S3 INTRODUCTORY EXAMPLES

SOLVED community

Purpose: This “cheat sheet” works as a good entry point for getting to know S3 for folks that are new to it. To start, we’re going to make the assumption that you have a set of account-level keys generated per the Scality documentation for creating an S3 account.

CONFIGURING THE CLI

All cheating in this document is to be done with the AWS CLI utility (aws). No other tools like s3cmd or GUIs like Cyberduck are covered here. This is because the AWS CLI is the only really good client out there. The “free” GUI clients are generally poor-quality and why not use the de facto Amazon tool?
Get the client from the Amazon CLI page or install the “awscli” package if you’re on a Linux distribution..
Configure the aws cli with your account keys:

```
aws configure --profile s3c-account
```

Use whatever name you like for “s3c-account”. Pro tip: set “us-east-1” as your default region when prompted to avoid client weirdness later.

USE IAM ROLES FOR APPLICATIONS AND SERVICES THAT REQUIRE S3 ACCESS

Create an alias to simplify accessing your S3 install as the “aws” tool defaults to Amazon’s cloud services. If your endpoint is clear HTTP this will do the trick:

```
alias s3caws='aws --endpoint=http://valid.s3.endpoint.com --profile s3c-account'
```

If your load-balancer or endpoint is HTTPS and you used your own CA to sign you’ll need to set the AWS_CA_BUNDLE variable or provide it in the alias:

```
alias s3caws='aws --endpoint=https://valid.s3.endpoint.com --profile s3c-account
--ca-bundle==/etc/pki/tls/certs/my_ca.crt'
```

BASIC S3 ACTIONS

AWS CLI reference for basic actions can be found in the Amazon CLI Reference.

CHEAT-SHEET EXAMPLES

create a bucket:

```
s3caws s3 mb s3://mynewbucket
```

list buckets:

```
s3caws s3 ls
```

put an object in a bucket:

```
s3caws s3 cp /tmp/myfile s3://mynewbucket
```

list objects in a bucket:

```
s3caws s3 ls s3://mynewbucket
```

get an object out of a bucket:

```
s3caws s3 cp s3://mynewbucket/myfile /tmp/
```

check object metadata:

```
s3caws s3api head-object --bucket mynewbucket --key myfile
```

That last one uses the more complex "s3api" interface which can be documented in the s3api reference page for the AWS CLI.

USING IAM (SCALITY VAULT)

Vault is the Scality IAM implementation with a feature-set built around S3. See the full IAM documentation for a good time.

BASIC USER/GROUP MANAGEMENT

Creating groups:

```
s3caws iam create-group --group-name=electricmayehm
s3caws iam create-group --group-name=hecklers
```

Create some users:

```
s3caws iam create-user --user-name drteeth
s3caws iam create-user --user-name zoot
s3caws iam create-user --user-name janis
s3caws iam create-user --user-name animal
s3caws iam create-user --user-name statler
s3caws iam create-user --user-name waldorf
```

Add users to groups:

```
s3caws iam add-user-to-group --group-name electricmayehm --user-name drteeth
s3caws iam add-user-to-group --group-name electricmayehm --user-name zoot
s3caws iam add-user-to-group --group-name electricmayehm --user-name janis
s3caws iam add-user-to-group --group-name electricmayehm --user-name animal
s3caws iam add-user-to-group --group-name hecklers --user-name statler
s3caws iam add-user-to-group --group-name hecklers --user-name waldorf
```

Generate a key pair for a user:

```
s3caws iam create-access-key --user-name statler
{
   "AccessKey": {
      "UserName": "statler",
      "Status": "Active",
      "SecretAccessKey": "QfSP8onXI0vo=gtsLHoH6+EH72pMN34BFjxnnDfY",
      "AccessKeyId": "97Y1VRM0Y31CNUK5YZSA"
   }
}
```

Show group members:

```
s3caws iam get-group --group-name hecklers
s3caws iam get-group --group-name electricmayehm
```

IAM POLICY CREATION

IAM policies are rules applied to IAM entities (users/groups) that can be used to control access to resources. In the case of Scality, those resources will be S3-related (or IAM itself). Users and groups are (almost) meaningless if there are no policies attached to them. Policy creation is a complex topic - it is important to understand that these examples aren't really useful, just examples. The IAM documentation should be referenced for understanding IAM but once you know what you're doing you can use the AWS policy generator to create valid JSON for use with S3C.
(select "IAM Policy" in the drop-down, then "Amazon S3" for the service)

IAM POLICY EXAMPLES

Scality implements a lot of IAM actions. See the Vault API Reference for a full list of supported actions.
Create a couple of simple policies:

```
mkdir ~/policies
cat << EOF > ~/policies/all-access.policy
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid":"AllowAll",
         "Effect":"Allow",
         "Action":"s3:*",
         "Resource": "*"
      }
   ]
}
EOF
cat << EOF > ~/policies/all-deny.policy
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid":"DenyAll",
         "Effect":"Deny",
         "Action":"s3:*",
         "Resource": "*"
      }
   ]
}
EOF
```

Load the policy description documents:

```
s3caws iam create-policy --policy-name all-access \
--policy-document file:///root/policies/all-access.policy
awsiam iam create-policy --policy-name no-access \
--policy-document file:///root/policies/all-deny.policy
```

Attach a policy to a user (replacing policy ARN with the ARN output when doing the “create-policy” above:

```
s3caws iam attach-user-policy --user-name drteeth \
--policy-arn “arn:aws:iam::017061715665:policy/all-access”
```

Here’s an example “home directory” type policy:

```
cat << EOF > /tmp/homedirs.txt
{
   “Version”: “2012-10-17”,
   “Statement”: [
      {
         “Sid”: “AllowGroupToSeeBucketList”,
         “Action”: [“s3:ListAllMyBuckets”, “s3:GetBucketLocation”],
         “Effect”: “Allow”,
         “Resource”: [“arn:aws:s3:::*”]
      },
      {
         “Sid”: “AllowRootLevelListingOfThisBucketAndHomePrefix”,
         “Action”: [“s3:ListBucket”],
         “Effect”: “Allow”,
         “Resource”: [“arn:aws:s3:::mybucket”],
         “Condition”: {“StringEquals”:{“s3:prefix”: [“”,”home/”],”s3:delimiter”:[“/”]}}
      },
      {
         “Sid”: “AllowListBucketofASepecificUserPrefix”,
         “Action”: [“s3:ListBucket”],
         “Effect”: “Allow”,
         “Resource”: [“arn:aws:s3:::mybucket”],
         “Condition”: {“StringLike”: {“s3:prefix”:[“home/${aws:username}/*”]}}
      },
      {
        “Sid”: “AllowUserFullAcccesstoJustSpecificUserPrefix”,
         “Action”: [“s3:*”],
         “Effect”: “Allow”,
         “Resource”: [“arn:aws:s3:::mybucket/home/${aws:username}”,
         “arn:aws:s3:::mybucket/home/${aws:username}/*”]
      }
   ]
}
EOF
```

Create the actual policy:

```
s3caws iam create-policy --policy-name homedir --policy-document
file:///tmp/homedirs.txt
```

Note the returned Policy-ARN and attach the homedir policy to a group:

```
s3caws iam attach-group-policy --group-name hecklers --policy-arn <Policy-ARN noted
from homedir policy creation A1>
```

Attach users to the group:

```
s3caws iam add-user-to-group --group-name hecklers --user-name statler
s3caws iam add-user-to-group --group-name hecklers --user-name waldorf
```

BUCKET ACLS

Simple, coarse-grained (read, write, read/write) access permissions for buckets. Complete documentation on managing bucket ACLs via the AWS CLI can be found in the AWS CLI ACL reference:

Get current bucket ACLs:

```
s3caws s3api get-bucket-acl --bucket examplebucket
{
   “Owner”: {
      “DisplayName”: “johndoe”,
      “ID”: “d35f.....”
   },
   “Grants”: [
      {
         “Grantee”: {
            “Type”: “CanonicalUser”,
            “DisplayName”: “johndoe”,
            “ID”: “d35f.....”
         },
         “Permission”: “FULL_CONTROL”
      }
   ]
}
```

Set a bucket ACL:

```
s3caws s3api put-bucket-acl --bucket examplebucket --grant-read
emailaddress=someone@example.com
```

BUCKET POLICIES

Bucket policies are available in S3C. See the AWS bucket policy examples to get a good feel for what you can do with them. You can also use the AWS policy generator to create valid JSON for bucket policies.

(select “S3 Bucket Policy” in the drop-down)

Create a bucket policy document. This one will restrict access to a bucket by IP address:

```
echo << EOF > /tmp/ip_policy.json
{
   “Version”: “2012-10-17”,
   “Id”: “SourceIP”,
   “Statement”: [{
      “Sid”: “SourceIP”,
      “Effect”: “Deny”,
      “Principal”: “*”,
      “Action”: “s3:*”,
      “Resource”: [
         “arn:aws:s3:::examplebucket”,
         “arn:aws:s3:::examplebucket/*”
      ],
      “Condition”: {
         “NotIpAddress”: {
            “aws:SourceIp”: [
               “11.11.11.11/32”,
               “22.22.22.22/32”
            ]
         }
      }
   }]
}
EOF
```

You can apply the policy to the bucket with AWS CLI using the alias created above:

```
s3caws s3api put-bucket-policy --bucket examplebucket --policy
file:///tmp/ip_policy.json
```

OBJECT TAGGING

You can tag objects on creation or after the fact. Here's an object getting tagged during creation:

```
s3caws s3api put-object --bucket mybucket --key myobject --tagging 'color=blue' --body ./myobject
s3caws s3api get-object-tagging --bucket mybucket --key myobject
{
   "TagSet": [
      {
         "Value": "blue",
         "Key": "color"
      }
   ]
}
```

You can also tag things after upload:

```
galaxy_aws s3api put-object-tagging --bucket mybucket --key myobject --tagging '{"TagSet": [{"Key": "thoughts",
"Value": "you are wrong"}]}'
s3caws s3api get-object-tagging --bucket mybucket --key myobject
{
   "TagSet": [
      {
         "Value": "you are wrong",
         "Key": "thoughts"
      }
   ]
}
```

That's all for now. Feel free to open a ticket to suggest updates.