Eucalyptus 4.2.2 IAM Guide

Contents

IAM Overview

Welcome to the IAM Guide. IAM is an acronymn for Identity Access Management. IAM is used for account management where cloud administrator establish accounts, users and their identities.

You can do many things with IAM, including:

- Create and manage IAM users and groups
- Set quotas
- Establish access policies
- Create and manage roles

This document is intended to be a reference. You do not need to read it in order, unless you are following the directions for a particular task.

For IAM operations associated with the Management Console, see the *Eucalyptus Management Console Guide*.

Document version: Build 3221 (2016-07-14 22:01:24)

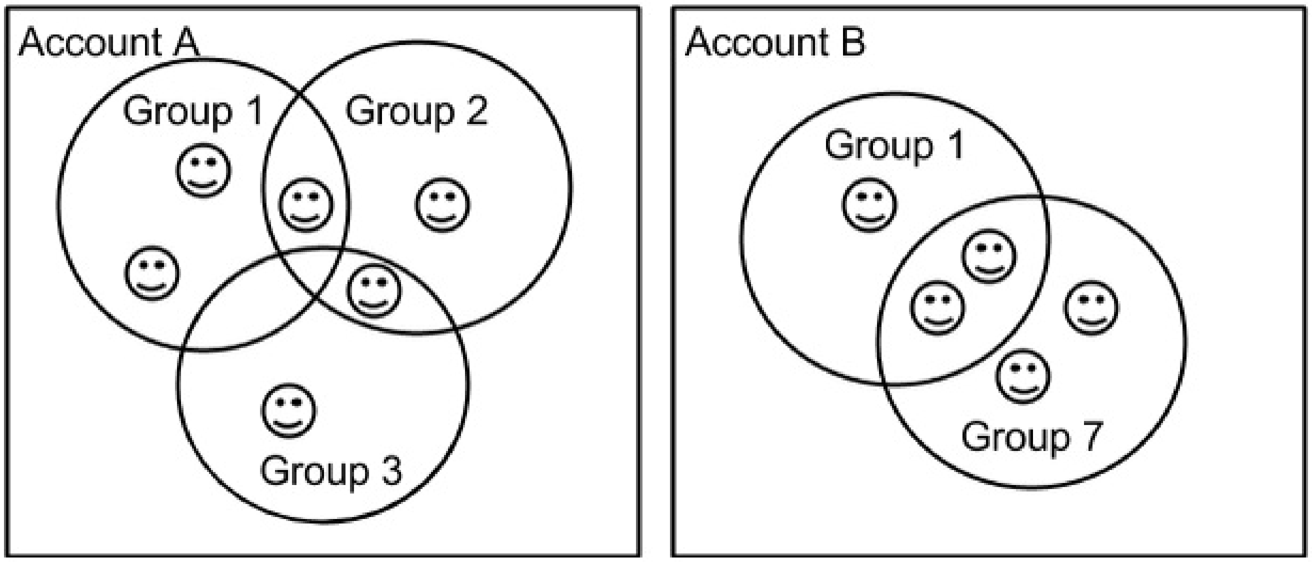
Work with IAM

The Eucalyptus IAM design provides more layers in the hierarchical organization of user identities, and more refined control over resource access. This complies with the Amazon AWS IAM service. There are a few Eucalyptus-specific extensions to meet the needs of enterprise customers.

The IAM service is a global service, meaning a user can interact with any region using the same credentials, subjected to the same policies, and having uniformly accessible and structured principals (accounts, users, groups, roles, etc.)

Manage Identities Overview

Like IAM, the user identities in Eucalyptus are organized into Accounts. An account is the unit of resource usage accounting, and also a separate name space for many resources, for example, security groups, key pairs, users, and so on. Unique ID (UUID) or a unique name identifies an account. The account name is equivalent to IAM's account alias. In Eucalyptus, the account name is used to manipulate accounts in most cases. However, to be compatible with AWS, the EC2 commands often use account ID to display resource ownership. There are command line tools to discover the correspondence of account ID with the account name. For example, `euare-accountlist` lists all the accounts with both their IDs and names.



An account can have multiple users, but a user can only be in one account. Within an account, users can be associated with Groups. A Group is used to attach access permissions to multiple users. A user can be associated with multiple groups. Because an account is a separate name space, user names and group names have to be unique only within an account. Therefore, user X in account A and user X in account B are two different identities. Both users and groups are identified by their names, which are unique within an account (they also have UUIDs, but rarely used).

Special Identities

Eucalyptus uses some special identities for the convenience of administration of the system.

- The ***eucalyptus*** account. This is similar to the root group in a Linux system. Every user in the *eucalyptus* account has full access to the resources in the system. These users are often referred to as system admin or cloud admin. This account is automatically created when the system starts for the first time. You cannot remove *eucalyptus* account from the system.
- The ***admin*** user of an account. Each account, including the *eucalyptus* account, has a user named *admin* by default, also created automatically by the system when an account is created. The *admin* of an account has full access to the resources owned by the account. You cannot remove the *admin* user from an account. The *admin* can delegate resource access to other users in the account by using policies.

Authentication and Access Control Best Practices

This topic describes best practices for Identity and Access Management and the `eucalyptus` account.

Identity and Access Management

Eucalyptus manages access control through an authentication, authorization, and accounting system. This system manages user identities, enforces access controls over resources, and provides reporting on resource usage as a basis for auditing and managing cloud activities. The user identity organizational model and the scheme of authorizations used to access resources are based on and compatible with the AWS Identity and Access Management (IAM) system, with some Eucalyptus extensions provided that support ease-of-use in a private cloud environment.

For a general introduction to IAM in Eucalyptus, see *Access Concepts* in the IAM Guide. For information about using IAM quotas to enforce limits on resource usage by users and accounts in Eucalyptus, see the *Quotas* section in the IAM Guide.

The *Amazon Web Services IAM Best Practices* are also generally applicable to Eucalyptus.

Credential Management

Protection and careful management of user credentials (passwords, access keys, X.509 certificates, and key pairs) is critical to cloud security. When dealing with credentials, we recommend:

- Limit the number of active credentials and do not create more credentials than needed.
- Only create users and credentials for the interfaces that you will actually use. For example, if a user is only going to use the Management Console, do not create credentials access keys for that user.
- Using `euca_conf --get-credentials` creates access keys and X.509 certificates; avoid unnecessary use of the command and use `euare-useraddkey` and `euare-usercreatecert` or `euare-useraddcert` instead to get a specific set of credentials if needed.
- Regularly check for active credentials using `euare-` commands and remove unused credentials. Ideally, only one pair of active credentials should be available at any time.
- Rotate credentials regularly and delete old credentials as soon as possible. Credentials can be created and deleted using `euare-` commands, such as `euare-useraddkey` and `euare-userdelkey`.
- When rotating credentials, there is an option to deactivate, instead of removing, existing access/secret keys and X.509 certificates. Requests made using deactivated credentials will not be accepted, but the credentials remain in the Eucalyptus database and can be restored if needed. You can deactivate credentials using `euare-usermodkey` and `euare-usermodcert`.

Privileged Roles

The `eucalyptus` account is a super-privileged account in Eucalyptus. It has access to all cloud resources, cloud setup, and management. The users within this account do not obey IAM policies and compromised credentials can result in a complete cloud compromisation that is not easy to contain. We recommend limiting the use of this account and associated users' credentials as much as possible.

For all unprivileged operations, use regular accounts. If you require super-privileged access (for example, management of resources across accounts and cloud setup administration), we recommend that you use one of the predefined privileged roles.

The Account, Infrastructure, and Resource Administrator *roles* provide a more secure way to gain super privileges in the cloud. Credentials returned by an assume-role operation are short-lived (unlike regular user credentials). Privileges available to each role are limited in scope and can be revoked easily by modifying the trust or access policy for the role.

Manage Users and Groups

You can also perform user authentication by integrating Eucalyptus with an existing LDAP or Active Directory. This information cannot be changed from Eucalyptus side when LDAP/AD integration is turned on. However, other Eucalyptus-specific information about user, group and account is still stored within the local database of Eucalyptus, including certificates, secret keys and attached policies.

For more information about synchronizing an existing LDAP or Active Directory with Eucalyptus, see *LDAP/AD Integration*.

Access Overview

The Eucalyptus design of user identity and access management provides layers in the organization of user identities. This gives you refined control over resource access. Though compatible with the AWS IAM, there are also a few Eucalyptus-specific extensions that meet the needs of enterprise customers.

Access Concepts

This section describes what Eucalyptus access is and what you need to know about it so that you can configure access to your cloud.

User Identities

In Eucalyptus, user identities are organized into accounts. An account is the unit of resource usage accounting, and also a separate namespace for many resources (security groups, key pairs, users, etc.).

Accounts are identified by either a unique ID (UUID) or a unique name. The account name is like IAM's account alias. It is used to manipulate accounts. However, for AWS compatibility, the EC2 commands often use account ID to display resource ownership.

There are command line tools to discover the correspondence of account ID and account name. For example, euare-accountlist lists all the accounts with both their IDs and names.

An account can have multiple users, but a user can only be in one account. Within an account, users can be associated with Groups. Group is used to attach access permissions to multiple users. A user can be associated with multiple groups. Because an account is a separate name space, user names and group names have to be unique only within an account. Therefore, user X in account A and user X in account B are two different identities.

Both users and groups are identified by their names, which are unique within an account (they also have UUIDs, but are rarely used).

Special Identities

Eucalyptus has two special identities for the convenience of administration and use of the system.

- The **eucalyptus** account: Each user in the eucalyptus account has unrestricted access to all of the cloud's resources, similar to the superuser on a typical Linux system. These users are often referred to as system administrators or cloud administrators. This account is automatically created when the system starts for the first time. You cannot remove the eucalyptus account from the system.
- The **admin** user of an account: Each account, including the eucalyptus account, has a user named admin. This user is created automatically by the system when an account is created. The admin of an account has full access to the resources owned by the account. You can not remove the admin user from an account. The admin can delegate resource access to other users in the account by using policies.

Credentials

This topic describes the different types of credentials used by Eucalyptus.

Each user has a unique set of credentials. These credentials are used to authenticate access to resources. There are three types of credentials:

- An **X.509 certificate** is used to authenticate requests to the SOAP API service.
- A **secret access key** is used to authenticate requests to the REST API service.

You can manage credentials using the command line tools (the `euare-` commands). For more information about the command line tools, see the *Euca2ools Reference Guide*.

In IAM, each account has its own credentials. In Eucalyptus, the equivalent of account credentials are the credentials of admin user of the account.

You can download the full set of credentials for a user or an account, including X509 certificate and secret access key, by:

```
/usr/sbin/euca_conf --get-credentials
```

or:

```
euca-get-credentials
```

Eucalyptus returns the following:

- An arbitrary existing active secret access key
- A newly generated X509 certificate

Account Creation

This topic describes the process for creating an account.

You can create accounts using the command line tool. You must be a cloud administrator to use this command. Accounts created are available for immediate access.

To create an account, run the following command:

```
euare-accountcreate -a account_name
```

Note: Eucalyptus has discontinued account registration status retrieval, however, for compatibility with older versions of Eucalyptus, the ability to view and manipulate registration status is limited to the system administrator through Euca2ools. For more information about the command line tools, see the *Euca2ools Reference Guide*.

Special User Attributes

Eucalyptus extends the IAM model by providing the following extra attributes for a user.

- **Registration status:** This is only meaningful for the account administrator (that is, the account level).
- **Enabled status:** . Use this attribute to temporarily disable a user.
- **Password expiration date**
- **Custom information:** Add any name-value pair to a user's custom information attribute. This is useful for attaching pure text information, like an address, phone number, or department. This is especially helpful with external LDAP or Active Directory services.

You can retrieve and modify the registration status, enabled status, and password expiration date using the `euare-usergetattributes` and `euare-usermod` commands. You can retrieve and modify custom information using `euare-usergetinfo` and `euare-userupdateinfo` commands. For more information, see the *Euca2ools Reference Guide* for details about these commands.

Roles

A *role* A role is a mechanism that enables the delegation of access to users or applications.

A role is associated with an account, and has a set of permissions associated with it that are defined in the form of an IAM *policy*. A policy specifies a set of actions and resources within that account that the role is allowed to access.

Note: For more information on policies, see *policies*Policy Overview.

By assuming a role, a user or an applications gets a set of permissions associated with that role. When a role is assumed, the Eucalyptus STS service returns a set of temporary security credentials that can then be used to make programmatic requests to resources in your account. This eliminates the need to share or hardcode security credentials with applications that need access to resources in your cloud.

Eucalyptus roles are managed through the Eucalyptus Euare service, which is compatible with Amazon's Identity and Access Management service. For more information on IAM and roles, please see the *Amazon IAM User Guide*.

Usage Scenarios for Roles

There are several scenarios in which roles can be useful, including:

Applications

Applications running on instances in your Eucalyptus cloud will often need access to other resources in your cloud. Instead of creating AWS credentials for each application, or distributing your own credentials,, you can use roles to enable you to delegate permission to the application to make API requests. For more information, see *Launch an Instance with a Role*.

Account Delegation

You can use roles to allow one account to access resources owned by another account. IAM Roles under the 'eucalyptus' account can be assumed by users under 'non-eucalyptus' account(s). For example, if you had an 'infrastructure auditing' account, and an audit was needed to be performed on all the cloud resources used on the cloud, a user could assume the 'Resource Administrator' role and audit all the resources used by all the accounts on the cloud. For more information on IAM account delegation, see *Using Roles to Delegate Permissions and Federate Identities*. Also, go to the walkthrough provided in the*AWS Identity and Access Management* section of the AWS documentation.

Pre-Defined Roles

Eucalyptus offers a number of pre-defined privileged roles. These roles are associated with the `eucalyptus` account, and have privileges to manage resources across the cloud and non-privileged accounts. Only the eucalyptus account can manage or modify these roles.

To see the pre-defined roles, use `euare-rolelistbypath` with the credentials of a user that is part of the `eucalyptus` account. For example:

```
# euare-rolelistbypath
arn:aws:iam::944786667073:role/eucalyptus/AccountAdministrator
arn:aws:iam::944786667073:role/eucalyptus/InfrastructureAdministrator
arn:aws:iam::944786667073:role/eucalyptus/ResourceAdministrator
```

Account Administrator

The Account Administrator (AA) can manage Eucalyptus accounts. To view the policy associated with the Account Administrator role, use `euare-rolelistpolicies` with the credentials of a user that is part of the `eucalyptus` account. For example:

```
# euare-rolelistpolicies --role-name AccountAdministrator --verbose
AccountAdministrator
{
  "Statement":[ {
    "Effect": "Allow",
    "Action": [
      "iam:*"
    ],
    "NotResource": "arn:aws:iam::eucalyptus:*",
    "Condition": {
      "Bool": { "iam:SystemAccount": "false" }
    }
  } ]
}
IsTruncated: false
```

Resource Administrator

The Resource Administrator (RA) can manage AWS-defined resources (such as S3 objects, instances, users, etc) across accounts. To view the policy associated with the Resource Administrator role, use `euare-rolelistpolicies` with the credentials of a user that is part of the `eucalyptus` account. For example:

```
# euare-rolelistpolicies --role-name ResourceAdministrator --verbose
ResourceAdministrator
{
  "Statement":[ {
    "Effect": "Allow",
    "Action": [
      "autoscaling:*",
      "cloudwatch:*",
      "ec2:DescribeInstanceAttribute",
      "ec2:DescribeInstances",
      "ec2:DescribeInstanceStatus",
      "ec2:DescribeInstanceTypes",
      "ec2:GetConsoleOutput",
      "ec2:GetPasswordData",
      "ec2:ImportInstance",
      "ec2:ModifyInstanceAttribute",
      "ec2:MonitorInstances",
      "ec2:RebootInstances",
      "ec2:ReportInstanceStatus",
      "ec2:ResetInstanceAttribute",
      "ec2:RunInstances",
      "ec2:StartInstances",
      "ec2:StopInstances",
      "ec2:TerminateInstances",
      "ec2:UnmonitorInstances",
      "ec2:*AccountAttributes*",
      "ec2:*Address*",
      "ec2:*AvailabilityZones*",
      "ec2:*Bundle*",
      "ec2:*ConversionTask*",
      "ec2:*CustomerGateway*",
      "ec2:*DhcpOptions*",
      "ec2:*ExportTask*",
      "ec2:*Image*",
      "ec2:*InternetGateway*",
      "ec2:*KeyPair*",
      "ec2:*NetworkAcl*",
      "ec2:*NetworkInterface*",
      "ec2:*PlacementGroup*",
      "ec2:*ProductInstance*",
      "ec2:*Region*",
      "ec2:*ReservedInstance*",
      "ec2:*Route*",
      "ec2:*SecurityGroup*",
      "ec2:*Snapshot*",
      "ec2:*SpotDatafeedSubscription*",
      "ec2:*SpotInstance*",
      "ec2:*SpotPrice*",
      "ec2:*Subnet*",
      "ec2:*Tag*",
      "ec2:*Volume*",
      "ec2:*Vpc*",
      "ec2:*Vpn*",
      "ec2:*VpnGateway*",
      "elasticloadbalancing:*",
      "s3:*"
    ],
    "Resource": "*"
  }, {
    "Effect": "Allow",
    "Action": [
      "iam:Get*",
      "iam:List*"
    ],
    "NotResource": "arn:aws:iam::eucalyptus:*"
  } ]
}
IsTruncated: false
```

Infrastructure Administrator

The Infrastructre Administrator (IA) can perform operations related to cloud setup and management. Typical responibilities include:

- Installation and configuration (prepare environment, install Eucalyptus, configure Eucalyptus)
- Monitoring and maintenance (infrastructure supporting the cloud, cloud management layer, upgrades, security patches, diagnostics and troubleshooting)
- Backup and restoration

To view the policy associated with the Infrastructure Administrator role, use `euare-rolelistpolicies` with the credentials of a user that is part of the `eucalyptus` account. For example:

```
# euare-rolelistpolicies --role-name InfrastructureAdministrator --verbose
InfrastructureAdministrator
{
  "Statement":[ {
    "Effect": "Allow",
    "Action": [
      "euprop:*",
      "euserv:*",
      "euconfig:*",
      "ec2:MigrateInstances"
    ],
    "Resource": "*"
  } ]
}
IsTruncated: false
```

Policy Overview

Eucalyptus uses the policy language to specify user level permissions as AWS IAM. Policies are written in JSON. Each policy file can contain multiple statements, each specifying a permission.

A permission statement specifies whether to allow or deny a list of actions to be performed on a list of resources, under specific conditions.

A permission statement has the following components:

- **Effect:** Begins the decision that applies to all following components. Either: `“Allow”` or `“Deny”`
- **Action or NotAction:** Indicates service-specific and case-sensitive commands. For example: `“ec2:RunInstances”`
- **Resource or NotResource:** Indicates selected resources, each specified as an Amazon resource name (ARN). For example: `“arn:aws:s3:::acme_bucket/blob”`
- **Condition:** Indicates additional constraints of the permission. For example: `"DateGreaterThan"`

The following policy example contains a statement that gives a user with full permission. This is the same access as the account administrator:

```
{
  "Version":"2011-04-01",
  "Statement":[{
    "Sid":"1",
    "Effect":"Allow",
    "Action":"*",
    "Resource":"*"
  }]
}
```

For more information about policy language, go to the *IAM User Guide*.

Policy Notes

You can combine IAM policies with account level permissions. For example, the admin of account A can give users in account B permission to launch one of account A’s images by changing the image attributes. Then the admin of account B can use IAM policy to designate the users who can actually use the shared images.

You can attach IAM policies to both users and groups. When attached to groups, a policy is equivalent to attaching the same policy to the users within that group. Therefore, a user might have multiple policies attached, both policies attached to the user, and policies attached to the group that the user belongs to.

Do not attach IAM policies (except quota policies, a Eucalyptus extension) to account admins. At this point, doing so won’t result in a failure but may have unexpected consequences.

Policy Extensions

Eucalyptus extends the IAM policy in order to meet the needs of enterprise customers.

EC2 Resource

In IAM, you cannot specify EC2 resources in a policy statement except a wildcard,`“*”`. So, you can't restrict a permission to specific EC2 entities. For example, you can't restrict a user to run instances on a specific image or VM type. To solve that, Eucalyptus created the EC2 resource for the policy language. The following example shows the ARN of an EC2 resource.

```
arn:aws:ec2::<account_id>:<resource_type>/<resource_id>
```

Where account id is optional.

Eucalyptus supports the following resource types for EC2:

- image
- securitygroup
- address (either an address or address range: 192.168.7.1-192.168.7.255)
- availabilityzone
- instance
- keypair
- volume
- snapshot
- vmtype

The following example specifies permission to launch instances with only an m1.small VM type:

```
{
  "Version":"2011-04-01",
  "Statement":[{
    "Sid":"2",
    "Effect":"Allow",
    "Action":"ec2:RunInstances",

    "Resource": [
    "arn:aws:ec2:::vmtype/m1.small",
    "arn:aws:ec2:::image/*",
    "arn:aws:ec2:::securitygroup/*",
    "arn:aws:ec2:::keypair/*",
    "arn:aws:ec2:::availabilityzone/*",
    "arn:aws:ec2:::instance/*"
]
  }]
}
```

Policy Key

Eucalyptus implements the following AWS policy keys:

- aws:CurrentTime
- aws:SourceIp

Eucalyptus extends the policy keys by adding the following to the lifetime of an instance:

- ec2:KeepAlive: specifies the length of time (in minutes) that an instance can run
- ec2:ExpirationTime: specifies the expiration time (in minutes) for an instance

The following example restricts an instance running time to 24 hours:

```
{
  "Version":"2011-04-01",
  "Statement":[{
    "Sid":"3",
    "Effect":"Allow",
    "Action":"ec2:RunInstances",
    "Resource":"*",
    "Condition":{
      “NumericEquals”:{
        “ec2:KeepAlive”:”1440”
      }
    }
  }]
}
```

If there are multiple `ec2:KeepAlive` or `ec2:ExpirationTime` keys that match a request, Eucalyptus chooses the longer lifetime for the instance to run.

Default Permissions

Different identities have different default access permissions. When no policy is associated with them, these identities have the permission listed in the following table.

Identity	Permission
System admin	Access to all resources in the system
Account admin	Access to all account resources, including those shared resources from other accounts like public images and shared snapshots
Regular user	No access to any resource

Quotas

Eucalyptus adds quota enforcement to resource usage. To avoid introducing another configuration language into Eucalyptus, and simplify the management, we extend the IAM policy language to support quotas.

The only addition added to the language is the new `limit` effect. If a policy statement’s `effect` is `limit`, it is a quota statement.

A quota statement also has action and resource fields. You can use these fields to match specific requests, for example, quota only being checked on matched requests. The actual quota type and value are specified using special quota keys, and listed in the `condition` part of the statement. Only condition type `NumericLessThanEquals` can be used with quota keys.

Important: An account can only have a quota policy. Accounts can only accept IAM policies where Effect is "Deny" or "Limit". If you attach an IAM policy to an account where the Effect is "Allow", you will get unexpected results.

The following quota policy statement limits the attached user to only launch a maximum of 16 instances in an account.

```
{
 "Version":"2011-04-01",
 "Statement":[{
   "Sid":"4",
   "Effect":"Limit",
   "Action":"ec2:RunInstances",
   "Resource":"*",
   "Condition":{
     “NumericLessThanEquals”:{
       “ec2:quota-vminstancenumber”:”16”
     }
   }
 }]
}
```

You can attach quotas to both users and accounts, although some of the quotas only apply to accounts. Quota attached to groups will take no effect.

When a quota policy is attached to an account, it actually is attached to the account administrator user. Since only system administrator can specify account quotas, the account administrator can only inspect quotas but can't change the quotas attached to herself.

The following is all the quota keys implemented in Eucalyptus:

Quota Key	Description	Applies to
`autoscaling:quota-autoscalinggroupnumber`	The number of Autoscaling Groups	account and user
`autoscaling:quota-launchconfigurationnumber`	Number of Autoscaling Group Launch Configurations	account and user
`autoscaling:quota-scalingpolicynumber`	Number of Autoscaling Group Scaling Policies	account and user
`cloudformation:quota-stacknumber`	Number of Cloudformation stacks allowed to create	account
`ec2:quota-addressnumber`	Number of elastic IPs	account and user
`ec2:quota-cputotalsize`	Number of Total CPUs Used by EC2 Instances	account and user
`ec2:quota-disktotalsize`	Number of Total Disk Space (in GB) of EC2 Instances	account and user
`ec2:quota-imagenumber`	Number of EC2 images	account and user
`ec2:quota-internetgatewaynumber`	Number of EC2 VPC Internet Gateways	account and user
`ec2:quota-memorytotalsize`	Number of Total Amount of Memory Used by EC2 Instances	account and user
`ec2:quota-securitygroupnumber`	Number of EC2 security groups	account and user
`ec2:quota-snapshotnumber`	Number of EC2 snapshots	account and user
`ec2:quota-vminstancenumber`	Number of EC2 instances	account and user

Quota Key	Description	Applies to
`ec2:quota-vminstanceactivenumber`	Number of EC2 Instances Using Node Resources (pending, running, shutting-down, etc.)	account and user
`ec2:quota-volumenumber`	Number of EC2 volumes	account and user
`ec2:quota-volumetotalsize`	Number of total volume size, in GB	account and user
`ec2:quota-vpcnumber`	Number of EC2 VPCs	account and user
`elasticloadbalancing:quota-loadbalancernumber`	Number of Elastic Load Balancers	account
`iam:quota-groupnumber`	Number of IAM groups	account
`iam:quota-instanceprofilenumber`	Number of IAM Instance Profiles	account and user
`iam:quota-rolenumber`	Number of IAM Roles	account and user
`iam:quota-servercertificatenumber`	Number of IAM Server Certificates	account and user
`iam:quota-usernumber`	Number of IAM users	account
`s3:quota-bucketnumber`	Number of S3 buckets	account and user
`s3:quota-bucketobjectnumber`	Number of objects in each bucket	account and user
`s3:quota-bucketsize`	Size of bucket, in MB	account and user
`s3:quota-buckettotalsize`	total size of all buckets, in MB	account and user

Default Quota

Contrary to IAM policies, by default, there is no quota limits (except the hard system limit) on any resource allocations for a user or an account. Also, system administrators are not constrained by any quota. Account administrators are only be constrained by account quota.

Algorithms

This topic describes the algorithms used by Eucalyptus to determine access.

Policy Evaluation Algorithm

You can associated multiple policies and permission statements with a user. The way these are combined together to control the access to resources in an account is defined by the policy evaluation algorithm. Eucalyptus implements the *same policy evaluation algorithm as AWS IAM*:

1. If the request user is account admin, access is allowed.
2. Otherwise, collect all the policy statements associated with the request user (attached to the user and all the groups the user belongs to), which match the incoming request (i.e. based on the API being invoked and the resources it is going to access).
    a. If there is no matched policy statement, access is denied (default implicit deny).
    b. Otherwise, evaluate each policy statement that matches.
        - If there is a statement that explicitly denies the access, the request is denied.
        - If there is no explicit deny, which means there is at least one explicit allow, access is allowed.

Access Evaluation Algorithm

Now we give the overall access evaluation combining both account level permissions and IAM permissions, which decides whether a request is accepted by Eucalyptus:

1. If the request user is sys admin, access is allowed.
2. Otherwise, check account level permissions, e.g. image launch permission, to see if the request user's account has access to the specific resources.
    a. If not, the access is denied.
    b. Otherwise, invoke the policy evaluation algorithm to check if the request user has access to the resources based on IAM policies.

Quota Evaluation Algorithm

Like the normal IAM policies, a user may be associated with multiple quota policies (and multiple quota statements). How all the quota policies are combined to take effect is defined by the quota evaluation algorithm:

1. If the request user is sys admin, there is no limit on resource usage.
2. Otherwise, collect all the quotas associated with the request user, including those attached to the request user's account and those attached to the request user himself/herself (for account admin, we only need collect account quotas).
3. Evaluate each quota one by one. Reject the request as long as there is one quota being exceeded by the request. Otherwise, accept the request.

Note: The hard limits on some resources override quota limits. For example, `walrusbackend.storagemaxbucketsizeinmb` (system property) overrides the `s3:quota-bucketsize` (quota key).

Sample Policies

A few example use cases and associated policies.

Here are some example use cases and associated polices. You can edit these polices for your use, or use them as examples of JSON syntax and form.

Tip: For more information about JSON syntax used with AWS resources, go to *Using AWS Identity and Access Management*.

Examples: Allowing Specific Actions

The following policy allows a user to only run instances and describe things.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":["ec2:*Describe*","ec2:*Run*"],
      "Resource":"*",
      }
   ]
   }
```

The following policy allows a user to only list things:

```
{
  "Statement": [
    {
      "Sid": "Stmt1313686153864",
      "Action": [
        "iam:List*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
  }
```

The following policy grants a generic basic user permission for running instances and describing things.

```
{
  "Statement": [
    {
      "Sid": "Stmt1313605116084",
      "Action": [
        "ec2:AllocateAddress",
        "ec2:AssociateAddress",
        "ec2:AttachVolume",
        "ec2:Authorize*",
        "ec2:CreateKeyPair",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSnapshot",
        "ec2:CreateVolume",
        "ec2:DeleteKeyPair",
        "ec2:DeleteSecurityGroup",
        "ec2:DeleteSnapshot",
        "ec2:DeleteVolume",
        "ec2:Describe*",
        "ec2:DetachVolume",
        "ec2:DisassociateAddress",
        "ec2:GetConsoleOutput",
        "ec2:RunInstances",
        "ec2:TerminateInstances"
        "ec2:ReleaseAddress"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Examples: Denying Specific Actions

The following policy allows a user to do anything but delete.

```
{
  "Statement": [
    {
      "Action": [
        "ec2:Delete*"
      ],
      "Effect": "Deny",
      "Resource": "*"
    }
  ]
  }
```

The following policy denies a user from creating other users.

```
{
  "Statement": [
    {
      "Sid": "Stmt1313686153864",
      "Action": [
        "iam:CreateUser"
      ],
      "Effect": "Deny",
      "Resource": "*"
    }
  ]
}
```

Examples: Specifying Time Limits

The following policy allows a user to run instances within a specific time.

```
{
  "Statement": [
    {
      "Sid": "Stmt1313453084396",
      "Action": [
        "ec2:RunInstances"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "DateLessThanEquals": {
          "aws:CurrentTime": "2011-08-16T00:00:00Z"
        }
      }
    }
  ]
}
```

The following policy blocks users from running instances at a specific time.

```
{
  "Statement": [
    {
      "Sid": "Stmt1313453084396",
      "Action": [
        "ec2:RunInstances"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "DateLessThanEquals": {
          "aws:CurrentTime": "2011-08-16T00:00:00Z"
        }
      }
    }
  ]
  }
```

The following policy keeps alive an instance for 1,000 hours (60,000 minutes).

```
{
  "Statement": [
    {
      "Action": ["ec2:RunInstances" ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": { "NumericEquals":{"ec2:KeepAlive":"60000"}}
    }
  ]
  }
```

The following policy sets an expiration date on running instances.

```
{
  "Statement": [
    {
      "Action": ["ec2:RunInstances" ],
      "Effect": "Allow",
      "Resource": "*",
     "Condition": { "DateEquals":{"ec2:ExpirationTime":"2011-08-16T00:00:00Z"}}

    }
  ]
}
```

Examples: Restricting Resources

The following policy allows users to only launch instances with a large image type.

```
{
  "Statement": [
    {
      "Action": [
        "ec2:RunInstances"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:ec2:::vmtype/m1.xlarge"
    }
  ]
  }
```

The following policy restricts users from launching instances with a specific image ID.

```
{
  "Statement": [
    {
      "Action": [
        "ec2:RunInstances"
      ],
      "Effect": "Deny",
      "Resource": "arn:aws:ec2:::image/emi-0FFF1874"
    }
  ]
  }
```

The following policy restricts users from allocating addresses to a specific elastic IP address.

```
{
  "Statement": [
    {
      "Sid": "Stmt1313626078249",
      "Action": "*",
      "Effect": "Deny",
      "Resource": "arn:aws:ec2:::address/192.168.10.140"
    }
  ]
  }
```

The following policy denies volume access.

```
{
  "Statement": [
    {
      "Action": [
        "ec2:*"
      ],
      "Effect": "Deny",
      "Resource": "arn:aws:ec2:::volume/*"
    }
  ]

}
```

Note: For policies attached to an account, quota limits can be specified. See the Quotas section for further details.

LDAP/AD Integration

You can use the Eucalyptus LDAP/Active Directory (AD) integration to synchronize existing LDAP/AD user and group information with Eucalyptus.

When you enable LDAP/AD synchronization, Eucalyptus imports specified user and group information from LDAP or AD and maps them into a predefined two-tier account/group/user structure

Note that Eucalyptus only imports the identities and some related information. Any Eucalyptus-specific attributes are still managed from Eucalyptus. These include:

- User credentials: secret access keys and X.509 certificates.
- Policies: IAM policies and quotas. Policies are associated with identities within Eucalyptus, and stored in internal database.

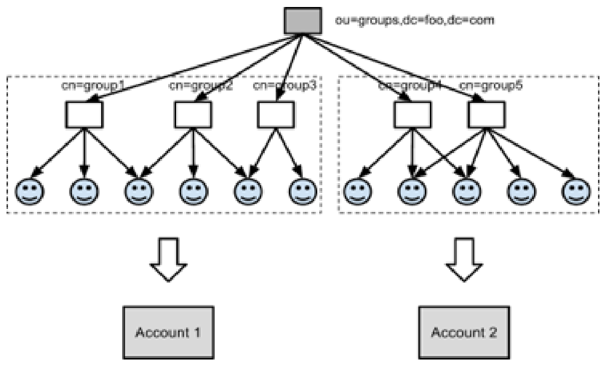
Also note that special identities, including system administrators and account administrators, are created in Eucalyptus and not imported from LDAP/AD. Only normal user identities are imported.

Important: If you integrate LDAP/AD, you do not need to create IAM user login profiles for your users.

Identity Mapping

Identities in LDAP/AD are organized differently from the identity structure in Eucalyptus. So a transformation is required to map LDAP/AD identities into Eucalyptus.

The following image shows a simple scheme of how the mapping works. In this scheme, the user groups in LDAP tree are partitioned into two sets. Each set is mapped into one separate account. Group 1, 2 and 3 are mapped to Account 1 and Group 4 and 5 are mapped to Account 2. As the result, all users in Group 1, 2 and 3 will be in Account 1, and all users in Group 4 and 5 will be in Account 2.



To summarize the mapping method:

1. Pick user groups from LDAP/AD and combine them into different accounts. There are two ways of doing this:

   - Use something called accounting groups. Account groups are essentially groups of groups. Accounting groups rely on a key understanding of object class types in LDAP. In short, accounting groups are mapped to STRUCTURAL object classes in LDAP. For more information about object class types, refer to the *LDAP Models RFC* under the "2.4. Object Classes". Each accounting group contains multiple user groups in LDAP/AD. Then each accounting group maps to an account in Eucalyptus.
   - Manually partition groups into accounts. Each group partition maps to an account.

2. Once the accounts are defined (by accounting groups or group partitions), all the LDAP/AD user groups will be mapped into Eucalyptus groups within specific accounts; and LDAP/AD users will be mapped into Eucalyptus users. Using the options to filter the groups and users to be imported into Eucalyptus allows granular control.
3. Groups are group object types in LDAP. The group object type in LDAP/AD needs to have the attribute type determining membership where the value is the Fully Distinguished Name (FDN) of the user(s). Some examples of group object types for LDAP/AD are as follows:

   - *groupOfNames*
   - *groupOfUniqueNames*

- *Group-Of-Names*
- *groupOfUniqueNames*
- *Group*

Note that each group can be mapped into multiple accounts. But understand that Eucalyptus accounts are separate name spaces. So for groups and users that are mapped into different accounts, their information (name, attributes, etc) will be duplicated in different accounts. And duplicated users will have separate credentials in different accounts. For example, Group 1 may map to both Account 1 and Account 2. Say user A belongs to Group 1. Then Account 1 will have user A and Account 2 will also have user A. User A in Account 1 and user A in Account 2 will have different credentials, policies, etc., but the same user information.

Note: Currently, there is not a way to map individual users into an account. The mapping unit is LDAP user group. What maps where groups and users end up regarding accounts DEPENDS upon the accounting-groups or groups-partition definitions.

LDAP/AD Integration Configuration

The LDAP/AD Integration Configuration (LIC) is a JSON format file. This file specifies everything Eucalyptus needs to know about how to synchronize with an LDAP or AD service.

You can find a LIC template at `/usr/share/eucalyptus/lic_template`. This template shows all the fields of the LIC, and provides detailed documentation and example values for each field.

To start a LIC file, use the LIC command line tool.

```
/usr/sbin/euca-lictool --password <password> --out example.lic
```

The above command invokes the LIC tool to create a template LIC and fill in the encrypted password for authenticating to LDAP/AD service (i.e. the password of the administrative user for accessing the LDAP/AD during synchronization). The LIC tool’s primary functions are to encrypt the LDAP/AD password and to generate the starting LIC template. The usage of the LIC tool shows different ways to invoke the command.

Once you have the LIC template, you can fill in the details by editing the “*.lic” file using your favorite editor as it is a simple text file. As we said above, the LIC file is in JSON format. Each top level entity specifies one aspect of the LDAP/AD synchronization. The following shows one possible example of a LIC file.

```
{
"ldap-service":{
  "server-url":"ldap://localhost:7733",
  "auth-method":"simple",
  "user-auth-method":"simple",
  "auth-principal":"cn=ldapadmin,dc=foo,dc=com",
  "auth-credentials": "{RSA/ECB/PKCS1Padding}EAXRnvwnKtCZOxSrD/F3ng/yHH3J4jMxNUS

  kJJf6oqNMsUihjUerZ20e5iyXImPgjK1ELAPnppEfJvhCs7woS7jtFsedunsp5DJCNhgmOb2CR/MnH

  11V3FNY7bBWoew5A8Wwy6x7YrPMS0j7dJkwM7yfp1Z6AbKOo2688I9uIvJUQwEKS4dOp7RVdA0izlJ

  BDPAxiFZ2qa40VjFI/1mggbiWDNlgxiVtZXAEK7x9SRHJytLS8nrNPpIvPuTg3djKiWPVOLZ6vpSgP

  cVEliP261qdUfnf3GDKi3jqbPpRRQ6n8yI6aHw0gAtq8/qPyqjkkDP8JsGBgmXMxiCNPogbWg==",

  "use-ssl":"false",
  "ignore-ssl-cert-validation":"false",
  "krb5-conf":"/path/to/krb5.conf",
},

"sync":{
  "enable":"true",
  "auto":"true",
  "interval":"900000",
  "clean-deletion":"false",
},
"accounting-groups":{
  "base-dn":"ou=groups,dc=foo,dc=com",
  "id-attribute":"cn",
  "member-attribute":"member",
  "selection":{
      "filter":"objectClass=accountingGroup",
      "select":["cn=accountingToSelect,ou=Groups,dc=foo,dc=com"],
      "not-select":["cn=accountingToIgnore,ou=Groups,dc=foo,dc=com"],
  }
},

"groups":{
  "base-dn":"ou=groups,dc=foo,dc=com",
  "id-attribute":"cn",
  "member-attribute":"member",
  "selection":{
      "filter":"objectClass=groupOfNames",
      "select":["cn=groupToSelect,ou=Groups,dc=foo,dc=com"],
      "not-select":["cn=groupToIgnore,ou=Groups,dc=foo,dc=com"],
  }
},

"users":{
  "base-dn":"ou=people,dc=foo,dc=com",
  "id-attribute":"uid",
  "user-info-attributes":{
      "fullName":"Full Name",
      "email":"Email"
  },
  "selection":{
      "filter":"objectClass=inetOrgPerson",
      "select":["uid=john,ou=People,dc=foo,dc=com",
"uid=jack,ou=People,dc=foo,dc=com"],
      "not-select":["uid=tom,ou=People,dc=foo,dc=com"],
  }
},
```

In the following sections explain each field of LIC in detail.

ldap-service

The `ldap-service` element contains everything related to the LDAP/AD service.

Element	Description
server-url	The LDAP/AD server URL, starting with ldap://.
auth-method	The LDAP/AD authentication method to perform synchronization.
auth-principal	The ID of the administrative user for synchronization.
auth-credentials	The credentials for LDAP/AD authentication, like a password. We recommend that you encrypt this using `/usr/sbin/euca-lictool`.
user-auth-method	The LDAP/AD authentication method for normal users to perform Management Console login. • *simple*: for clear text user/password authentication. • *DIGEST-MD5*: for SASL authentication using MD5 • *GSSAPI*: SASL authentication using Kerberos V5.

Element	Description
use-ssl	Specifies whether to use SSL for connecting to LDAP/AD service. If this option is enabled, make sure the SSL port for LDAP is defined as part of the server-url. The default port for LDAP+SSL is port 636.
ignore-ssl-cert-validation	Specifies whether to ignore self-signed SSL certs. This is useful when you only have self-signed SSL certs for your LDAP/AD services.
krb5-conf	The file path for krb5.conf, if you use GSSAPI authentication method.

sync

The `sync` element contains elements for controlling synchronization.

Element	Description
enable	Set to "true" to enable LDAP synchronization. When this is “false”, all other fields can be ignored. Default value: false
auto	Set to true to turn on automatic synchronization. Set to false to turn off synchronization.
interval	The length in milliseconds of the automatic synchronization interval.
clean-deletion	Parameter denoting whether to remove identity entities from Eucalyptus when they are deleted from LDAP. Set to true if you want Eucalyptus to remove any identities once their counterparts in LDAP are deleted. Set to false if you want these identities kept without being purged.

accounting-groups

This section uses a special group in LDAP/AD to designate accounts in the Eucalyptus “accounting group.” The accounting group takes normal LDAP/AD groups as members, i.e., they are groups of groups.

The accounting group’s name becomes the account name in Eucalyptus. The member groups become Eucalyptus groups in that account. And the users of all those groups become Eucalyptus users within that account and corresponding Eucalyptus groups.

Important: If you use `accounting-groups`, remove the `groups-partition` section. These two sections are mutually exclusive.

Element	Description
base-dn	The base DN of accounting groups in the LDAP/AD tree.
id-attribute	The ID attribute name of the accounting group entry in LDAP/AD tree.
member-attribute	The LDAP/AD attribute name for members of the accounting group.

Element	Description
selection	The accounting groups you want to map to. This contains the following elements: • *filter*: The LDAP/AD searching filter used for the LDAP/AD search to get the relevant LDAP/AD entities, e.g. the users to be synchronized. (Example: objectClass=groupOfNames). This element works the same as the filter option that is found in ldapsearch, therefore when doing more advanced searching using compound filters, use boolean operators - AND (&), OR (|), and/or NOT (!). (Example: `(&(ou=Sales)(objectClass=groupOfNames))` • *select*: Explicitly gives the full DN of entities to be synchronized, in case they can not be specified by the search filter. (Example: cn=groupToSelect,ou=Groups,dc=foo,dc=com) • *not-select:* Explicitly gives the full DN of entities NOT to be synchronized, in case this can not be specified by the search filter. (Example: cn=groupToIgnore,ou=Groups,dc=foo,dc=com)

groups-partition

Like accounting-groups, groups-partition specifies how to map LDAP/AD groups to Eucalyptus accounts. However, in this section you to manually specify which LDAP/AD groups you want to map to Eucalyptus accounts.

Important: If you use `groups-partition`, remove the `accounting-groups` section. These two sections are mutually exclusive.

The Eucalyptus accounts are created by partitioning LDAP/AD groups. Each partition composes an Eucalyptus account. So all the groups within the partition become Eucalyptus groups within that account. All the users of those groups will become Eucalyptus users within that account and the corresponding Eucalyptus groups.

This section requires that you specify one partition at a time, using a list of JSON key-value pairs. For each entry, the key is the account name to be mapped and the value is a list of names of LDAP/AD groups to be mapped into the account. For example:

```
"groups-partition": {
          "salesmarketing": ["sales", "marketing"],
          "devsupport": ["engineering", "support"],
}
```

Here salesmarketing and devsupport are names for the groups partition and are used as the corresponding Eucalyptus account names.

Tip: If you use groups-partition, remove the accounting-groups section. These two sections are mutually exclusive.

groups

The `groups` element specifies how to map LDAP/AD groups to Eucalyptus groups. It contains the elements listed in the following table. The meanings are similar to those in accounting-groups element.

Element	Description
base-dn	The base DN for searching groups.
id-attribute	The ID attribute name of the LDAP group.

Element	Description
member-attribute	The name of the attribute for group members. Usually, it is member in modern LDAP implementation, which lists full user DN.
selection	The specific LDAP/AD groups you want to map to. This contains the following elements: • *filter*: The LDAP/AD searching filter used for the LDAP/AD search to get the relevant LDAP/AD entities, e.g. the users to be synchronized. (Example: objectClass=groupOfNames). This element works the same as the filter option that is found in ldapsearch, therefore when doing more advanced searching using compound filters, use boolean operators - AND (&), OR (|), and/or NOT (!). (Example: `(&(ou=Sales)(objectClass=groupOfNames))` • *select*: The LDAP/AD searching filter used for the LDAP/AD search to get the relevant LDAP/AD entities, e.g. the users to be synchronized. (Example: objectClass=groupOfNames) • *not-select:* Explicitly gives the full DN of entities NOT to be synchronized, in case this can not be specified by the search filter. (Example: cn=groupToIgnore,ou=Groups,dc=foo,dc=com)

users

Explicitly gives the full DN of entities NOT to be synchronized, in case this can not be specified by the search filter.

Element	Description
base-dn	The base DN for searching users.
id-attribute	The attribute ID of the LDAP user.
selection	The specific LDAP/AD users you want to map to. This contains the following elements: • *filter*: The LDAP/AD searching filter used for the LDAP/AD search to get the relevant LDAP/AD entities, e.g. the users to be synchronized. (Example: objectClass=organizationalPerson). This element works the same as the filter option that is found in ldapsearch, therefore when doing more advanced searching using compound filters, use boolean operators - AND (&), OR (|), and/or NOT (!). (Example: `(&(ou=Sales)(objectClass=organizationalPerson)))` • *select*: Explicitly gives the full DN of entities to be synchronized, in case they can not be specified by the search filter. (Example: cn=userToSelect,ou=People,dc=foo,dc=com) • *not-select:* Explicitly gives the full DN of entities NOT to be synchronized, in case this can not be specified by the search filter. (Example: cn=userToIgnore,ou=People,dc=foo,dc=com)

Synchronization Process

This topic explains what happens to start the synchronization process and what the synchronization process does.

The synchronization always starts when the following happens:

- You manually upload a LDAP/AD Integration Configuration (LIC) file. Every new or updated LIC upload triggers a new synchronization.
- If the automatic synchronization is enabled, a synchronization is started when the timer goes off.

Note: Eucalyptus does not allow concurrent synchronization. If you trigger synchronization more than once within a short time period, Eucalyptus only allows the first one.

During a synchronization, everything specified by an LIC in the LDAP/AD tree will be downloaded into Eucalyptus' internal database. Each synchronization is a merging process of the information already in the database and the information from LDAP/AD. There are three cases for each entity: user, group or account:

- If an entity from LDAP/AD is not in Eucalyptus, a new one is created in the database.
- If an entity from LDAP/AD is already in Eucalyptus, the Eucalyptus version is updated. For example, if a user's info attributes are changed, those changes are downloaded and updated.
- If an entity in Eucalyptus is missing from LDAP/AD, it will be removed from the database if the clean-deletion option in LIC is set to true. Otherwise, it will be left in the database.

Important: If clean-deletion is set to true, the removed entities in Eucalyptus will be lost forever, along with all its permissions and credentials. The resources associated with the entity will be left untouched. It is system administrator's job to recycle these resources.

Synchronize LDAP/AD

To start an LDAP/AD synchronization:

1. Create an LDAP/AD Integration Configuration (LIC) file to specify all the details about the LDAP/AD synchronization.
2. Upload the LIC file to Eucalyptus using `euca-modify-property`.

Start a LIC File

To start a LIC file perform the steps listed in this topic.

The LIC is a file in JSON format, specifying everything Eucalyptus needs to know about how to synchronize with an LDAP or AD service. Eucalyptus provides a LIC template at `${EUCALYPTUS}/usr/share/eucalyptus/lic_template`. This template shows all the fields of the LIC, and provides detailed documentation and example values for each field.

To start a LIC file:

Enter the following command:

```
/usr/sbin/euca-lictool --password secret --out example.lic
```

The above command invokes the LIC tool to create a template LIC and fill in the encrypted password for authenticating to LDAP/AD service (i.e. the password of the administrative user for accessing the LDAP/AD during synchronization). The LIC tool's primary functions are to encrypt the LDAP/AD password and to generate the starting LIC template. The usage of the LIC tool shows different ways to invoke the command.

Once you have the LIC template, you can fill in the details by editing the `*.lic` file using a text editor. Each top level entity specifies one aspect of the LDAP/AD synchronization.

Upload a New LIC File

To upload a new LIC file perform the steps listed in this topic.

To upload a new LIC file:

Enter the following:

```
/usr/sbin/euca-modify-property -f
             authentication.ldap_integration_configuration=<lic_filename.lic>
```

This triggers a new synchronization using the uploaded LIC file.

Access Tasks

This section provides details about the tasks you perform using policies and identities. The tasks you can perform are divided up into tasks for users, tasks for groups, and tasks for policies.

The following use cases detail work flows for common processes:

- *Use Case: Create an Administrator*
- *Use Case: Create a User*

You can perform the following access-related tasks listed in the following sections:

- Accounts:
  - *Add an Account*
  - *Rename an Account*
  - *List Accounts*
  - *Delete an Account*
- Groups:
  - *Create a Group*
  - *Add a Group Policy*
  - *Modify a Group*
  - *Add a User to a Group*
  - *Remove a User from a Group*
  - *List Groups*
  - *List Policies for a Group*
  - *Delete a Group*
- Users:
  - *Add a User*
  - *Add a User to a Group*
  - *Create a Login Profile*
  - *Modify a User*
  - *List Users*
  - *Delete a User*
- Credentials:
  - *Generating User Credentials*
  - *Retrieving Existing User Credentials*
  - *Uploading a Certificate*
  - *Working with Administrator Credentials*
- Roles:
  - *Launch an Instance with a Role*
  - *Use a Role with an Instance Application*

Note: To distinguish an IAM operation, run commands with --help and look for *iam* described in the --url option, which indicates the IAM service that the command is talking to.

Use Case: Create an Administrator

This use case details tasks for creating an administrator. These tasks require that you have your account credentials for sending requests to Eucalyptus using the command line interface (CLI) tools.

To create an administrator account, perform the tasks that follows.

Create an Admin Group

Eucalyptus recommends using account credentials as little as possible. You can avoid using account credentials by creating a group of users with administrative privileges.

1. Create a group called `administrators`.

   ```
   euare-groupcreate -g administrators
   ```

2. Verify that the group was created.

   ```
   euare-grouplistbypath
   ```

   Eucalyptus returns a listing of the groups that have been created, as in the following example.

   ```
   arn:aws:iam::123456789012:group/administrators
   ```

Add a Policy to the Group

Add a policy to the administrators group that allows its members to perform all actions in Eucalyptus.

Enter the following command to create a policy called `admin-root` that grants all actions on all resources to all users in the administrators group:

```
euare-groupaddpolicy -p admin-root -g administrators -e Allow -a "*" -r "*"
-o
```

Create an Administrative User

Create a user for day-to-day administrative work and add that user to the administrators group.

1. Enter the following command to create an administrative user named `alice`:

   ```
   euare-usercreate -u alice
   ```

2. Add the new administrative user to the administrators group.

   ```
   euare-groupadduser -g administrators -u alice
   ```

Generate Administrative Credentials

To start running commands as the new administrative user, you must create an access key for that user.

1. Enter the following command to generate an access key for the administrative user:

   ```
   euare-useraddkey -u alice
   ```

   Eucalyptus returns the access key ID and the user's secret key.

2. Open the `~/.eucarc` file and replace your account credentials you just created, as in this example:

   ```
   export EC2_ACCESS_KEY='WOKSEQRNM1LVIR702XVX1'
   export EC2_SECRET_KEY='0SmLCQ8DAZPKoaC7oJYcRMfeDUgGbiSVv1ip5WaH'
   ```

3. Save and close the file.
4. Open the `~/.iamrc` file and replace your account credentials, as in this example:

   ```
   AWSAccessKeyId=WOKSEQRNM1LVIR702XVX1
   AWSSecretKey=0SmLCQ8DAZPKoaC7oJYcRMfeDUgGbiSVv1ip5WaH
   ```

5. Save and close the file.
6. Switch euca2ools over to using the new credentials.

   ```
   source ~/.eucarc
   ```

Use Case: Create a User

This use case details tasks needed to create a user with limited access.

Create a Group

We recommend that you apply permissions to groups, not users. In this example, we will create a group for users with limited access.

1. Enter the following command to create a group for users who will be allowed create snapshots of volumes in Eucalyptus.

```
euare-groupcreate -g ebs-backup
```

2. Open an editor and enter the following JSON policy:

```
{
  "Statement": [
    {
      "Action": [
        "ec2:CreateSnapshot"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

3. Save and close the file.
4. Enter the following to add the new policy name `allow-snapshot` and the JSON policy file to the `ebs-backup` group:

```
euare-groupuploadpolicy -g ebs-backup -p allow-snapshot -f allow-snapshot.json
```

Create the User

Create the user for the group with limited access.

Enter the following command to create the user `sam` in the group `ebs-backup` and generate a new key pair for the user:

```
euare-usercreate -u sam -g ebs-backup -k
```

Eucalyptus responds with the access key ID and the secret key, as in the following example:

```
AKIAJ25S6IJ5K53Y5GCA
QLKyiCpfjWAvlo9pWqWCbuGB9L3T61w7nYYF057l
```

Accounts

Accounts are the primary unit for resource usage accounting. Each account is a separate name space and is identified by its UUID (Universal Unique Identifier).

Tasks performed at the account level can only be done by the users in the **eucalyptus** account.

Add an Account

To add an account perform the steps listed in this topic.

To add a new account:

Enter the following command:

```
euare-accountcreate -a <account_name>
```

Eucalyptus returns the account name and its ID, as in this example:

```
account01  592459037010
```

Rename an Account

To rename an account perform the steps listed in this topic.

To change an account's name:

Enter the following command:

```
uare-accountaliascreate -a <new_name>
```

List Accounts

To list accounts perform the steps in this topic.

Use the `euare-accountlist` command to list all the accounts in an account or to list all the users with a particular path prefix. The output lists the ARN for each resulting user.

```
euare-userlistbypath -p <path>
```

Delete an Account

To delete an account perform the steps listed in this topic.

Tip: If there are resources tied to the account that you delete, the resources remain. We recommend that you delete these resources first.

Enter the following command:

```
euare-accountdel -a <account_name> -r true
```

Use the `-r` option set to `true` to delete the account recursively. You don't have to use this option if have already deleted users, keys, and passwords in this account.

Eucalyptus does not return any message.

Groups

Groups are used to share resource access authorizations among a set of users within an account. Users can belong to multiple groups.

Important: A group in the context of access is not the same as a security group.

This section details tasks that can be performed on groups.

Create a Group

To create a group perform the steps listed in this topic.

Enter the following command:

```
euare-groupcreate -g <group_name>
```

Eucalyptus does not return anything.

Add a Group Policy

To add a group policy perform the steps listed in this topic.

Enter the following command:

```
euare-groupaddpolicy -g <group_name> -p <policy_name> -e <effect> -a
       <actions> -o
```

The optional `-o` parameter tells Eucalyptus to return the JSON policy, as in this example:

```
{"Version":"2008-10-17","Statement":[{"Effect":"Allow",
"Action":["ec2:RunInstances"], "Resource":["*"]}]}
```

Modify a Group

To modify a group perform the steps listed in this topic.

Modifying a group is similar to a "move" operation. Whoever wants to modify the group must have permission to do it on both sides of the move. That is, you need permission to remove the group from its current path or name, and put that group in the new path or name.

For example, if a group changes from one area in a company to another, you can change the group's path from `/area_abc/` to `/area_efg/`. You need permission to remove the group from `/area_abc/`. You also need permission to put the group into `/area_efg/`. This means you need permission to call `UpdateGroup` on both `arn:aws:iam::123456789012:group/area_abc/*` and `arn:aws:iam::123456789012:group/area_efg/*`.

1. Enter the following command to modify the group's name:

   ```
   euare-groupmod -g <group_name> --new-group-name <new_name>
   ```

   Eucalyptus does not return a message.
2. Enter the following command to modify a group's path:

   ```
   euare-groupmod -g <group_name> -p <new_path>
   ```

   Eucalyptus does not return a message.

Remove a User from a Group

To remove a user from a group perform the steps listed in this topic.

Enter the following command:

```
euare-groupremoveuser -g <group_name> -u <user-name>
```

List Groups

To list groups perform the steps listed in this topic.

Enter the following command:

```
euare-grouplistbypath
```

Eucalyptus returns a list of paths followed by the ARNs for the groups in each path. For example:

```
arn:aws:iam::eucalyptus:group/groupa
```

List Policies for a Group

To list policies for a group perform the steps listed in this topic.

Enter the following command:

```
euare-grouplistpolicies -g <group_name>
```

Eucalyptus returns a listing of all policies associated with the group.

Delete a Group

To delete a group perform the steps listed in this topic.

When you delete a group, you have to remove users from the group and delete any policies from the group. You can do this with one command, using the `euare-groupdel` command with the `-r` option. Or you can follow the following steps to specify who and what you want to delete.

1. Individually remove all users from the group.

   ```
   euare-groupremoveuser -g <group_name> -u <user_name>
   ```

2. Delete the policies attached to the group.

```
euare-groupdelpolicy -g <group_name> -p <policy_name>
```

3. Delete the group.

```
euare-groupdel -g <group_name>
```

The group is now deleted.

Users

Users are subsets of accounts and are added to accounts by an appropriately credentialed administrator. While the term **user** typically refers to a specific person, in Eucalyptus, a **user** is defined by a specific set of credentials generated to enable access to a given account. Each set of user credentials is valid for accessing only the account for which they were created. Thus a user only has access to one account within a Eucalyptus system. If an individual person wishes to have access to more than one account within a Eucalyptus system, a separate set of credentials must be generated (in effect a new ‘user’) for each account (though the same username and password can be used for different accounts).

When you need to add a new user to your Eucalyptus cloud, you'll go through the following process:

1	*Create a user*
2	*Add user to a group*
3	*Give user a login profile*

Add a User

To add a user, perform the steps in this topic.

Enter the following command

```
euare-usercreate -u <user_name> -g <group_name> -k
```

Eucalyptus does not return a response.

Tip: If you include the `-v` parameter, Eucalyptus returns a response that includes the user's ARN and GUID.

Add a User to a Group

To add a user to a group perform the steps listed in this topic.

Enter the following command:

```
euare-groupadduser -g <group_name> -u <user-name>
```

Create a Login Profile

To create a login profile, perform the tasks in this topic.

Enter the following command:

```
euare-useraddloginprofile -u <user_name> -p <password>
```

Eucalyptus does not return a response.

Modify a User

Modifying a user is similar to a "move" operation. To modify a user, you need permission to remove the user from the current path or name, and put that user in the new path or name.

For example, if a user changes from one team in a company to another, you can change the user's path from `/team_abc/` to `/team_efg/`. You need permission to remove the user from `/team_abc/`. You also need permission to put the user into `/team_efg/`. This means you need permission to call UpdateUser on both

`arn:aws:iam::123456789012:user/team_abc/*` and
`arn:aws:iam::123456789012:user/team_efg/*`.

To rename a user:

1. Enter the following command to rename a user:

   ```
   euare-usermod -u <user_name> --new-user-name <new_name>
   ```

   Eucalyptus does not return a message.
2. Enter the following command:

   ```
   euare-groupmod -u <user_name> -p <new_path>
   ```

   Eucalyptus does not return a message.

List Users

To list users within a path, perform the steps in this topic.

Use the `euare-userlistbypath` command to list all the users in an account or to list all the users with a particular path prefix. The output lists the ARN for each resulting user.

```
euare-userlistbypath -p <path>
```

Delete a User

To delete a user, perform the tasks in this topic.

Enter the following command

```
euare-userdel -u <user_name>
```

Eucalyptus does not return a response.

Credentials

Eucalyptus uses different types of credentials for both user and administrative functions. Besides the login and password used for accessing the Eucalyptus Administrator Console, Eucalyptus uses an SSH keypair and an X.509 certificate to control access to instances and to Eucalyptus system functions using the command line tools. This section discusses the various types of credentials and how to use them.

- *Working with User Credentials*
- *Working with Administrator Credentials*

Working with User Credentials

This section describes how to create new user credentials and retrieve existing user credentials.

- *Generating User Credentials*
- *Retrieving Existing User Credentials*
- *Uploading a Certificate*

Generating User Credentials

The first time you get credentials using the `euca_conf` command, a new secret access key is generated. On each subsequent request to get credentials, an existing active secret key is returned. You can also generate new keys using the `euare-useraddkey` command.

Tip: Eucalyptus creates a new private key and X.509 certificate pair each time you request a user's credentials using `euca_conf`.

- To generate a new key for a user by an account administrator, enter the following

  ```
  euare-useraddkey -u <user_name>
  ```

- To generate a private key and an X.509 certificate pair, enter the following:

```
euare-usercreatecert -u <user_name>
```

Retrieving Existing User Credentials

When you retrieve existing credentials, Eucalyptus returns a zip file that contains keys, certificates, a bash script, and several other required files. To use these credentials with such CLI tools as euca2ools or ec2-tools, unzip the zip file to a directory of your choice.

- An administrator with a root access to the machine on which CLC is installed can get credentials using `euca_conf` CLI tool on that machine.

  ```
  /usr/sbin/euca_conf --cred-account <account> --cred-user <user_name>
   --get-credentials <filename>.zip
  ```

  Where <account> and <user_name> are the names of the account and the user whose credentials are retrieved.

  Tip: You can omit the `--cred-account` and `--cred-user` options when you get credentials for the **admin** user of the **eucalyptus** account.

- A user can get his or her credentials by logging in into the Eucalyptus Administrator Console and clicking **Download new credentials** in the drop-down menu at the top of the screen. This will result in a download of a zip file.

  In the following example we download the credentials zip file to `~/.euca`, then change access permissions, as shown:

  ```
  mkdir ~/.euca
  cd ~/.euca
  unzip <filepath>/<creds_zipfile>.zip
  chmod 0700 ~/.euca
  chmod 0600 *
  ```

  Important: The zip file with credentials contains security-sensitive information. We recommend that you remove or read- and write-protect the file from other users after unzipping.

  Alternatively, you can view and copy your access keys and X.509 certificates from the Eucalyptus Administrator Console after logging in, using the Navigation menu.

Uploading a Certificate

To upload a certificate provided by a user:

Enter the following command:

```
euare-useraddcert -u <user_name> -f <cert_file>
```

Working with Administrator Credentials

Important: When you run the following command, you are requesting a new X.509 and a corresponding private key. You cannot retrieve an existing private key.

To generate a set of credentials:

1. Log in to the CLC.
2. Get administrator credentials and source eucarc:

   ```
   /usr/sbin/euca_conf --get-credentials admin.zip
   unzip admin.zip
   chmod 0600 *
   source eucarc
   ```

Roles

A *role* is a mechanism that allows applications to request temporary security credentials on a user or application's behalf.

Note: Eucalyptus roles are managed through the Eucalyptus Euare service, which is compatible with Amazon's Identity and Access Management service. For more information on IAM and roles, see the *Amazon IAM User Guide*.

The suite of AssumeRoles gives you the ability to delegate users certain access to Eucalyptus cloud resources by allowing them to assume certain roles.

Launch an Instance with a Role

To create a role for a Eucalyptus instance, you must first create a trust policy that you can use for it.

Create Trust Policy

You can create trust policies in two ways:

- a file method
- a command line method

Create trust policy using a file

1. Create a trust policy file with the contents below and save it in a text file called `role-trust-policy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "ec2.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Create the role using the `euare-rolecreate` command, specifying the trust policy file that was previously created:

```
# euare-rolecreate --role-name describe-instances -f role-trust-policy.json
# euare-rolelistbypath
arn:aws:iam::408396244283:role/describe-instances
```

3. Proceed with applying an access policy to a role.

Create trust policy using the command line

The other way to create the role is to use the command line options to specify the trust policy:

1. Issue the following string on the command line:

```
# euare-rolecreate --role-name describe-instances --service
http://compute.acme.eucalyptus-systems.com:8773/
# euare-rolelistbypath
arn:aws:iam::408396244283:role/describe-instances
```

2. Proceed with applying an access policy to a role.

Create and apply an access policy to a role

1. Create a policy and save it in a text file with a `.json` extension. The following example shows a policy that allows listing the contents of an S3 bucket called "mybucket":

```
{
  "Statement": [
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::mybucket"
    }
  ]
}
```

Note: For more information on policies, see *Policy Overview*.

2. Apply the access policy to the role using the `euare-roleuploadpolicy` command, passing in the filename of the policy you created in the previous step:

   ```
   euare-roleuploadpolicy --role-name mytestrole --policy-name s3-list-bucket
   --policy-document my-test-policy.json
   ```

Use a Role with an Instance Application

You can use the AWS Java SDK to programmatically perform IAM role-related operations in your Eucalyptus cloud. This example shows how to use the AWS SDK to retrieve the credentials for the IAM role associated with the Eucalyptus instance.

1. The following program lists the contents of the bucket "my-test-bucket" using the credentials stored in the Java system properties:

```
import com.amazonaws.auth.*;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.ClasspathPropertiesFileCredentialsProvider;
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2Client;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;

public class MyTestApp {

    static AmazonEC2 ec2;
    static AmazonS3 s3;

    private static void init() throws Exception {

        AWSCredentialsProvider credentials = new
ClasspathPropertiesFileCredentialsProvider();

        s3 = new AmazonS3Client(credentials);
        s3.setEndpoint("http://128.0.0.1:8773/services/Walrus");
    }


    public static void main(String[] args) throws Exception {

        init();

        try {

            String bucketName = "my-test-bucket";
            System.out.println("Listing bucket " + bucketName + ":");
            ListObjectsRequest listObjectsRequest = new
ListObjectsRequest(bucketName, "", "", "", 200);
            ObjectListing bucketList;
```

```
            do {
                bucketList = s3.listObjects(listObjectsRequest);
                for (S3ObjectSummary objectInfo :
                        bucketList.getObjectSummaries()) {
                    System.out.println(" - " + objectInfo.getKey() + "  " +
                            "(size = " + objectInfo.getSize() +
                            ")");
                }
                listObjectsRequest.setMarker(bucketList.getNextMarker());
            } while (bucketList.isTruncated());

        } catch (AmazonServiceException eucaServiceException ) {
            System.out.println("Exception: " +
eucaServiceException.getMessage());
            System.out.println("Status Code: " +
eucaServiceException.getStatusCode());
            System.out.println("Error Code: " +
eucaServiceException.getErrorCode());
            System.out.println("Request ID: " +
eucaServiceException.getRequestId());
        } catch (AmazonClientException eucaClientException) {
            System.out.println("Error Message: " +
eucaClientException.getMessage());
        }
        System.out.println("===== FINISHED =====");
    }

}
```

This application produces output similar to the following:

```
Listing bucket my-test-bucket:
 - precise-server-cloudimg-amd64-vmlinuz-virtual.manifest.xml  (size = 3553)
 - precise-server-cloudimg-amd64-vmlinuz-virtual.part.0  (size = 4904032)
 - precise-server-cloudimg-amd64.img.manifest.xml  (size = 7014)
 - precise-server-cloudimg-amd64.img.part.0  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.1  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.10  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.11  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.12  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.13  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.14  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.15  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.16  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.17  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.18  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.19  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.2  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.20  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.21  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.22  (size = 2570400)
 - precise-server-cloudimg-amd64.img.part.3  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.4  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.5  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.6  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.7  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.8  (size = 10485760)
 - precise-server-cloudimg-amd64.img.part.9  (size = 10485760)
===== FINISHED =====
```

The problem with this approach is that the credentials are hardcoded into the application - this makes them less secure, and makes the application more difficult to maintain. Using IAM roles is a more secure and easier way to manage credentials for applications that run on Eucalyptus cloud instances.

2. Create a role with a policy that allows an instance to list the contents of a specific bucket, and then launch an instance with that role (for an example, see *Launch an Instance with a Role*. An example policy that allows listing of a specific bucket will look similar to the following:

```
{
  "Statement": [
    {
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::my-test-bucket"
    }
  ]
}
```

3. The following line of code retrieves the credentials that are stored in the application's credentials profile:

   `AWSCredentialsProvider credentials = new ClasspathPropertiesFileCredentialsProvider();`

   To use the role-based credentials associated with the instance, replace that line of code with the following:

   `AWSCredentialsProvider credentials = new InstanceProfileCredentialsProvider();`

   The program now looks like this:

```
import com.amazonaws.auth.*;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.ClasspathPropertiesFileCredentialsProvider;
import com.amazonaws.services.ec2.AmazonEC2;
import com.amazonaws.services.ec2.AmazonEC2Client;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;

public class MyTestApp {

    static AmazonEC2 ec2;
    static AmazonS3 s3;

    private static void init() throws Exception {

        AWSCredentialsProvider credentials = new
InstanceProfileCredentialsProvider();

        s3 = new AmazonS3Client(credentials);
        s3.setEndpoint("http://128.0.0.1:8773/services/Walrus");
    }


    public static void main(String[] args) throws Exception {

        init();

        try {

            String bucketName = "my-test-bucket";

            System.out.println("Listing bucket " + bucketName + ":");
            ListObjectsRequest listObjectsRequest = new
ListObjectsRequest(bucketName, "", "", "", 200);
            ObjectListing bucketList;
            do {
                bucketList = s3.listObjects(listObjectsRequest);
                for (S3ObjectSummary objectInfo :
                        bucketList.getObjectSummaries()) {
                    System.out.println(" - " + objectInfo.getKey() + "  " +
                            "(size = " + objectInfo.getSize() +
                            ")");
                }
                listObjectsRequest.setMarker(bucketList.getNextMarker());
            } while (bucketList.isTruncated());

        } catch (AmazonServiceException eucaServiceException ) {
            System.out.println("Exception: " +
eucaServiceException.getMessage());
            System.out.println("Status Code: " +
eucaServiceException.getStatusCode());
            System.out.println("Error Code: " +
eucaServiceException.getErrorCode());
            System.out.println("Request ID: " +
eucaServiceException.getRequestId());
        } catch (AmazonClientException eucaClientException) {
            System.out.println("Error Message: " +
eucaClientException.getMessage());
        }
        System.out.println("===== FINISHED =====");
    }

}
```

NOTE: Running this code outside of an instance will result in the following error message:

```
Listing bucket my-test-bucket:
Error Message: Unable to load credentials from Amazon EC2 metadata service
```

When the application is running on an instance that was launched with the role you created, the credentials for the role assigned to the instance will be retrieved from the Instance Metadata Service.

Assume a Role

A role is assigned a specific set of tasks and permissions. Users may assume a different role than the one they have in order to perform a different set of tasks. For example, the primary administrator is unavailable and the backup administrator is asked to assume the role of the primary administrator during his or her absence.

A few points to consider before assuming a role:

- A role must first be set up by an administrator.
- You must log in as an IAM user, not as an account root user.
- Once you assume another role, you temporarily give up your existing user permissions and assume the permissions of your new role.
- When you are no longer assuming another role, your usual user permissions are automatically restored.

Create Role

The scenario described in this section outlines the procedure for creating a role in order to delegate permissions to an IAM user.

Create a role that allows users of an account to manage keypairs. Management of keypairs include the following EC2 actions:

- CreateKeyPair
- DeleteKeyPair
- DescribeKeyPairs
- ImportKeyPair

1. Create a role for managing keypairs for the account. In this example, the admin user of 'devops' account (001827830003) is creating the role:

```
# cat devops-role-trustpolicy.json
{
 "Version": "2012-10-17",
 "Statement": [{
 "Effect": "Allow",
 "Principal": {"AWS": "arn:aws:iam::001827830003:root"},
 "Action": "sts:AssumeRole"
 }]
}

# euare-rolecreate -f devops-role-trustpolicy.json devops-ec2-keypair-mgmt-role
 --region devops-admin@future
```

2. Add IAM access policy for keypair management to the role:

```
# cat keypair-mgmt-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1445362739663",
      "Action": [
        "ec2:CreateKeyPair",
        "ec2:DeleteKeyPair",
        "ec2:DescribeKeyPairs",
        "ec2:ImportKeyPair"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}

# euare-roleuploadpolicy --policy-name ec2-keypair-actions --policy-document
keypair-mgmt-policy.json devops-ec2-keypair-mgmt-role --region
devops-admin@future
```

3. Now that the role has been created, follow the *AWS IAM best practice of using groups to assign permission to IAM users* and attach an IAM access policy to the group to allow any members (example shows 'user01' user) to assume the 'devops-ec2-keypair-mgmt-role' role:

```
# euare-groupcreate -g Key-Managers --region devops-admin@future
# euare-groupadduser -u user01 -g Key-Managers --region devops-admin@future
# cat devops-keypair-mgmt-assume-role-policy.json
{
 "Version": "2012-10-17",
 "Statement": [{
 "Effect": "Allow",
 "Action": "sts:AssumeRole",
 "Resource": "arn:aws:iam::001827830003:role/devops-ec2-keypair-mgmt-role"
 }]
}
# euare-groupuploadpolicy -p keypair-mgmt-role-perm -f
devops-keypair-mgmt-assume-role-policy.json Key-Managers --region
devops-admin@future
```

4. Now that members can assume the 'devops-ec2-keypair-mgmt-role' role, run the following command to list all keypairs under the account:

```
# eval `/usr/bin/euare-assumerole devops-ec2-keypair-mgmt-role --region
devops-user01@future`
# euca-describe-keypairs --region @future
KEYPAIR devops-admin 9e:1a:bc:ac:98:b1:97:7c:65:b0:b3:7c:96:f5:d5:7b:a1:3e:36:a6
```

5. When done assuming the role, the role must be released using `euare-releaserole`:

```
# eval `/usr/bin/euare-releaserole --region devops-user01@future`
```

For information about Euca2ools IAM commands, see *IAM-Compatible Commands*.

Delegate Access Across Your Accounts Using Roles

A role can be used to delegate access to resources that are in different accounts that you own.

Using roles to access resources across different accounts allows users to assume a role that can access all the resources in in different acccounts, rather than having users log into different accounts to achieve the same result.

Using Roles to Access Resources in Another Account

The scenario described in this section outlines the procedure for a user in Account B to create a role that provides access to a particular OSGObject Storage Gateway (OSG) bucket owned by Account B, which can be assumed by user in Account A.

1. Using *s3cmd*, list bucket that will be shared through role:

```
# ./s3cmd/s3cmd --config=.s3cfg-acctB-user11 ls s3://mongodb-snapshots
2014-12-01 22:34 188563920   s3://mongodb-snapshots/mongodb-backup-monday.img.xz
2014-12-02 13:34 188564068
s3://mongodb-snapshots/mongodb-backup-tuesday.img.xz
```

2. Create Role in Account B with Trust Policy for User from Account A:

```
# cat acctB-role-trust-acctA-policy.json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::290122656840:user/user01"},
    "Action": "sts:AssumeRole"
  }]
}

# euare-rolecreate --role-name cross-bucket-access-mongodb-logs
--policy-document acctB-role-trust-acctA-policy.json
```

3. Upload IAM Access Policy for Role in Account B:

```
# cat acctB-mongodb-snapshots-bucket-access-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::mongodb-snapshots"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::mongodb-snapshots/*"
    }
  ]
}

# euare-roleuploadpolicy --role-name cross-bucket-access-mongodb-logs
--policy-document acctB-mongodb-snapshots-bucket-access-policy.json
--policy-name mongodb-logs-bucket-access
```

4. Upload IAM access policy to Group (e.g. Testers) associated with user in Account A to allow for Role in Account B to be assumed. For more information, go to *Amazon Web Services IAM Best Practices*.

```
# cat acctA-assume-role-acctB-policy.json
{
  "Statement": [
    {
      "Sid": "Stmt1417531456446",
      "Action": [
        "sts:AssumeRole"
      ],
      "Effect": "Allow",
      "Resource":
"arn:aws:iam::325271821652:role/cross-bucket-access-mongodb-logs"
    }
  ]
}

# euare-groupuploadpolicy --policy-name mongodb-bucket-access-role --group-name
 Testers --policy-document acctA-assume-role-acctB-policy.json
```

5. The example below demonstrates how to use a python script leveraging the *boto* library. Another way to assume this role is to run the Euca2ools command, `euare-assumerole`, using the AccountA/user01 credentials. For more information regarding assuming a role, see an example from the *Assume a Role* section. The script below performs the following actions:

- Accesses STS to get temporary access key, secret key and token
- List contents of bucket "mongodb-snapshots"

```
=======================
#!/bin/env python

import boto
from boto.sts import STSConnection
from boto.s3.connection import S3Connection
from boto.s3.connection import OrdinaryCallingFormat

if __name__ == "__main__":
    """
    Assuming 'cross-bucket-access-mongodb-logs' role by AccountA, User01 user

    """
    STSConnection.DefaultRegionEndpoint =
"tokens.future.euca-hasp.cs.prc.eucalyptus-systems.com"
    sts_connection = STSConnection(aws_access_key_id="<AccountA User01 Access
 Key ID>",
    aws_secret_access_key="<AccountA User01 Secret Key>",
    is_secure=False, port="8773")
    assumedRoleObject = sts_connection.assume_role(
   role_arn="arn:aws:iam::325271821652:role/cross-bucket-access-mongodb-logs",

    role_session_name="AcctAUser01MongoDBBucketAccess")

    s3 =
S3Connection(aws_access_key_id=assumedRoleObject.credentials.access_key,
    aws_secret_access_key=assumedRoleObject.credentials.secret_key,
    security_token=assumedRoleObject.credentials.session_token,
    host="objectstorage.future.euca-hasp.cs.prc.eucalyptus-systems.com",
    is_secure=False, port=8773, calling_format=OrdinaryCallingFormat())

    bucket_name = "mongodb-snapshots"
    bucket = s3.lookup(bucket_name)
    if bucket:
        print "Bucket Information [%s]:" % bucket_name
        print "-------------------------------------------------------------"
        for key in bucket:
            print "\t" + key.name
    else:
        print "Bucket is not available: " + bucket_name + "\n"
==================
```

6. Run the script:

```
# ./describe-bucket-script.py
Bucket Information [mongodb-snapshots]:
------------------------------------------------------------
 mongodb-backup-monday.img.xz
 mongodb-backup-tuesday.img.xz
```

For information about Euca2ools IAM commands, see *IAM-Compatible Commands*.

IAM Guide History

This section contains information about changes to the IAM documentation in this release.

Section / Topic	Description of Change	Date Changed
Roles / Assume a Role	Added new section to support the ability for assuming a role.	December 7, 2015
Roles / Delegate Access Across Your Accounts Using Roles	Added new section to describe scenarios and method for allowing users to access multiple accounts.	December 7, 2015
New release	Manage Users and Groups moved from Administration Guide.	October 22, 2015

Index