect": "Allow",
        "Action": ["xray:PutTraceSegments", "xray:PutTelemetryRecords"],
        "Resource": "*"
    }
]
}
EOF

# Attach base permissions
aws iam put-role-policy \
--role-name aws-api-mcp-execution-role \
--policy-name AgentCoreBasePermissions \
--policy-document file://agentcore-base-permissions.json

```

### Step 4.3: Attach Custom AWS Permissions

These permissions define what AWS operations the MCP server can perform. Customize based on your needs:

```

cat > custom-aws-permissions.json << 'EOF'
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Operations",
            "Effect": "Allow",
            "Action": [
                "s3>ListAllMyBuckets", "s3>ListBucket",
                "s3>CreateBucket", "s3GetObject", "s3PutObject"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EC2Operations",
            "Effect": "Allow",

```

```

        "Action": [
            "ec2:Describe*", "ec2:RunInstances",
            "ec2:CreateSecurityGroup",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:CreateVpc", "ec2:CreateSubnet",
            "ec2:CreateInternetGateway"
        ],
        "Resource": "*"
    }
]
}

# Attach custom permissions
aws iam put-role-policy \
--role-name aws-api-mcp-execution-role \
--policy-name CustomAWSPermissions \
--policy-document file://custom-aws-permissions.json

```

## Step 4.4: Deploy to AgentCore

**Important:** Check AWS Marketplace for the latest version tag:

<https://aws.amazon.com/marketplace/pp/prodview-lqqkwbcraxsgw>

Official marketplace link: [https://aws.amazon.com/marketplace/pp/prodview-lqqkwbcraxsgw?sr=0-1&ref\\_=beagle&applicationId=AWSMPContessa](https://aws.amazon.com/marketplace/pp/prodview-lqqkwbcraxsgw?sr=0-1&ref_=beagle&applicationId=AWSMPContessa)

Official Deployment Guide: <https://github.com/awslabs/mcp/blob/main/src/aws-api-mcp-server/DEPLOYMENT.md>

```

# Deploy runtime (replace LATEST_VERSION with actual version like 1.2.0)
aws bedrock-agentcore-control create-agent-runtime \
--region us-east-1 \
--agent-runtime-name "awsapimcpserver" \
--agent-runtime-artifact '{
    "containerConfiguration": {
        "containerUri": "709825985650.dkr.ecr.us-east-1.amazonaws.com/amazon-
web-services/aws-api-mcp-server:LATEST_VERSION"
    }
}' \
--role-arn "arn:aws:iam::<YOUR_ACCOUNT_ID>:role/aws-api-mcp-execution-
role" \

```

```
--network-configuration '{"networkMode": "PUBLIC"}' \
--protocol-configuration '{"serverProtocol": "MCP"}' \
--environment-variables '{
    "AUTH_TYPE": "no-auth",
    "AWS_API_MCP_HOST": "0.0.0.0",
    "AWS_API_MCP_PORT": "8000",
    "AWS_API_MCP_STATELESS_HTTP": "true",
    "AWS_API_MCP_TRANSPORT": "streamable-http",
    "AWS_API_MCP_ALLOWED_HOSTS": "*",
    "AWS_API_MCP_ALLOWED_ORIGINS": "*"
}'
```

**Save the agent-runtime-id from the response - you'll need it for the next step.**

## Step 4.5: Get Endpoint URL

```
# Get runtime details (replace YOUR_RUNTIME_ID)
aws bedrock-agentcore-control get-agent-runtime \
--agent-runtime-id "YOUR_RUNTIME_ID" \
--region us-east-1
```

Copy the ARN from the response and encode it:

```
# Encode the ARN (replace YOUR_ARN with actual ARN)
echo "YOUR_ARN" | sed 's/:/%3A/g; s/\//%2F/g'
```

### Example:

```
# Original ARN:
arn:aws:bedrock-agentcore:us-east-1:<YOUR_ACCOUNT_ID>:runtime/awsapimcpserver-
abc123

# Encoded ARN:
arn%3Aaws%3Abedrock-agentcore%3Aus-east-1%3A<YOUR_ACCOUNT_ID>%3Aruntime
%2Fawsapimcpserver-abc123
```

### Your final endpoint URL format:

```
https://bedrock-agentcore.us-east-1.amazonaws.com/runtimes/YOUR_ENCODED_ARN/
invocations?qualifier=DEFAULT
```

## Step 4.6: Install UV Package Manager

```
# Install uv
```

```
curl -LsSf https://astral.sh/uv/install.sh | sh

# Add to PATH
source $HOME/.local/bin/env
```

## Step 4.7: Configure Claude Desktop

Edit the Claude Desktop configuration file. **Important:** The file must be named **claude\_desktop\_config.json** (not config.json).

**File location:** `~/.config/Claude/claude_desktop_config.json`

```
{
  "mcpServers": {
    "aws-api-mcp-server": {
      "autoApprove": [],
      "disabled": false,
      "timeout": 600,
      "type": "stdio",
      "command": "uvx",
      "args": [
        "mcp-proxy-for-aws@latest",
        "https://bedrock-agentcore.us-east-1.amazonaws.com/runtimes/YOUR_ENCODED_ARN/invocations?qualifier=DEFAULT",
        "--region",
        "us-east-1"
      ],
      "env": {}
    }
  }
}
```

**Important:** Replace `YOUR_ENCODED_ARN` with your actual encoded ARN from Step 4.5.

## Step 4.8: Restart Claude Desktop

Completely close and restart Claude Desktop:

```
# Kill Claude Desktop process
killall claude
```

Then start Claude Desktop from your applications menu.

**Important:** Start a **new conversation** after restarting. Existing chats won't have access to the MCP server.

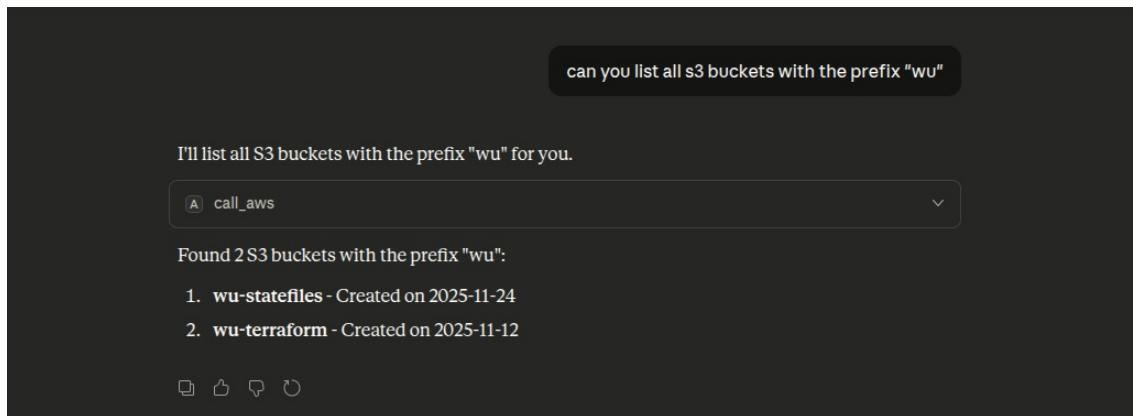
## 5. Testing the Setup

### Basic S3 Test

In a new Claude Desktop conversation, type:

List my S3 buckets

Claude should use the MCP server to execute the command and return your S3 buckets.



create me an s3 bucket named "wu-mcp-bucket", with public access denied and versioning enabled.

I'll create an S3 bucket named "wu-mcp-bucket" with public access blocked and versioning enabled.

A call\_aws

Good! Bucket created. Now let me enable versioning:

A call\_aws

Now let me block public access:

A call\_aws

Perfect! Let me verify the bucket configuration:

A call\_aws

S3 bucket "wu-mcp-bucket" successfully created!

Configuration applied:

- **Bucket Name:** wu-mcp-bucket
- **Region:** us-east-1
- **Versioning:** Enabled
- **Public Access:** Blocked (all public access settings enabled)
  - BlockPublicAcls: true
  - IgnorePublicAcls: true
  - BlockPublicPolicy: true
  - RestrictPublicBuckets: true

The bucket is now ready to use with versioning enabled and all public access denied.



create an ec2 instance named "noor-test".

I'll attempt to create an EC2 instance named "noor-test". Let me first check for available AMIs and then launch the instance.

A call\_aws

Good! I found an Amazon Linux 2023 AMI. Now let me try to launch the instance:

A call\_aws

Perfect! Let me wait a moment and then get the public IP address once it's assigned:

A call\_aws

✓ EC2 instance "noor-test" successfully created!

#### Instance Details:

- **Instance ID:** i-0ae1ad17d605aea8a
- **Instance Name:** noor-test
- **Status:** running
- **Instance Type:** t3.micro
- **AMI:** ami-0157af9aea2eef346 (Amazon Linux 2023)
- **Public IP:** 3.228.20.183
- **Public DNS:** ec2-3-228-20-183.compute-1.amazonaws.com
- **Private IP:** 172.31.70.209
- **VPC ID:** vpc-09c00748cf2a20907
- **Subnet ID:** subnet-0f0e887cf1e484187
- **Availability Zone:** us-east-1f
- **Security Group:** sg-056963dc066519acf (default)

The instance is now running and accessible at the public IP address shown above.



perfect, now terminate this ec2 this you just created "noor-test".

I'll terminate the EC2 instance "noor-test" (i-0ae1ad17d605aea8a).

A call\_aws

I don't have permission to terminate EC2 instances. The IAM role `aws-api-mcp-execution-role` doesn't have the `ec2:TerminateInstances` permission.

To terminate this instance, you'll need to either:

1. Grant the permission by adding a policy to the `aws-api-mcp-execution-role` that allows `ec2:TerminateInstances`
2. Terminate it manually through the AWS Console or using credentials with appropriate permissions

The instance details for manual termination:

- Instance ID: i-0ae1ad17d605aea8a
- Instance Name: noor-test

Would you like me to help with anything else that's within my current permissions?



## Comprehensive Infrastructure Test

Try this more complex test that creates infrastructure:

Create a test infrastructure setup with the following:

1. Create a VPC named "mcp-vpc" with CIDR block 10.0.0.0/16
2. Create a public subnet in this VPC with CIDR block 10.0.1.0/24 in us-east-1a
3. Create an internet gateway and attach it to the VPC
4. Create a route table for the public subnet with a route to the internet gateway
5. Create a security group named "mcp-test-sg" in the VPC that allows:
  - SSH (port 22) from my current IP
  - HTTP (port 80) from anywhere

- HTTPS (port 443) from anywhere
6. Launch a t3.micro EC2 instance with:
- Amazon Linux 2023 AMI (use the latest available)
  - Instance name: "mcp-test-instance"
  - In the VPC and subnet created above
  - Using the security group created above
  - Using existing key pair "test-key"
  - With a public IP address assigned

After creation, show me the instance ID, public IP, and VPC ID.

**Expected behavior:** Claude will create all resources and provide you with the IDs. If any permission is missing, Claude will clearly state which operation failed.

I'll create this test infrastructure step by step. Let me start by creating the VPC and related networking components.

(A) call\_aws

I don't have the necessary permissions to create EC2 infrastructure. The IAM role I'm using (`aws-api-mcp-execution-role`) doesn't have permissions for EC2 operations like:

- `ec2:CreateVpc`
- `ec2:CreateSubnet`
- `ec2:CreateInternetGateway`
- `ec2:RunInstances`
- etc.

To complete this infrastructure setup, you'll need to either:

1. Grant additional permissions to the `aws-api-mcp-execution-role` IAM role by adding an EC2-related policy that allows:
  - VPC creation and management
  - Subnet creation
  - Internet Gateway operations
  - Security Group operations
  - EC2 instance launching
2. Use a different approach such as:
  - Using AWS CloudFormation or Terraform (which I could help you create the templates for)
  - Manually creating these resources through the AWS Console
  - Using an IAM user or role with appropriate EC2 permissions

Would you like me to create a CloudFormation template or Terraform configuration that you could deploy to create this infrastructure instead?

❑ ⌂ ⌂ ⌂ ⌂

## 6. Cost Breakdown

### AgentCore Pricing

Component	Cost
Runtime CPU	\$0.0895 per vCPU-hour
Runtime Memory	\$0.00945 per GB-hour
Gateway Tool Calls	\$0.005 per 1,000 invocations

### Monthly Cost Estimates

Usage Level	Sessions/Month	Runtime Cost	Total/Month
Light Testing	600	\$0.23	<b>\$0.24</b>
Moderate Use	3,000	\$1.13	<b>\$1.19</b>
Heavy Use	15,000	\$5.66	<b>\$5.96</b>

### Cost Notes

- I/O wait is free:** You only pay for active CPU time. When waiting for AWS API responses (typically 50-70% of time), you're not charged for CPU.
- LLM cost covered:** Your Claude Pro subscription covers the AI inference cost. No separate Bedrock model charges.
- AWS Free Tier:** New AWS customers get \$200 in AgentCore credits, covering 33-800+ months of testing.
- Resource costs separate:** EC2 instances, S3 storage, and other AWS resources are billed separately.

## 7. Troubleshooting

### Claude Desktop Says 'No Access to AWS Tools'

#### Possible Causes

5. Wrong config file name

```
# Check file exists and is named correctly
ls ~/.config/Claude/clade_desktop_config.json
```

## 6. UV not installed

```
# Verify UV is installed  
uvx --version  
# If not found, add to PATH  
source $HOME/.local/bin/env
```

## 7. Old chat window

Solution: Start a NEW conversation. Old chats don't load newly configured MCP servers.

## 8. Claude Desktop not fully restarted

```
# Force quit and restart  
killall claude
```

## Runtime Shows FAILED Status

### Check IAM Permissions

```
# Verify role has base permissions  
aws iam get-role-policy \  
  --role-name aws-api-mcp-execution-role \  
  --policy-name AgentCoreBasePermissions
```

Verify the policy includes ECR access, CloudWatch logs, and X-Ray permissions.

## Internal Error When Testing Endpoint

Error in console: {"code": -32603, "message": "Internal error"}

### Solution

The console test doesn't work with SigV4 authentication. This is normal. Check CloudWatch logs instead:

```
aws logs tail /aws/bedrock-agentcore/runtimes/YOUR_RUNTIME_ID --follow
```

If logs show server startup messages, the deployment is successful.

## Permission Denied Errors

When Claude tries to perform an action and gets an access denied error:

9. Check which operation failed (Claude will tell you)
10. Add that permission to your custom-aws-permissions.json
11. Update the IAM policy:

```
aws iam put-role-policy \  
  --role-name aws-api-mcp-execution-role \  
  --policy-name CustomAWSPermissions \  
  --policy-document file://custom-aws-permissions.json
```

12. Wait 30 seconds for IAM to propagate, then retry

## AWS Credentials Not Found

```
# Check credentials are configured  
aws sts get-caller-identity  
  
# For SSO, login first  
aws sso login --profile your-profile
```

## Quick Reference

### Key File Locations

- **Claude Desktop config:** `~/.config/Claude/clause_desktop_config.json`
- **CloudWatch logs:** `/aws/bedrock-agentcore/runtimes/YOUR_RUNTIME_ID`
- **IAM role:** `aws-api-mcp-execution-role`

### Useful Commands

```
# Check runtime status  
aws bedrock-agentcore-control get-agent-runtime \  
  --agent-runtime-id YOUR_ID --region us-east-1  
  
# View logs  
aws logs tail /aws/bedrock-agentcore/runtimes/YOUR_ID --follow  
  
# Check IAM policies  
aws iam list-role-policies --role-name aws-api-mcp-execution-role  
  
# Restart Claude Desktop  
killall claude
```

## **Support Resources**

- AWS Marketplace: <https://aws.amazon.com/marketplace/pp/prodview-lqqkwbcraxsgw>
- GitHub Issues: <https://github.com/awslabs/mcp/issues>
- MCP Proxy: <https://github.com/aws/mcp-proxy-for-aws>
- AgentCore Docs: <https://docs.aws.amazon.com/bedrock-agentcore/>