AWS CLI and boto3 Student Guide

Table of Contents

Introduction

This guide covers two essential tools for managing AWS resources:

- **AWS CLI**: Command-line interface for AWS services
- **boto3**: AWS SDK for Python

Both tools allow you to programmatically interact with AWS services, create resources, and automate cloud infrastructure management.

Prerequisites

- Basic understanding of AWS concepts (EC2, IAM, Security Groups)
- Linux/Unix terminal knowledge
- Python basics (for boto3 section)
- Active AWS account

AWS CLI Installation and Setup

Step 1: Download and Install AWS CLI

```bash
```

```
# Download AWS CLI v2
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

# Install unzip utility if not available
sudo apt install unzip

# Extract the installer
unzip awscliv2.zip

# Install AWS CLI
sudo ./aws/install

# Verify installation
aws --version
```

Expected Output:

```
aws-cli/2.x.x Python/3.x.x Linux/x86_64 botocore/2.x.x
```

Step 2: Clean Up Installation Files

```bash
rm -rf awscliv2.zip aws/
```

📚 **Additional Resources:**

- Official AWS CLI Installation Guide

AWS Credential Configuration

Step 1: Create IAM User with Admin Access

1. **Log into AWS Console**
   - Navigate to IAM service
   - Go to "Users" section
2. **Create New User**
   - Click "Create user"
   - Enter username (e.g., `cli-admin-user`)
   - Select "Programmatic access"

3. **Attach Permissions**
   - Attach existing policy: `AdministratorAccess`
   - ⚠️ **Note:** In production, use least-privilege principle
4. **Generate Access Keys**
   - Go to "Security credentials" tab
   - Click "Create access key"
   - Choose "Command Line Interface (CLI)"
   - Save the credentials securely

Step 2: Configure AWS CLI

```bash
aws configure
```

Interactive Prompts:

```
AWS Access Key ID [None]: AKIAT4ID72Q5HDGVB3VO
AWS Secret Access Key [None]: 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
Default region name [None]: il-central-1
Default output format [None]: json
```

Verification:

```bash
# Test configuration
aws sts get-caller-identity
```

Creating EC2 Resources with AWS CLI

Step 1: Create Key Pair

```bash
# Create a new key pair
aws ec2 create-key-pair --key-name key-cli-01
```

Expected Output:

```json
{
  "KeyPairId": "key-0e31800e15163ea6d",
  "KeyName": "key-cli-01",
  "KeyFingerprint": "eb:35:6b:f9:b1:08:11:21:01:09:8a:6e:c9:7e:a5:ea:5e:07:23:bb",
  "KeyMaterial": "-----BEGIN RSA PRIVATE KEY-----\n[PRIVATE KEY CONTENT]\n-----END RSA PRIVATE KEY-----"
}
```

Save Private Key:

```bash
# Create the key file in ~/.ssh directory
cat > ~/.ssh/key-cli-01.pem << 'EOF'
-----BEGIN RSA PRIVATE KEY-----
[PASTE THE KeyMaterial CONTENT HERE]
-----END RSA PRIVATE KEY-----
EOF

# Set proper permissions
chmod 400 ~/.ssh/key-cli-01.pem
```

📚 **Reference:** AWS CLI Key Pair Documentation

Step 2: Create Security Group

```bash
# Create security group
aws ec2 create-security-group \
  --group-name sec-grp-cli-01 \
  --description "My security group created via CLI"
```

Expected Output:

```json
{
  "GroupId": "sg-0a4bbf87c1c16d60c",
  "SecurityGroupArn": "arn:aws:ec2:il-central-1:266833220666:security-group/sg-0a4bbf87c1c16d60c"
}
```

Add SSH Access Rule:

```bash
# Get your public IP
MY_IP=$(curl -s ifconfig.me)

# Add SSH rule to security group
aws ec2 authorize-security-group-ingress \
  --group-id sg-0a4bbf87c1c16d60c \
  --protocol tcp \
  --port 22 \
  --cidr ${MY_IP}/32
```

📚 **Reference:** AWS CLI Security Group Documentation

Step 3: Launch EC2 Instance

```bash
# Launch EC2 instance
aws ec2 run-instances \
  --image-id ami-0b00c1e12b92531a8 \
  --count 1 \
  --instance-type t3.micro \
  --key-name key-cli-01 \
  --security-group-ids sg-0a4bbf87c1c16d60c \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=CLI-Created-Instance}]'
```

📚 **Reference:** AWS CLI EC2 Instances Documentation

Step 4: Connect to EC2 Instance

```bash
# Get instance public DNS (replace instance-id with your actual ID)
aws ec2 describe-instances \
  --instance-ids i-1234567890abcdef0 \
  --query 'Reservations[0].Instances[0].PublicDnsName' \
  --output text

# SSH to instance
ssh -i "~/.ssh/key-cli-01.pem" ec2-user@ec2-xx-xx-xxx-xx.il-central-1.compute.amazonaws.com
```

---

boto3 Setup and Usage

Step 1: Python Environment Setup

```bash
# Create project directory
mkdir boto3-project
cd boto3-project

# Create Python virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate

# Install boto3
pip install boto3

# Verify installation
pip list | grep boto3
```

Step 2: Create EC2 Instance with boto3

Create a file named `ec2-create.py`:

```python
```

```python
import boto3
import json

def create_ec2_instance():
    """
    Create an EC2 instance using boto3
    """
    # Initialize the EC2 client
    ec2_client = boto3.client('ec2')

    try:
        response = ec2_client.run_instances(
            ImageId='ami-0b00c1e12b92531a8',  # Amazon Linux 2023
            MinCount=1,
            MaxCount=1,
            InstanceType='t3.micro',
            KeyName='key-cli-01',  # Existing key pair
            SecurityGroupIds=[
                'sg-0a4bbf87c1c16d60c',  # Existing security group
            ],
            TagSpecifications=[
                {
                    'ResourceType': 'instance',
                    'Tags': [
                        {
                            'Key': 'Name',
                            'Value': 'MyBoto3Instance'
                        },
                        {
                            'Key': 'Environment',
                            'Value': 'Development'
                        }
                    ]
                },
            ]
        )

        # Extract instance information
        for instance in response['Instances']:
            instance_id = instance['InstanceId']
            instance_type = instance['InstanceType']
            state = instance['State']['Name']
```

```
            print(f"✅ Successfully launched EC2 Instance!")
            print(f"   Instance ID: {instance_id}")
            print(f"   Instance Type: {instance_type}")
            print(f"   State: {state}")

            return instance_id

    except Exception as e:
        print(f"✗ Error launching EC2 instance: {e}")
        return None

def get_instance_info(instance_id):
    """
    Get detailed information about an EC2 instance
    """
    ec2_client = boto3.client('ec2')

    try:
        response = ec2_client.describe_instances(InstanceIds=[instance_id])
        instance = response['Reservations'][0]['Instances'][0]

        print(f"\n📋 Instance Details:")
        print(f"   Instance ID: {instance['InstanceId']}")
        print(f"   Public IP: {instance.get('PublicIpAddress', 'N/A')}")
        print(f"   Private IP: {instance.get('PrivateIpAddress', 'N/A')}")
        print(f"   State: {instance['State']['Name']}")

    except Exception as e:
        print(f"✗ Error getting instance info: {e}")

if __name__ == "__main__":
    print("🚀 Creating EC2 instance with boto3...")
    instance_id = create_ec2_instance()

    if instance_id:
        # Wait a moment for instance to initialize
        import time
        time.sleep(10)
        get_instance_info(instance_id)
```

Step 3: Run the Script

bash

```
# Execute the script
python3 ec2-create.py
```

Expected Output:

```
🚀 Creating EC2 instance with boto3...
✅ Successfully launched EC2 Instance!
  Instance ID: i-05157ca20e83e730c
  Instance Type: t3.micro
  State: pending

📋 Instance Details:
  Instance ID: i-05157ca20e83e730c
  Public IP: 51.17.251.97
  Private IP: 172.31.10.197
  State: running
```

Step 4: Deactivate Virtual Environment

```bash
# When finished working
deactivate
```

📚 **Reference:** boto3 EC2 Documentation

Comparison: CLI vs boto3

Aspect	AWS CLI	boto3
Language	Shell/Bash	Python
Use Case	Quick operations, scripts	Complex applications
Error Handling	Shell exit codes	Python exceptions
Data Processing	JSON parsing with jq	Native Python objects
Integration	Shell scripts	Python applications
Learning Curve	Easier for simple tasks	Better for complex logic

When to Use Each Tool

Use AWS CLI when:

- Writing shell scripts
- One-off administrative tasks
- CI/CD pipelines
- Quick resource queries

Use boto3 when:

- Building Python applications
- Complex error handling needed
- Data processing and analysis
- Building web applications with AWS integration

Best Practices and Security

Security Best Practices

1. **Credential Management**

```bash
# Use IAM roles when possible
# Store credentials in ~/.aws/credentials
# Never commit credentials to version control
```

2. **Least Privilege Principle**
   - Create specific IAM policies instead of using `AdministratorAccess`
   - Regularly audit and rotate access keys

3. **Resource Tagging**

```bash
# Always tag resources for better management
aws ec2 run-instances \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Environment,Value=Dev},{Key=Project,Value=MyApp}]
```

Cost Management

1. **Clean Up Resources**

```bash
```

```
# Terminate instances
aws ec2 terminate-instances --instance-ids i-1234567890abcdef0

# Delete security groups
aws ec2 delete-security-group --group-id sg-0a4bbf87c1c16d60c

# Delete key pairs
aws ec2 delete-key-pair --key-name key-cli-01
```

2. **Use Free Tier Eligible Resources**
   - t3.micro instances
   - Monitor usage in AWS billing console

Troubleshooting

Common Issues and Solutions

1. **Authentication Errors**

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

# Reconfigure if needed
aws configure
```

2. **Permission Denied (SSH)**

```bash
# Fix key permissions
chmod 400 ~/.ssh/key-cli-01.pem
```

3. **boto3 Import Error**

```bash
# Ensure virtual environment is activated
source venv/bin/activate

# Reinstall if needed
pip install --upgrade boto3
```

4. **Region Mismatch**

```bash
# Check your default region
aws configure get region

# Use specific region in commands
aws ec2 describe-instances --region us-east-1
```

Debugging Tips

1. **Enable Debug Mode**

```bash
# For AWS CLI
aws ec2 describe-instances --debug
```

2. **boto3 Logging**

```python
import boto3
import logging

# Enable debug logging
boto3.set_stream_logger('', logging.DEBUG)
```

Summary

This guide covered:

- Installing and configuring AWS CLI
- Setting up IAM credentials
- Creating EC2 resources via CLI
- Setting up boto3 Python environment
- Creating EC2 instances programmatically
- Best practices for security and cost management

Next Steps

- Explore other AWS services (S3, RDS, Lambda)
- Learn about AWS CloudFormation for infrastructure as code

- Practice automation with both CLI and boto3
- Study AWS security best practices

Additional Resources

- AWS CLI User Guide
- boto3 Documentation
- AWS Free Tier
- AWS Well-Architected Framework