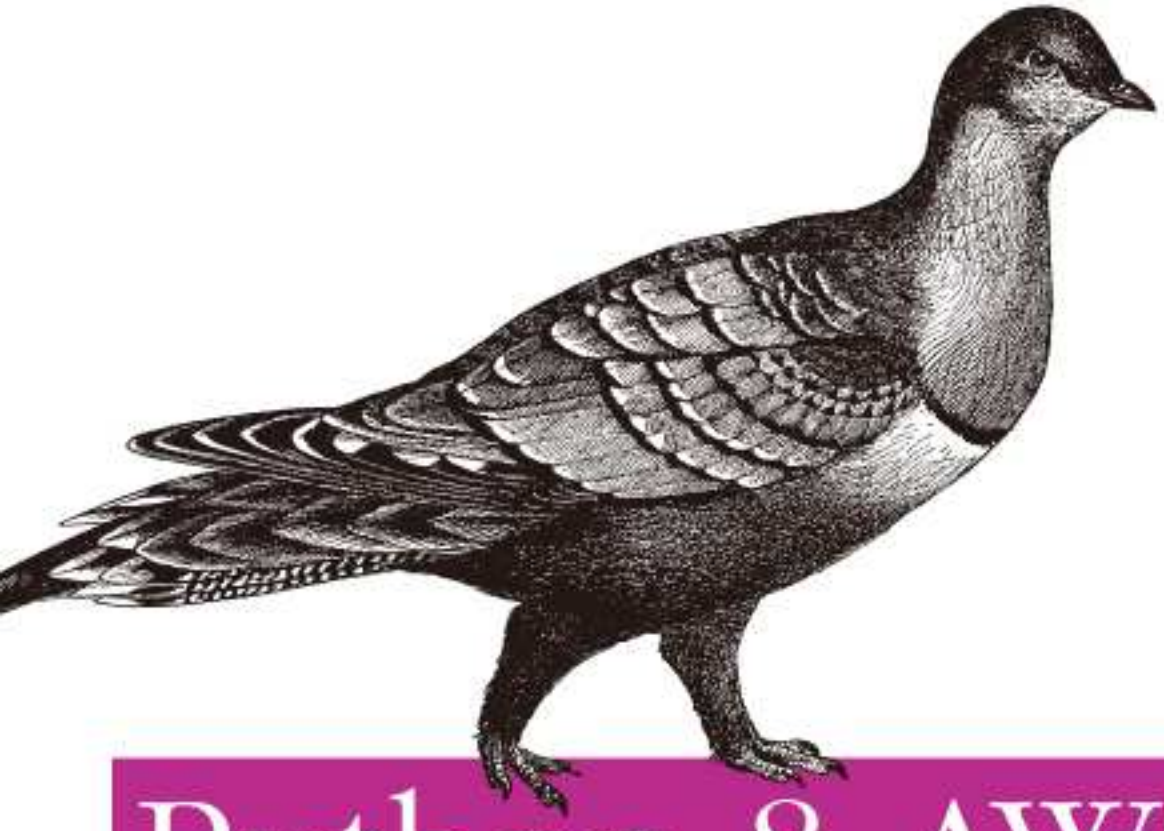
Managing Your Cloud with Python and Boto



Python & AWS クックブック

Mitch Garnaat 著
成田 昇司
株式会社トップスタジオ 訳

O'REILLY®
オライリー・ジャパン

Python & AWS クックブック

Mitch Garnaat　著
成田 昇司
株式会社トップスタジオ　訳

O'REILLY®
オライリー・ジャパン

Python and AWS Cookbook

Mitch Garnaat

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

まえがき

筆者が初めて Amazon Web Services（AWS）を体験したのは、2006 年 3 月 14 日のことでした。それに先立って Simple Storage Service（S3）と呼ばれる Web ベースの新しいストレージサービスを発表したプレスリリースを見たときには、Amazon がそのようなサービスを提供するとは何と奇妙なことだろうと思ったのを今でも覚えています。それはともかく、筆者はサインアップしてアカウントを取得し、**ドキュメント**を読み始めました。

筆者は S3 に圧倒されました。単純で、手ごろな価格のモデル。洗練された REST API。実質的に無制限のストレージ容量。驚くべきことです。これを何かさらに改善できるとするなら、と筆者は自問し——Python インタフェースこそがそれだと考えました。その日、後に boto ライブラリ（https://github.com/boto/boto）となるもののコーディングを始めました。このライブラリは、Amazon Web Services とのインタフェースとして本書で使用します。

私は今でも確信していますが、Python は、AWS を始めとするクラウドサービスを対話的に操作するのに最適な言語です。すばらしい標準ライブラリがすべての Python のインストールに付属しており（「バッテリー同梱」!）、これまでに蓄積された膨大な数のモジュールを Python Cheese Shop（http://pypi.python.org/）†を介して簡単にダウンロードでき、要求を試みれば即座に結果が表示されるというようにクラウドサービスを対話的に操作できる機能が備わっています。これらの特徴があいまって、アプリケーションを開発し、クラウドベースのインフラストラクチャを制御するための、強力で楽しい方法を提供してくれます。

何か新しいことを学習する最善の方法は多くの例を見るほかないということに、筆者はいつも気づかされます。本書は、短いながらも、その点に焦点を当てながら、Elastic Compute Cloud（EC2）および Simple Storage Service（S3）に関連する多くの共通する問題に対する解法を（Python と boto を使用して）紹介します。これが読者の役に立つことを期待しています。

† Python Cheese Shop は以前の名称で、現在は PyPI という名称に変更されている。

本書で使用されている表記規則

本書では以下に示すような活字の表記規則を使用しています。

Bold（太字）
: 新しい用語、URL、電子メールアドレス、ファイル名、およびファイル拡張子を表します。

`Constant Width`（等幅）
: プログラムコードに使用します。さらに、本文の中で変数名や関数名、データベース、データ型、環境変数、ステートメント、キーワードなどのプログラム要素を参照する場合にも使用します。

`Constant Width Bold`（等幅の太字）
: ユーザがそのまま入力するコマンドなどのテキストを示します。

`Constant-width Italic`（等幅の斜体）
: ユーザが指定する値または前後関係に応じて決定される値に置き換えるテキストを示します。

このアイコンは、ヒント、提案、または一般的なメモを意味します。

このアイコンは、警告または注意を表します。

サンプルコードの使用について

本書は、読者の皆さんが仕事を遂行するための役に立つことを目的にしています。一般的に言えば、本書に含まれているコードは、皆さんのプログラムやドキュメントで使用できます。本書の関係者に連絡して許可を得る必要はありません。ただし、大量のコードをまとめて再利用する場合は別です。例えば、本書に含まれるコードをいくつか使用してプログラムを作成する場合には、許可を得る必要がありません。しかし、O'Reilly 社の書籍から引用したサンプルを CD-ROM にコピーして販売したり配布したりする場合には、必ず許可を得る必要があります。また、質問に答える際に本書を引き合いに出してサンプルコードを引用する場合には、許可を得る必要はありません。それに対して、かなり大量のサンプルコードを本書から皆さんの製品のドキュメントに取り入れる場合には、必ず許可を得る必要があります。

サンプルコードの帰属を明示していただければ感謝しますが、要求はしません。出典を示す際にはタイトル、著者、発行者などを含めます。例えば「『Python & AWS クックブック』、Mitch Garnaat 著、オライリー・ジャパン刊」のような形です。

サンプルコードの使用法が適正な使用法から逸脱していたり上記の許可される範囲を超えていると思われる場合は、permissions@oreilly.com 宛てにメールでご連絡下さい（英語のみ）。

お問合せ先

本書に関するご意見やご質問は、下記の出版社にお寄せ下さい。

株式会社オライリー・ジャパン
〒 160-0002　東京都新宿区坂町 26 番地 27 インテリジェントプラザビル 1F
電話：　03-3356-5227
FAX：　03-3356-5261

本書専用の Web ページを用意し、正誤表、サンプル、その他の追加情報などを掲載しています。このページには次の URL でアクセスできます。

http://shop.oreilly.com/product/0636920020202.do（英語）
http://www.oreilly.co.jp/books/9784873115436（日本語）

本書に関するご意見や技術的なご質問は、次のアドレス宛の電子メールでも受け付けています。

bookquestions@oreilly.com（英語）
japan@oreilly.co.jp（日本語）

当社の書籍、講座、カンファレンス、ニュースなどに関する詳細な情報については、当社の Web サイトをご覧下さい。

http://www.oreilly.com（英語）
http://www.oreilly.co.jp（日本語）

目　次

1章
一般情報

Python に関するクイックメモ

本書に掲載されているサンプルでは、Python 2.7.1 を使用していますが、2.5.x から 2.7.x までの任意のバージョンの Python で動作するはずです†。botoライブラリについては、Python 3.x への移植と完全なテストをまだ行っていませんが、近い将来に移行する計画になっています。

Python のすべてのバージョンは、ソースコードの形式と多くの一般的なプラットフォーム用にコンパイル済みの状態の両方が、http://python.org で公開されています。

boto のインストール

本書のサンプルを実行するには、botoのバージョン 2.1 以降が必要です。 botoをインストールする際には、いくつかの選択肢が用意されています。

github.com からのダウンロードとインストール

botoプロジェクトは、github（https://github.com/）をソースコードリポジトリとして使用しています。したがって、github のリポジトリのクローンを自分のコンピュータ上に作成すれば、クローンされた配布物から botoをインストールできます。この方法では、 botoに加えられた最新の変更内容にいつでもアクセスすることができます。その中には最新の機能はもちろんですが、最新のバグも含まれます。したがって、そのような危険を考慮に入れた上で、次に示す手順に従ってインストールするかどうかを自分で判断する必要があります。

```
% git clone https://github.com/boto/boto
% cd boto
% sudo python setup.py install
```

手動による boto のダウンロードとインストール

Python Cheese Shop は、Python パッケージの公式リポジトリです。Cheese Shop（http://pypi.python.org/、PyPI とも呼ばれます）に移動して botoを検索すると、 **図 1-1**に示すようなページが表示さ

† 日本語版のサンプルコードは Python 2.7.3 で動作確認をしました。

れます（図に示したのはバージョン 2.0 ですが、実際にはバージョン 2.3 となっているはずです）。

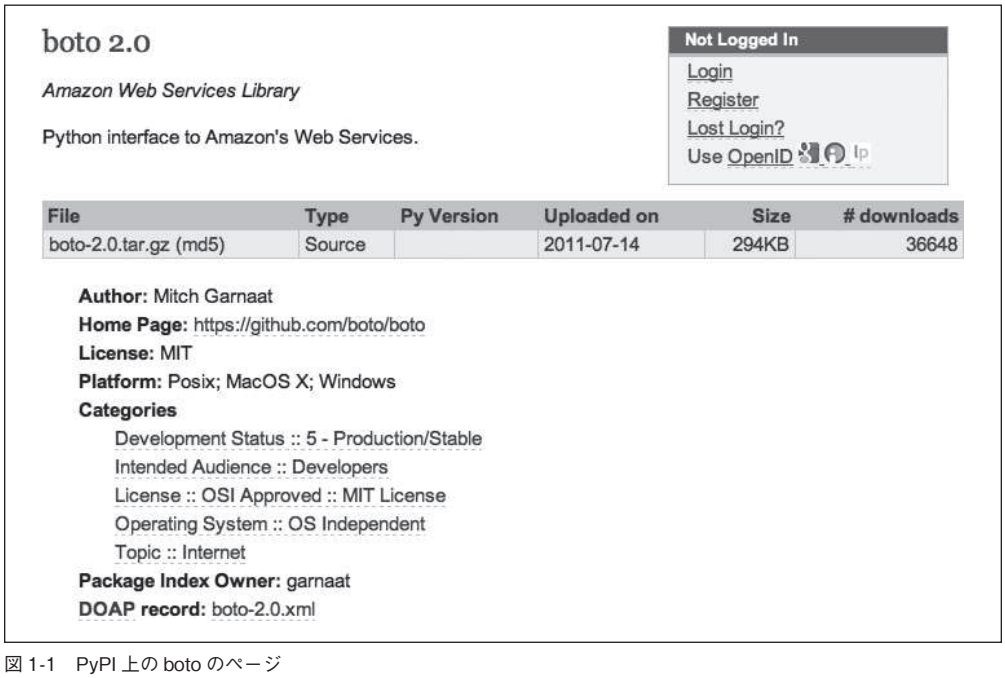


図 1-1　PyPI 上の boto のページ

boto-2.3.tar.gzのリンクをクリックすると、 botoのソースパッケージを含んだ状態で圧縮された tar ボールがダウンロードされます。このファイルを自分のコンピュータに保存し、次の手順に従ってソースパッケージからインストールします。

```
% tar xfz boto-2.3.tar.gz
% cd boto-2.3
% sudo python setup.py install
```

easy_install による boto のインストール

easy_installユーティリティを使用すると、Python パッケージの検索、インストール、アップグレード、およびアンインストールを（その名のとおり）簡単に行うことができます。easy_installの優れた機能を使用できるようにするには――最初にこれをインストールする必要があります。setuptoolsパッケージ（easy_installはその一部です）のインストールに関する詳細な手順については、http://pypi.python.org/pypi/setuptools に記載されています。多くの Linux ディストリビューションにも setuptoolsのパッケージが含まれています。例えば、Ubuntu の場合なら、次に示すコマンドを実行します。

```
% sudo apt-get install python-setuptools
```

Fedora や CentOS を始めとする yum 対応のディストリビューションであれば、次に示すコマンドを実行します。

```
% sudo yum install python-setuptools
```

easy_installを自分のコンピュータにセットアップした後ならば、botoのインストールは、次に示すように簡単に行えます。

```
% sudo easy_install boto
```

pip による boto のインストール

pipユーティリティも、Python パッケージの検索、インストール、アップグレード、およびアンインストールを行うためのツールです。筆者の経験では、easy_installと pipのどちらも十分に機能します。したがって、少し調べてみて、どちらを使用するかご自身で決めるとよいでしょう。あるいは、両方使用するという手もあります。pipのインストールに関する詳細な手順については、http://pypi.python.org/pypi/pip に記載されています。pipを自分のコンピュータにセットアップした後であれば、botoのインストールは、同じように簡単に行えます。

```
% sudo pip install boto
```

virtualenv による boto のインストール

インストール方法の選択肢を長々と紹介してきましたが、最後に紹介するのは、実際にはインストールツールではなく、むしろ独立した Python の環境を非常に手軽に作成するための方法です。virtualenvを使用すると、自己完結した Python の独立環境を自分のワークステーション上にいくつでも作成することができます。これらの仮想環境にインストールしたパッケージは、システムの Python パッケージやその他の仮想環境に影響を与えません。筆者は、筆者のラップトップ上で、何か 1 つのプロジェクトでの変更や誤りが他のプロジェクトや全体環境に及ぶ心配なしに、多くの異なるプロジェクトを進行できることがとても有用だと感じています。おまけに、一度 virtualenvをインストールして環境を作成してしまえば、それらの環境内での easy_installと pipの両方に自動でアクセスできるようになり、環境内にソフトウェアをインストールする際にスーパーユーザや管理者権限を必要としません。

virtualenvのインストールに関する詳細な手順については、http://pypi.python.org/pypi/virtualenv に記載されています。virtualenvを自分のコンピュータにセットアップした後であれば、次に示すコマンドによって仮想環境をセットアップできます。

```
% virtualenv paws
```

仮想環境を作成するときには、当然ながら任意の名前を付けることが可能です。それからその仮想環境を有効にして、botoをインストールできます †。

† Windows 環境の場合、実行ファイルは仮想環境の Script フォルダ以下に作成されます。

```
% cd paws
% source bin/activate
% pip install boto
```

paramiko のインストール

paramikoパッケージは、リモートマシンとの間でセキュリティで保護された接続を確立するための SSH2 プロトコルを実装しています。paramikoをインストールしなくても botoを使用することは可能ですが、本書で紹介する EC2 のレシピの中には paramikoパッケージに依存しているものもあるので、インストールすることをお勧めします。手動でインストールする場合は、http://www.lag.net/paramiko/ に記載されている手順に従って下さい。あるいは、すでに easy_installまたは pipをインストールしてある場合は、easy_install paramikoまたは pip install paramikoというコマンドを実行します。

euca2ools のインストール

インストールすることが望ましい最後のパッケージは、euca2oolsです。このパッケージは Eucalyptus チームによって開発されたものであり、AWS で提供されているツールと互換性のあるコマンドラインツールを提供します。euca2oolsは Python で書かれており、 botoを基盤として構築されています。クラウドインフラストラクチャの管理に役立つ一連の優れたツールを提供するとともに、学習や拡張の素材に適したサンプルコードも提供しています。

euca2oolsは、多くの Linux ディストリビューションでパッケージ化されてきました。Ubuntu の場合には、sudo apt-get -y euca2oolsというコマンドでインストールできます。yum 対応のディストリビューションの場合には、sudo yum install euca2oolsというコマンドでインストールできます。また、最新バージョンのソースコードは https://launchpad.net/euca2ools で入手することができ、パッケージ化されたソースリリースは http://open.eucalyptus.com/downloads からダウンロードできます。

Amazon Web Services の使用開始

AWS アカウントの作成

Amazon Web Services を使用するには、まず最初にサインアップしてアカウントを取得する必要があります。インターネットで http://aws.amazon.com/ のページを開き、［サインアップ］ボタンをクリックします。Amazon.com のアカウントをすでに持っていて、AWS の操作をそのアカウントに関連付けたい場合は、単にそれらの認証情報を使用してログインします。あるいは、お望みなら、AWS の操作専用のアカウントを新しく作成することもできます。

AWS へのサインアップに関する詳細な手順については、RightScale によって提供されている**チュートリアル**（http://support.rightscale.com/1._Tutorials/01-RightScale/1.5_Sign-up_for_AWS、**英語**）を参照して下さい†。

自分のアカウントが最低でも EC2 サービスと S3 サービスに対して有効になっていることを確認します。上記のチュートリアルには各種のサービスへのサインアップに関する詳細な手順が記載されています。

† 日本語のドキュメントへのリンクが http://aws.amazon.com/jp/aws-first-step/に掲載されています。

アカウントを作成し終えると、次に示すようなさまざまな認証情報がそのアカウントに関連付けられます。

AWS アカウント認証情報

AWS Web ポータルと AWS Management Console にログインする際に使用する認証情報です。電子メールアドレスとパスワードで構成されます。これらの認証情報は、後で検討する他の認証情報のすべてに対するアクセスを制御します。したがって、このアカウントには強力なパスワードを選択し、そのパスワードをなるべく短い期間で積極的に変更することが非常に重要です。

AWS アカウント番号

自分の AWS アカウントに関連付けられた一意な番号で、12 桁の 10 進数で表されます。これから論じる他の認証情報とは異なり、この情報は秘密事項ではありません。自分のアカウント番号を最も簡単に調べるには、AWS ポータルにログインした後で、Web ページの右上隅のほうを見ます。すると、**図 1-2**に示すような表示を確認できます。

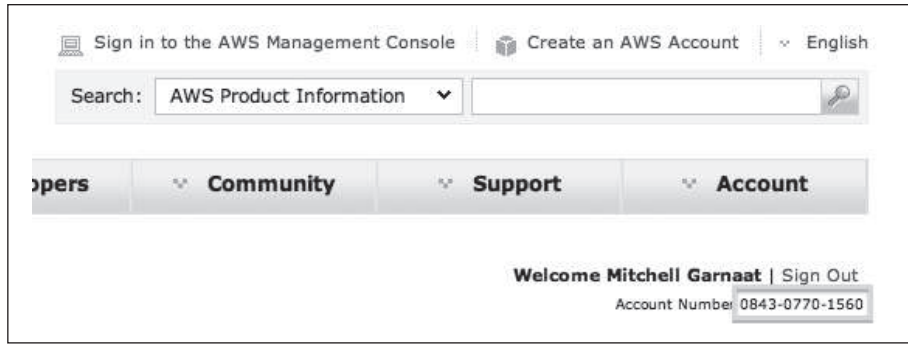


図 1-2　AWS アカウント番号の確認

アカウント番号は公開識別子であり、主に AWS 内でリソースを共有する場合に使用します。例えば、EC2 の中で AMI を作成して、その AMI を公開することなく特定のユーザと共有したい場合に、そのユーザのアカウント番号を AMI に関連付けられたユーザ ID のリストに追加することが必要になります。ここに混乱を引き起こす可能性のある問題がひとつあります。アカウント番号が表示されるときには、4 つの数字ごとに**ハイフン**によって 3 つのグループに分けられています。ところが、API を介して使用するときには、これらのハイフンを取り去らなければならないのです。

AccessKeyID および SecretAccessKey

これらのアクセス識別子は、AWS におけるすべての API アクセスの核心にあるものです。各 AWS サービスに対して REST または Query API のリクエストを行うときには、そのすべてにおいて自分の AccessKeyID（アクセス鍵 ID）を渡して自分が誰であるかを明確にする必要があります。さらに API では、**シグネチャ**を計算してリクエストに含めることも必要になります。

シグネチャを計算するには、さまざまなリクエストの要素（例えば、タイムスタンプ、リクエ

スト名、パラメータなど）を連結して署名する文字列（StringToSign）を構成し、次に自分のSecretAccessKey（秘密アクセス鍵）を鍵として使用してStringToSignのHMACを計算することによって**シグネチャ**を作成します。

リクエストがAWSに受信されたときには、同サービスによって同じようにStringToSignが連結され、次にリクエストに含めて送られたAccessKeyIDとAWSによって関連付けられたSecretAccessKeyに基づいて**シグネチャ**が計算されます。それらが一致すれば、リクエストは認証されます。一致しなければ、拒絶されます。

アカウントに関連付けられたAccessKeyIDは変更できませんが、SecretAccessKeyは、AWSポータルを使用していつでも再生成できます。SecretAccessKeyは、認証メカニズム全体の基礎となっている共有秘密鍵なので、自分のSecretAccessKeyが侵害された恐れがある場合には再生成すべきです。

X.509 証明書

アカウントに関連付けられるもうひとつのアクセス識別子は、X.509証明書です。自分で独自の証明書を用意することもできますが、AWSによって証明書を生成することもできます。この証明書は、SOAPバージョンのAWS APIを使用するときに、リクエストを認証するために使用することができ、さらにEC2において独自のS3ベースのAMIを作成するときにも使用されます。基本的に、AMIをバンドルしたときに作成されたファイルは、アカウントと関連付けられたX.509証明書を使用して暗号で署名されています。したがって、誰かがバンドルされたAMIを不正に変更しようという気になれば、シグネチャは破られ容易に検知されてしまいます。

SOAP APIを使用する場合、X.509証明書は、セキュリティの見地からも非常に重要です。その重要性は先ほど述べたSecretAccessKeyと変わらないので、同じくらい慎重に管理する必要があります。たとえ皆さんがSOAPを使用しなくても、ハッカーには可能なのだということを忘れないで下さい。

SSH 鍵

説明する必要のある認証情報の最後は、EC2インスタンスへのSSHアクセスに使用される公開鍵と秘密鍵のペアです。デフォルトでは、EC2インスタンスは公開鍵認証によるSSHアクセスのみを許可します。あなたが自分で使用するために作成するすべてのAMIにおいてもこの方針に従うことを強くお勧めします。SSHの鍵ペアはAWS ConsoleとAPIを使用して生成できますし、既存の鍵ペアをインポートすることもできます。実行中のインスタンスにアクセスする必要がある組織内のすべての個人に対して鍵ペアをそれぞれ作成し、それらのSSH鍵を慎重に保護しなければなりません。

botoにおけるAWS認証情報の管理

サインアップしてAWSアカウントを取得し、さらに認証情報を取得し終えたら、それらをbotoで使用できるようにする必要があります。この方法は数通り用意されています。

認証情報を明示的に boto に渡す

いずれかのAWSサービスにアクセスするには、そのたびにbotoによってそのサービスへの接続を確立する必要があります。そのようなときには、次に示すように自分のアクセス鍵IDと秘密アクセス鍵を明示的に渡すことができます。botoではサービスとの通信を行うときにそれらの認証情報を使用します。

```
% python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto
>>> ec2 = boto.connect_ec2(aws_access_key_id='my_access_key',
aws_secret_access_key='my_secret_key')
```

大半の人は、接続するたびにこのように入力するのはわずらわしいとすぐに思うでしょう。

認証情報を環境変数に保存する

上記のようにして接続するときに、認証情報が明示的にbotoに渡されない場合は、AWS_ACCESS_KEY_IDと AWS_SECRET_ACCESS_KEYという環境変数がユーザの環境内で定義されているかどうかがbotoによって確認されます。これらの環境変数が定義されている場合は、botoによってそれらの値がアクセス鍵と秘密鍵として使用されます。

認証情報を boto の構成ファイルに保存する

認証情報が明示的にbotoに渡されない場合、およびユーザの環境内でそれらが見つからない場合は、botoの構成ファイルの内容からそれらを検索することが試みられます。デフォルトで botoによって検索されるのは、/etc/boto.cfgファイルと ~/.botoファイルに含まれる構成情報です。自分の構成情報を別の場所に保存したい場合は、BOTO_CONFIG環境変数にその構成ファイルのパスを設定します。すると、botoはそのファイルから読み込みます。自分の認証情報を botoの構成ファイルに追加するには、次に示すようなセクションを追加します。

```
[Credentials]
aws_access_key_id = your_access_key
aws_secret_access_key = your_secret_key
```

認証情報をbotoの構成ファイルに保存した後であれば、ユーザが botoを使用するたびに、認証情報は自動的に使用されます。AWS認証情報を取り扱うには、これが最も便利な方法です。ただし、自分のAWS認証情報を自分のbotoの構成ファイルに保存する場合は、そのファイルの読み取りを自分だけに許可するようにファイル保護の設定を変えるのが賢明です。

クイックテスト

うまくいけばこの時点で、皆さんはbotoをインストールし終えて、さらに自分のAWSアカウントを作成し、認証情報を環境変数またはbotoの構成ファイルに保存し終えていることでしょう。次の段階に進む前に、簡単なテストでそれらが正常に機能していることを確認してみましょう。

```
% python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto
>>> ec2 = boto.connect_ec2()
>>> ec2.get_all_zones()
[RegionInfo:eu-west-1, RegionInfo:us-east-1, RegionInfo:ap-northeast-1,
RegionInfo:us-west-1, RegionInfo:ap-southeast-1,RegionInfo:us-west-2,
RegionInfo:sa-east-1]
>>>
```

この短いテストを実行したときにほぼ同じ出力が得られれば、準備万端で次のレシピに進むことができます。そうでない場合には、これまでの手順をもう一度確認して下さい。それでもうまくいかない場合には、botoユーザグループ（http://groups.google.com/group/boto-users）に質問を投稿してみて下さい。

Eucalyptusによるbotoの使用

Eucalyptusとは何か

http://open.eucalyptus.com/learn/what-is-eucalyptusでは次のように説明されています。

> Eucalyptusは、業務用のプライベートクラウドの作成を可能にします。その際、組織の既存のITインフラストラクチャを再編する必要はありませんし、専用のハードウェアを導入する必要もありません。Eucalyptusが実装するIaaS（Infrastructure as a Service）プライベートクラウドは、Amazon EC2およびAmazon S3と互換性のあるAPIによってアクセス可能です。この互換性のおかげで、すべてのEucalyptusクラウドは、パブリッククラウドからコンピュートリソースを引き出すことが可能なハイブリッドクラウドに変化させることができます。さらにEucalyptusは、EC2およびS3の事実上標準に準拠した豊富なツールやアプリケーションと互換性があります。

簡単に言えば、Eucalyptusは、独自のAWS互換ミニクラウド（あるいは、十分なハードウェアがある場合にはマクシクラウド）のセットアップを可能にします。本書で紹介するAWS用のレシピのほとんどすべては、Eucalyptusでも動作します。したがって、ハードウェアに余裕がある場合には、自分のネットワークの快適性と安全性を維持したままで、すべてが正常に機能するか試してみることができます。

Eucalyptusの入手とインストール

Eucalyptusをインストールすることに関心がある場合には、FastStart（http://go.eucalyptus.com/Download-FastStart.html）を試してみるとよいでしょう。FastStartを使用すると、可能な限り少ない操作でEucalyptusをすばやくインストールして実行することができます。

Eucalyptus Community Cloudの使用

特別なソフトウェアをまったくインストールすることなく、Eucalyptusが正常に機能するか試してみることも可能です。EucalyptusではEucalyptus Community Cloudと呼ばれるサンドボックスを中心とした環境が提供されており、その中でEucalyptusのクラウドソフトウェアを試用できます。詳細については、http://open.eucalyptus.com/try/community-cloudを参照して下さい。

botoにおけるEucalyptus認証情報の管理

Eucalyptusを起動して実行状態にした後は、botoを使用してシステムに簡単に接続できます。最初に自分のbotoの構成ファイルを編集して、次に示すようにEucalyptusのホスト名と**認証情報**を追加します。

```
[Credentials]
euca_access_key_id = 自分のEucalyptusアクセス鍵
euca_secret_access_key = 自分のEucalyptus秘密鍵

[Boto]
eucalyptus_host = "使用するEucalyptus CLCのDNSの名前またはIPアドレス"
walrus_host = "使用するWalrusのDNSの名前またはIPアドレス"
```

情報をbotoの構成ファイルに入力した後は、次のように操作することでEucalyptus（EC2互換のサービス）とWalrus（S3互換のサービス）に接続できます。

```
% python
>>> import boto
>>> euca = boto.connect_euca()
>>> walrus = boto.connect_walrus()
```

Google Cloud Storageによるbotoの使用

Google Cloud Storageとは何か

Google Cloud Storage for Developers（https://developers.google.com/storage/）は、Googleのインフラストラクチャの中で自分のデータを保存したり自分のデータにアクセスしたりするためのRESTfulな（つまり、RESTの原則に従った）サービスです。Google Cloud StorageとS3は、それぞれ独自の機能をいくつか提供していますが、機能性と基本的なAPIの両方において共通している箇所もかなり見られます。Googleの開発者は、botoの中でGoogle Cloud Storageの機能に完全にアクセスすることを許容したとい

う点で、botoに対してかなり大きな貢献をしてくれました。本書で紹介するS3レシピの多くは、Google Cloud Storageでも正常に機能します。

botoにおけるGoogle Cloud Storageの認証情報の管理

Google Cloud Storageは独立したサービスなので、一連の認証情報を独自に管理します。これらの認証情報は、次に示すようにbotoの構成ファイルに追加できます。

```
[Credentials]
gs_access_key_id = 自分のGoogle Cloud Storageアクセス鍵
gs_secret_access_key = 自分のGoogle Cloud Storage秘密鍵
```

これで次に示すようにGoogle Cloud Storageへの接続を確立できるようになります。

```
% python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto
>>> gs = boto.connect_gs()
```

AWSで利用可能なリージョンの検出

botoでEC2サービスへの接続を確立する際、デフォルトではUS-EAST-1リージョンに接続されます。当初はこれが利用できる唯一のリージョンでしたが、その後、AWSはその動作の規模をかなり拡張してきました。本書の執筆時点では、以下のリージョンでEC2を利用できるようになっており、それぞれ独自のエンドポイントが設定されています†。

- us-east-1［米国東部（バージニア北部）］
- us-west-1［米国西部（カリフォルニア北部）］
- us-west-2［米国西部（オレゴン）］
- eu-west-1［欧州西部（アイルランド）］
- ap-southeast-1［アジアパシフィック（シンガポール）］
- ap-northeast-1［アジアパシフィック（東京）］
- sa-east-1［南米（サンパウロ）］

botoではこれらのリージョンを検出して接続する方法がいくつも用意されています。例えば、次のコマ

† 原書では5つのリージョンが紹介されていますが、2012年7月現在は7つのリージョンが利用できます。

ンドを実行すると、指定したサービス（この場合はEC2）のすべてのRegionInfoオブジェクトのリストが返されます。それらのRegionInfoオブジェクトは、いずれもそのリージョンへのConnectionオブジェクトを返すconnectメソッドを持ちます。

```
$ python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto.ec2
>>> boto.ec2.regions()
[RegionInfo:eu-west-1, RegionInfo:us-east-1, RegionInfo:ap-northeast-1,
RegionInfo:us-west-1, RegionInfo:ap-southeast-1,RegionInfo:us-west-2,
RegionInfo:sa-east-1]
>>> eu_conn = _[0].connect()
```

接続するリージョンの名前がわかっている場合には、次の**方法**でも接続できます。

```
$ python
Python 2.7.1 (r271:86882M, Nov 30 2010, 10:35:34)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto.ec2
>>> eu_conn = boto.ec2.connect_to_region('eu-west-1')
>>>
```

最後に、EC2接続を確立するときにbotoで使用されるデフォルトのリージョンを変更するには、次に示すような記述を自分のbotoの構成ファイルに追加します。

```
[Boto]
ec2_region_name = eu-west-1
ec2-region-endpoint = ec2.eu-west-1.amazonaws.com
```

この記述を自分のbotoの構成ファイルに追加しておけば、それ以降はパラメータを指定せずにboto.connect_ec2を呼び出したときに、自動的にeu-west-1リージョンに接続できます†。

botoによるデバッグ出力の有効化

ときには何かが期待したとおりに動作しないことがあります。HTTPベースのAPIを介してリモートサービスと対話型処理を行っている場合、利用できる最良のデバッギングツールは、リモートサービスに送られた実際のHTTPリクエストと、リモートサービスから返された実際のHTTP応答を記録した詳細なログです。

† ec2-region-endpointにリージョンのエンドポイントを記述しておく必要もあります。

botoではPythonのロギングモジュールを使用して非常に詳細なログを記録する方法が用意されています。Pythonのロギングモジュールに組み込まれている多彩な機能の詳細については、http://docs.python.org/library/logging.html に記載されています。次に紹介するサンプルは、botoを使用するときに完全なデバッグ出力を対話型コンソールに表示する簡単な方法を示しています。

```
% python
>>> import boto
>>> boto.set_stream_logger('paws')
>>> ec2 = boto.connect_ec2(debug=2)
>>> s3 = boto.connect_s3(debug=2)
>>>
```

set_stream_loggerに渡す文字列は、何でも好きなものを指定できます。この文字列は、ログエントリのヘッダの位置に表示されます。この時点以降にec2または s3の接続を使用して実行されたすべての動作は、その完全なデバッグログが対話型のPythonシェルに出力されます。

botoにおけるソケットのタイムアウトの制御

クラウドサービスと通信するためbotoによって使用されるAPIは、すべてHTTPベースのものです。つまり、私たちは目に見えないところでソケットを介して分散サービスと通信していることになります。ときにはこれらのサービスが応答しなくなる可能性がありますし、自分のアプリケーションとサービスの間の通信層が信頼性を失う可能性もあります。このような事態に合理的な方法で対処する手立てとしては、タイムアウトが重要です。タイムアウトを使用すると、アプリケーションによってネットワークまたはサービスの問題を検出し、その処理を試みることが可能になります。あるいは、少なくとも**その事態について**ユーザに知らせることができます。

ソケットレベルのタイムアウトをbotoで明示的に管理するには、自分の botoの構成ファイルに次のようなセクションを追加します。タイムアウトは秒単位で指定します。

```
[Boto]
http_socket_timeout = 5
```