DEV323

Introduction to the AWS CLI

Leo Zhadanovsky
Principal Solutions Architect
AWS

November 28, 2017

AWS re:Invent

aws

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. In this session, we introduce the AWS CLI and how to use it to automate common administrative tasks in AWS. We cover several features and usage patterns including Amazon EBS snapshot management and Amazon S3 backups. We show how to combine AWS CLI features to create powerful tools for automation. See how to develop, debug, and deploy these examples in several live, end-to-end examples.

Agenda

- Basics
  - Installation
  - Configuration
  - Syntax
- Foundations
  - Profiles
  - Environment Variables
  - Roles
- Advanced
  - Querying & Filtering
  - MFA
  - S3 Commands

AWS Command Line Interface

Unified tool to manage your AWS Services

Basics

- Installing the CLI
- Setting up Credentials
- Configuration Files
- Syntax
- Help

Install the CLI

- Requirements
  - Python 2 version 2.6.5+ or Python 3 version 3.3+
  - OS: Windows, Linux, MacOS, or Unix
  - Installation Options
    - pip
    - Bundled Installer
    - MSI
  - virtualenv recommended when using pip

Install the CLI

```
$ pip install awscli
```

Update the CLI

```
$ pip install awscli --upgrade
```

Check version of the CLI

```
$ aws --version
```

Uninstall the CLI

```
$ pip uninstall awscli
```

Setting up Credentials

```
$ aws configure
AWS Access Key ID [None]:AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]:wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]:us-east-1
Default output format [None]:json
```

You then have to configure the CLI to work with your AWS account as above

Configuration files

~/.aws/credentials

- Supported by all AWS SDKs
- Contains credentials

~/.aws/config

- Some settings used only by CLI
- Can contain credentials

There are 2 different configuration files that control the CLI.

Configuration files

~/.aws/credentials

```
[prod]
aws_access_key_id = foo
aws_secret_access_key = bar
```

~/.aws/config

```
[profile prod]
aws_access_key_id = foo
aws_secret_access_key = bar
```

The profile you want is [prod], you then put your keys

Configuration files

~/.aws/credentials

```
[prod]
aws_access_key_id = foo
aws_secret_access_key = bar
```

~/.aws/config

```
[profile prod]
aws_access_key_id = foo
aws_secret_access_key = bar
region = us-east-1
emr =
  service_role = EMR_DefaultRole
  instance_profile = EMR_EC2_DefaultRole
  log_uri = s3://myBucket/logs
  enable_debugging = True
  key_name = myKeyName
  key_pair_file = /home/myUser/myKeyName.pem
output = json
```

You can do more with your config file as above. You can have a different set of these settings for each profile you want to be able to use

Syntax

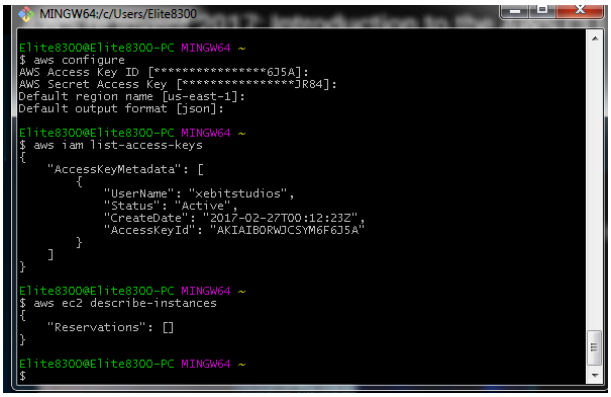
$ aws ec2 describe-instances

service (command) operation (subcommand)

Syntax

$ aws iam list-access-keys

service (command) operation (subcommand)



Help

$ aws ec2 help

This gives you the man-page with the syntax, expected results, the available sub-commands, etc

Help

```
[leozh@dca904790543 ~]$ aws ec2 help
```

Help

```
EC2()                                                    EC2()


NAME
       ec2 -

DESCRIPTION
       Amazon  Elastic Compute Cloud (Amazon EC2) provides resizable computing
       capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 elim-
       inates your need to invest in hardware up front, so you can develop and
       deploy applications faster.

AVAILABLE COMMANDS
       o accept-reserved-instances-exchange-quote

       o accept-vpc-peering-connection

       o allocate-address

       o allocate-hosts

       o assign-ipv6-addresses

:
```

AWS re:Invent

aws

Help

```
[leozh@dca904790543 ~]$ aws ec2 help
[leozh@dca904790543 ~]$ aws ec2 describe-instances help
```

Help

```
DESCRIBE-INSTANCES()                                    DESCRIBE-INSTANCES()


NAME
       describe-instances -

DESCRIPTION
       Describes one or more of your instances.

       If you specify one or more instance IDs, Amazon EC2 returns information
       for those instances. If you do not specify instance IDs, Amazon EC2
       returns information for all relevant instances. If you specify an
       instance ID that is not valid, an error is returned. If you specify an
       instance that you do not own, it is not included in the returned
       results.

       Recently terminated instances might appear in the returned results.
       This interval is usually less than one hour.

       If you describe instances in the rare case where an Availability Zone
       is experiencing a service disruption and you specify instance IDs that
       are in the affected zone, or do not specify any instance IDs at all,
       the call fails. If you describe instances and specify only instance IDs
:
```

AWS re:Invent

aws

Foundations

- Configuration Settings and Precedence
- Named Profiles
- Environment Variables
- Command Line Options
- Roles
- Output Types
- Tab Completion
- aws-shell
- Bastion Hosts

Configuration Settings and Precedence

1. **Command line options** – region, output format and profile can be specified as command options to override default settings.
2. Environment variables – AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN.
3. **The AWS credentials file** – located at ~/.aws/credentials on Linux, macOS, or Unix, or at C:\Users*USERNAME* \.aws\credentials on Windows. This file can contain multiple named profiles in addition to a default profile.
4. **The CLI configuration file** – typically located at ~/.aws/config on Linux, macOS, or Unix, or at C:\Users*USERNAME* \.aws\config on Windows. This file can contain a default profile, named profiles, and CLI specific configuration parameters for each.
5. **Container credentials** – provided by Amazon EC2 Container Service on container instances when you assign a role to your task.

You also have an ***instance_role*** that you can also use

Named Profiles

~/.aws/credentials

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYE
XAMPLEKEY

[user2]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbE
XAMPLEKEY
```

~/.aws/config

```
[default]
region=us-east-1
output=json

[profile user2]
region=us-west-2
output=text
```

Most customers usually end up with multiple AWS accounts having multiple profiles and user roles with different credentials. You need to put the details in your credentials file with the correct profile names, you then put the details for what they can access also in your config file as above

Using Profiles

```
$ aws ec2 describe-instances --profile user2
```

This specifies which profile and credentials you want to use for the execution

Environment Variables

- AWS_ACCESS_KEY_ID – AWS access key.
- AWS_SECRET_ACCESS_KEY – AWS secret key. Access and secret key variables override credentials stored in credential and config files.
- AWS_SESSION_TOKEN – Specify a session token if you are using temporary security credentials.
- AWS_DEFAULT_REGION – AWS region. This variable overrides the default region of the in-use profile, if set.
- AWS_DEFAULT_OUTPUT – Change the AWS CLI's output formatting to json, text, or table.
- AWS_PROFILE – name of the CLI profile to use. This can be the name of a profile stored in a credential or config file, or default to use the default profile.
- AWS_CA_BUNDLE – Specify the path to a certificate bundle to use for HTTPS certificate validation.
- AWS_SHARED_CREDENTIALS_FILE – Change the location of the file that the AWS CLI uses to store access keys.
- AWS_CONFIG_FILE – Change the location of the file that the AWS CLI uses to store configuration profiles.

Using Environment Variables

Linux, macOS, or Unix

```
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
$ export
AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYE
XAMPLEKEY
$ export AWS_DEFAULT_REGION=us-west-2
```

Windows

```
> set  AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
> set
AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEX
AMPLEKEY
> set AWS_DEFAULT_REGION=us-west-2
```

Above is how you can use your environment variables from the CLI, you use the ***set*** command as above

Command Line Options

- **--profile** – name of a profile to use, or "default" to use the default profile.
- **--region** – AWS region to call.
- **--output** – output format.
- **--endpoint-url** – The endpoint to make the call against. The endpoint can be the address of a proxy or an endpoint URL for the in-use AWS region. Specifying an endpoint is not required for normal use as the AWS CLI determines which endpoint to call based on the in-use region.

* The above options override the corresponding profile settings for a single operation.

The options for the ***- -output*** are JSON, table and text. The snowball device has a S3 endpoint_url that you can use with the ***- -endpoint_url*** option. Note that ***the command line options you set is not saved in your config file but only used for the execution of the specific command you are making from the CLI***.

Roles

- Set of permissions granted to a trusted entity
- Assumed by **IAM users**, applications or AWS services like EC2
- Use case:
  - Cross-services
  - Temporary access
  - Cross-account
  - Federation
- Benefits
  - Security: no sharing of secrets
  - Control: revoke access anytime

Roles are IAM policies that can be granted to trusted users. If I have an IAM user that can only assume other roles, we can't do much with this default credential for this user. Then I can assume a read-only role to run ***$ aws ec2 describe-instances*** command for an audit, I can assume a power user role to be able to spin up EC2 instances, or assume an Admin role to be able to modify IAM users and other IAM settings, etc.

Roles

~/.aws/config

```
[profilemarketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadmin
source_profile = default
```

This is the IAM console where we already have a role called ***marketingadmin*** that grants access to S3 so that the user assuming this role can read and edit S3 buckets for their static sites. Above is the way we can assume this role from the CLI to be able to do things in S3. (note, line 1 should be [***profile marketingadmin***]).

Output Types

json

```
{
   "Places": [
       {
            "City": "Seattle",
      "State": "WA"
      },
      {
            "City": "Las Vegas",
            "State": "NV"
      }
   ]
}
```

text

```
PLACES  Seattle    WA
PLACES  Las Vegas  NV
```

table

```
----------------------
|     SomeOperationName  |
+------------------------+
||         Places      ||
|+------------+---------+|
|| City       | State   ||
|+------------+---------+|
|| Seattle    | WA      ||
|| Las Vegas  | NV      ||
|+------------+---------+|
```

AWS re:Invent

aws

Tab Completion Setup

```
$ which aws_completer
/usr/local/bin/aws_completer

$ complete -C '/usr/local/bin/aws_completer' aws
```

* assuming bash

We also have tab completion in the CLI to help with commands and sub-commands by doing ***$ aws <tab>***

Tab Completion

```
[leozh@dca904790543 ~]$ aws
Display all 106 possibilities? (y or n)
```

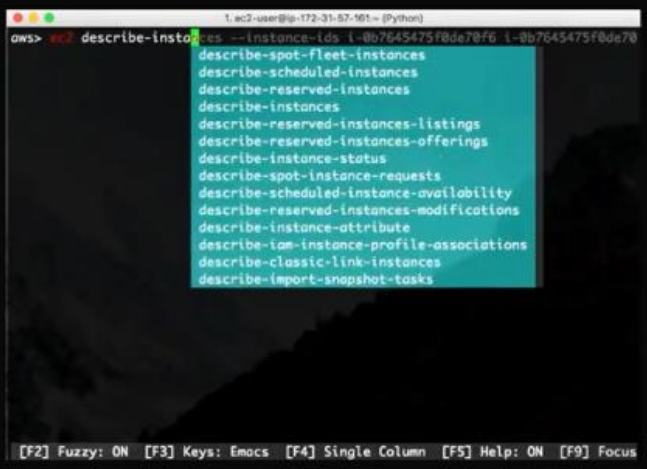
Tab Completion

```
acm                          greengrass
apigateway                   health
application-autoscaling      iam
appstream                    importexport
athena                       inspector
autoscaling                  iot
batch                        iot-data
budgets                      kinesis
clouddirectory               kinesisanalytics
cloudformation               kms
cloudfront                   lambda
cloudhsm                     lex-models
cloudhsmv2                   lex-runtime
cloudsearch                  lightsail
cloudsearchdomain            logs
cloudtrail                   machinelearning
cloudwatch                   marketplace-entitlement
codebuild                    marketplacecommerceanalytics
codecommit                   meteringmarketplace
codepipeline                 mgh
codestar                     mturk
cognito-identity             opsworks
cognito-idp                  opsworks-cm
cognito-sync                 organizations
--More--
```

AWS Shell

An integrated shell for working with the AWS CLI

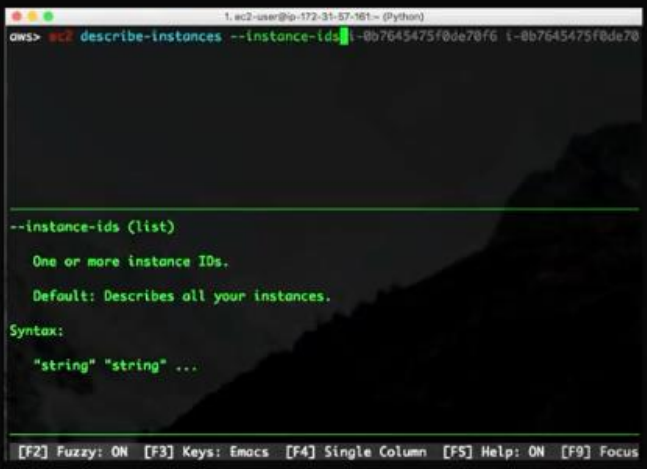
Get it here:
https://github.com/awslabs/aws-shell



AWS Shell

An integrated shell for working with the AWS CLI

Get it here:
https://github.com/awslabs/aws-shell



The AWS Shell gives you auto-completion

AWS Shell

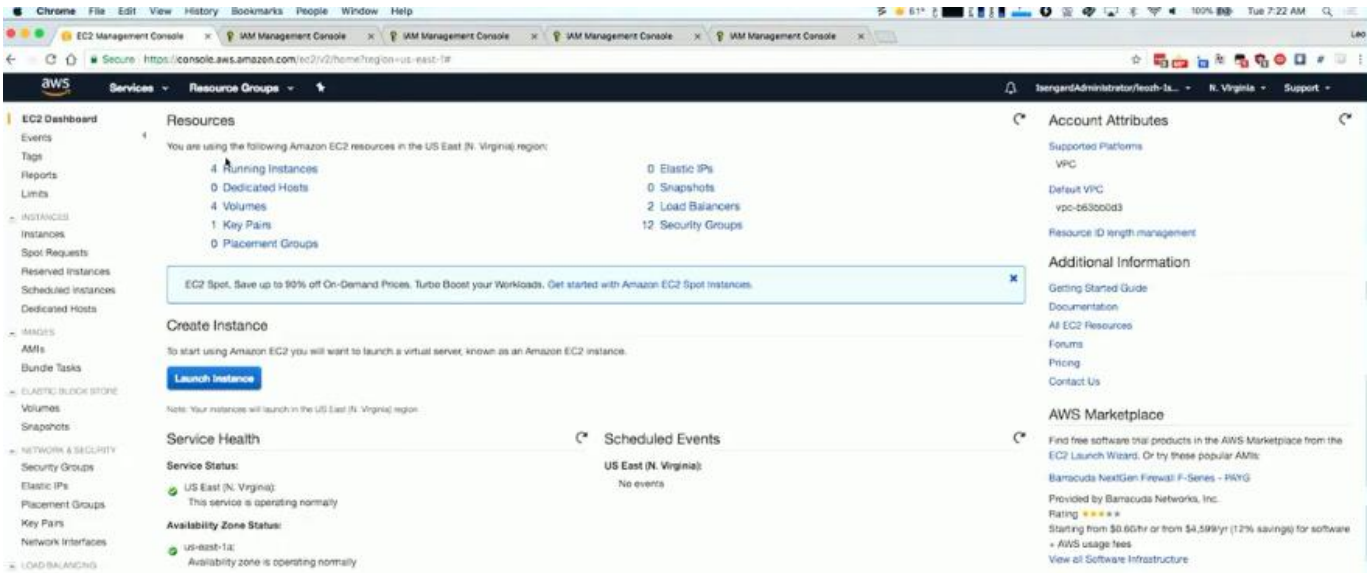
An integrated shell for working with the AWS CLI

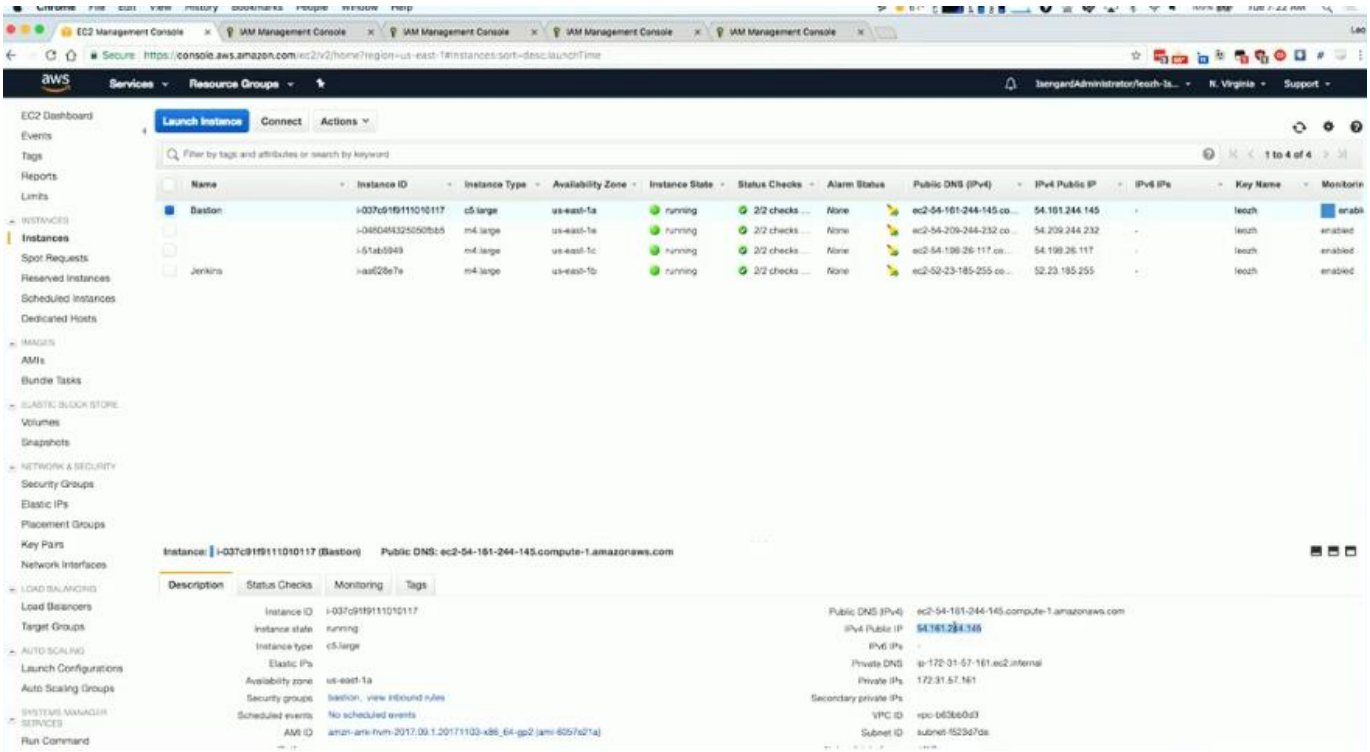
Get it here:
https://github.com/awslabs/aws-shell

Bastion Hosts

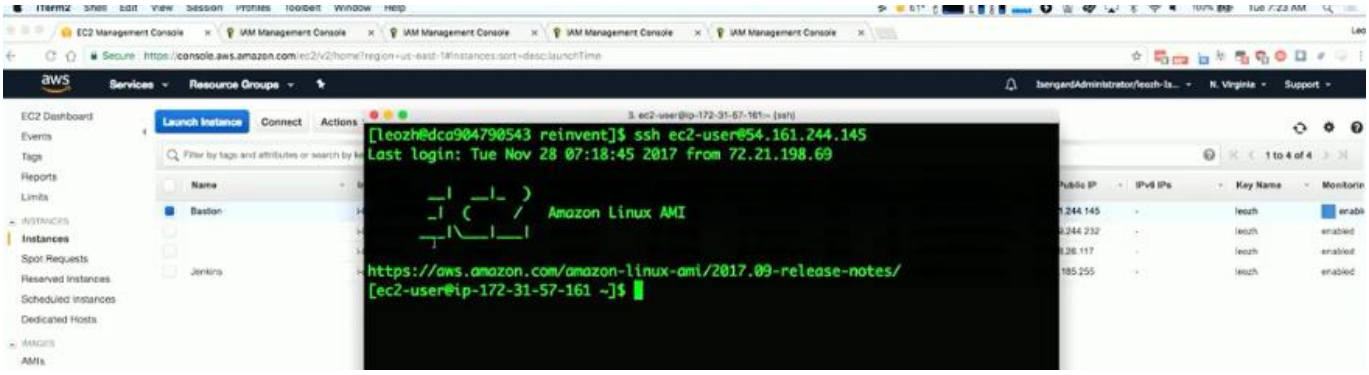
Demo

Instead of issuing static IAM roles for myself, I can have an EC2 instance that has a role that gives it access to do things using the AWS SDKs and AWS APIs. We then do not need to have statics credentials for ourselves anymore.

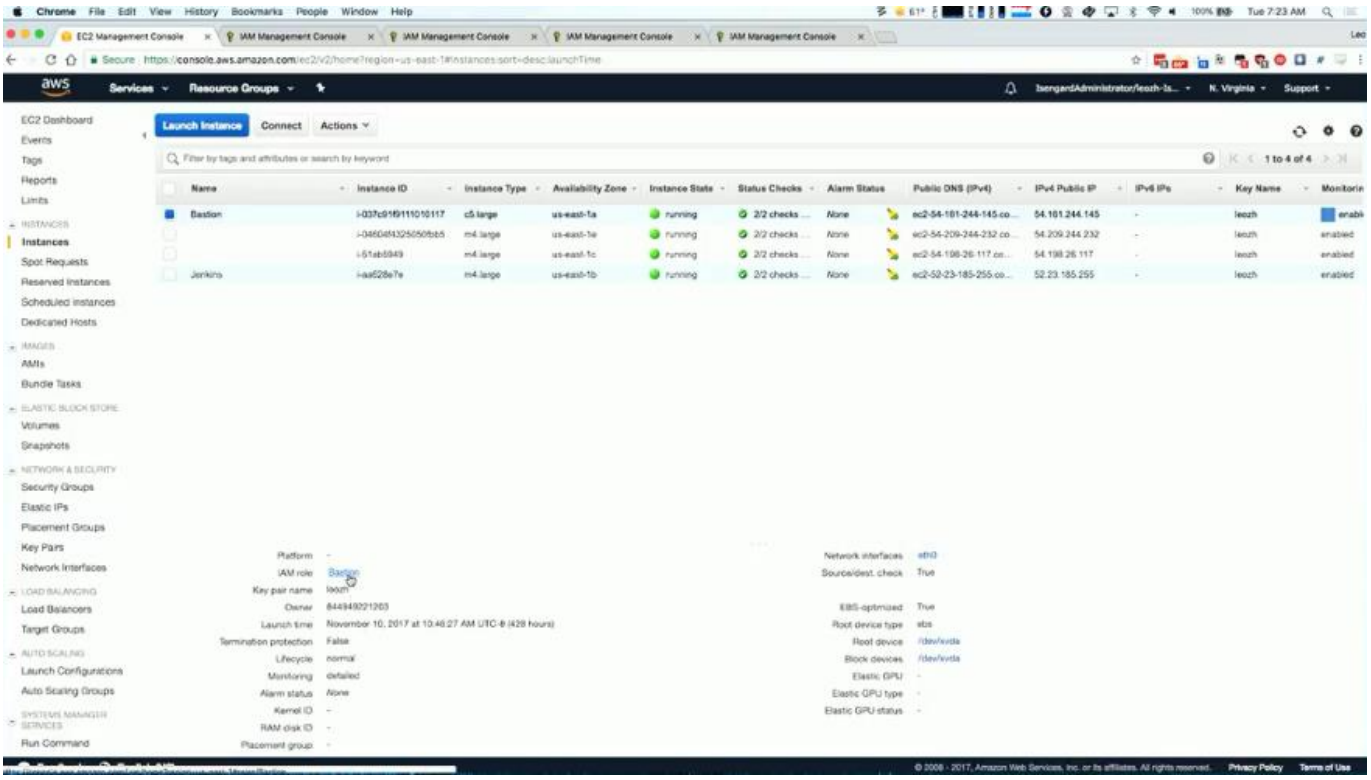




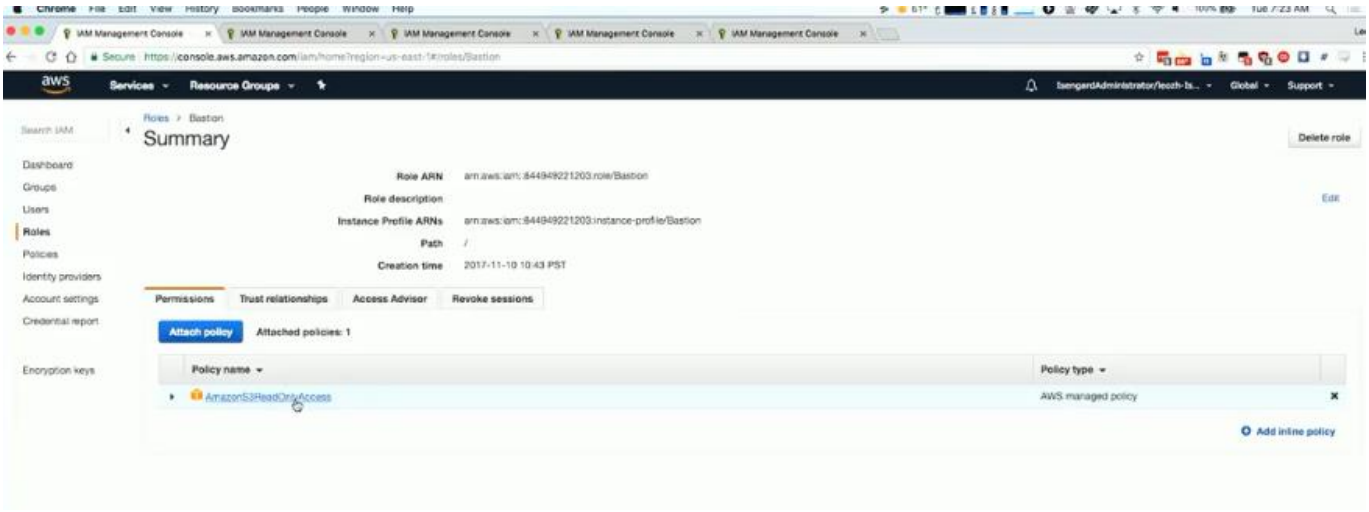
We copy the EC2 instance IP address and SSH into it as below



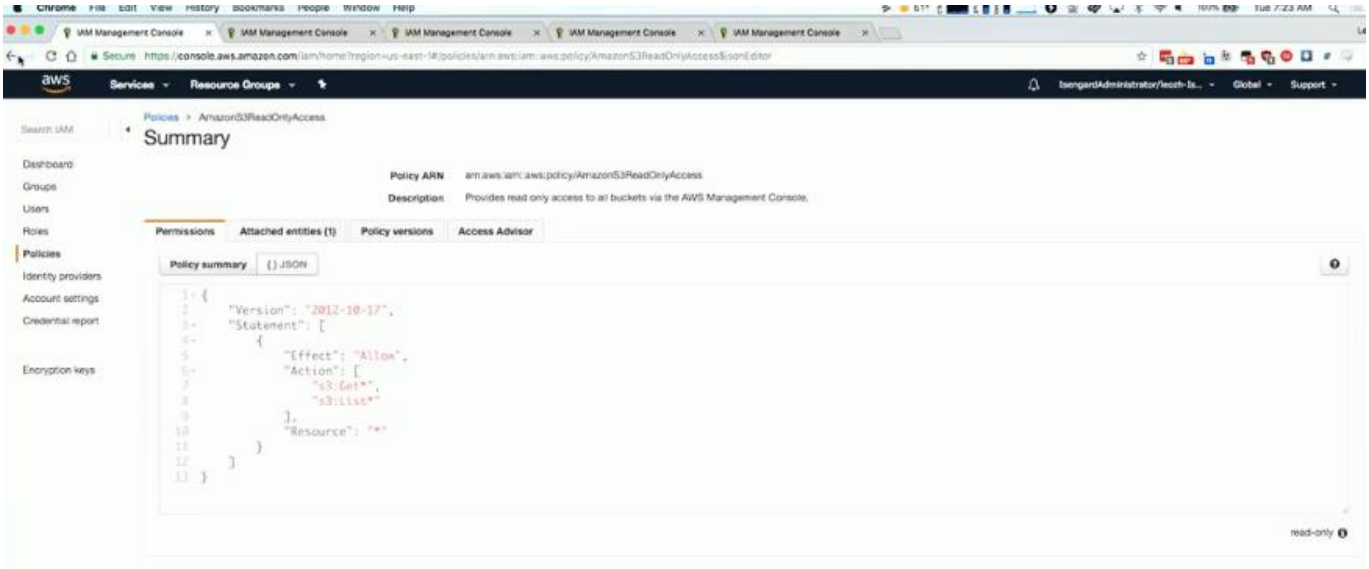
We SSH into the EC2 Bastion instance using the instance's IP address in the command ***$ ssh ec2-user@54.161.244.145***



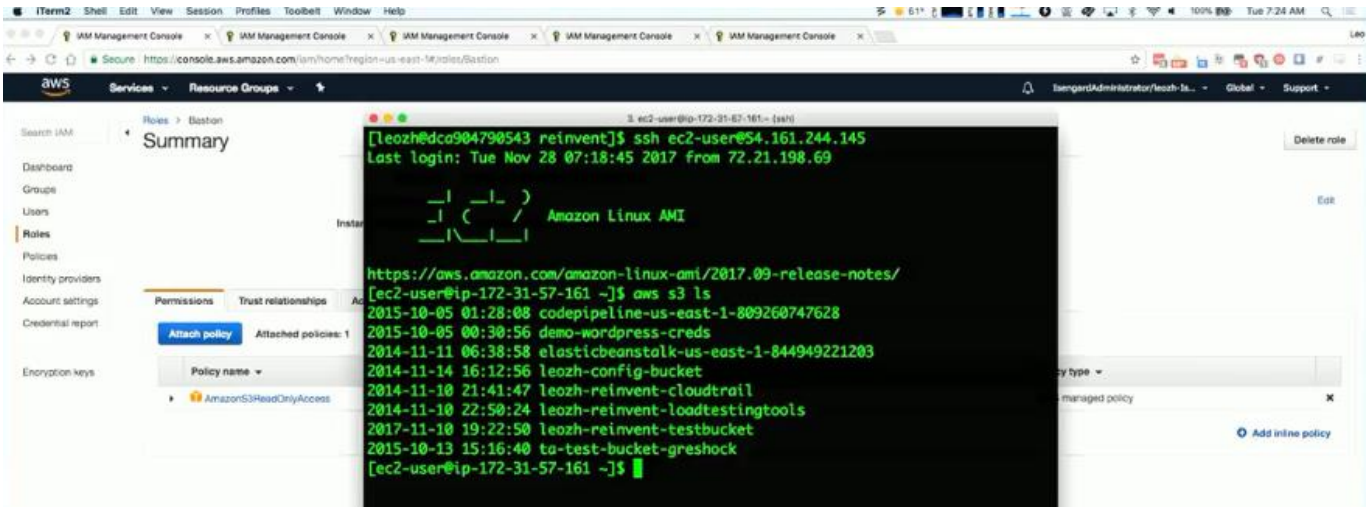
For this EC2 instance, we have given it the Bastion IAM role, let us see this Bastion IAM role below



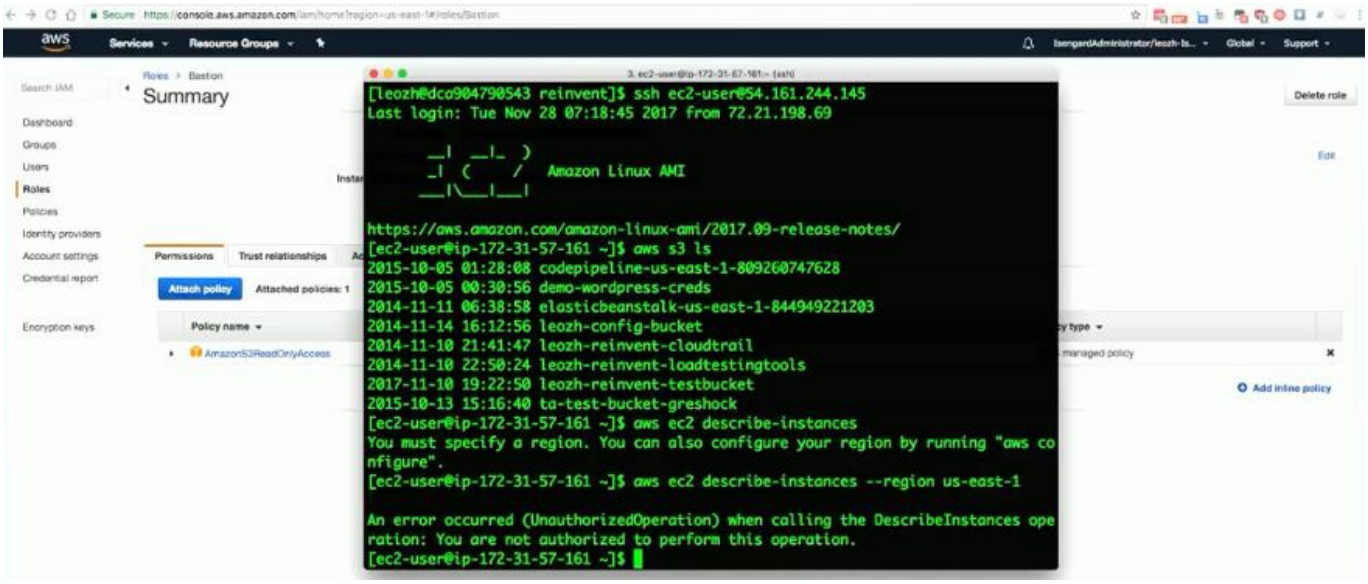
The Bastion IAM role currently has 1 policy attached to it, ***AmazonS3ReadOnlyAccess*** policy shown below



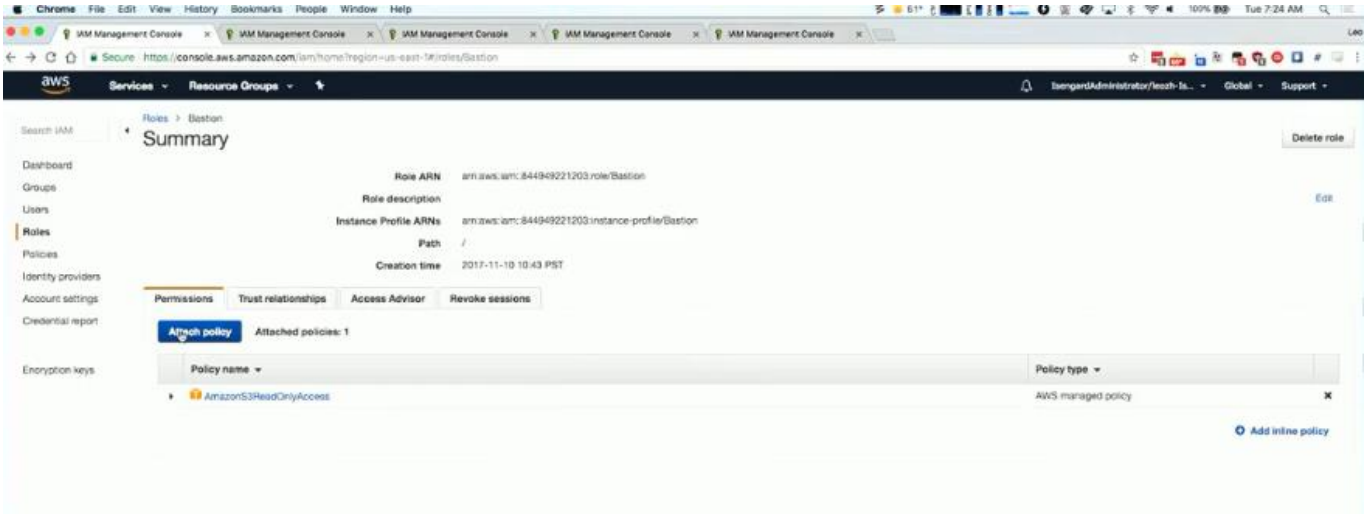
This allows us to read S3 buckets only from the Ec2 instance



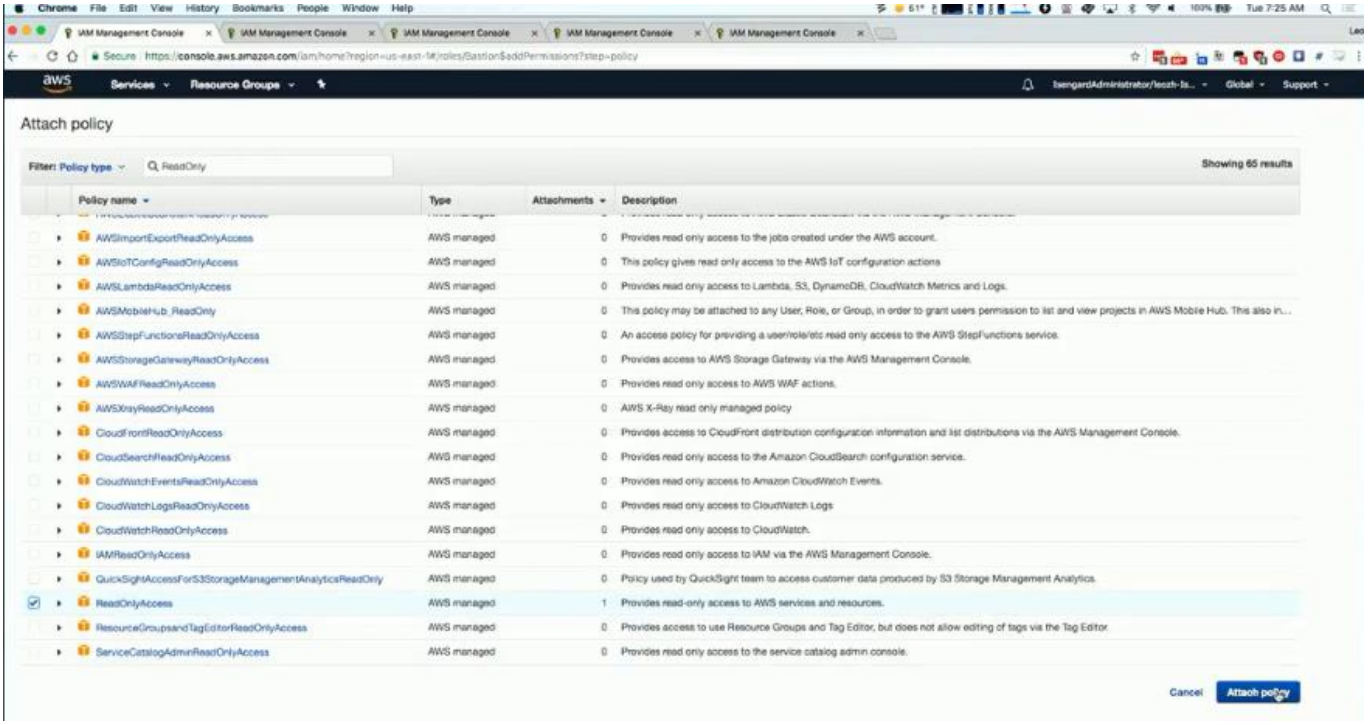
We can use the ***$ aws s3 ls*** command to list our available S3 buckets above

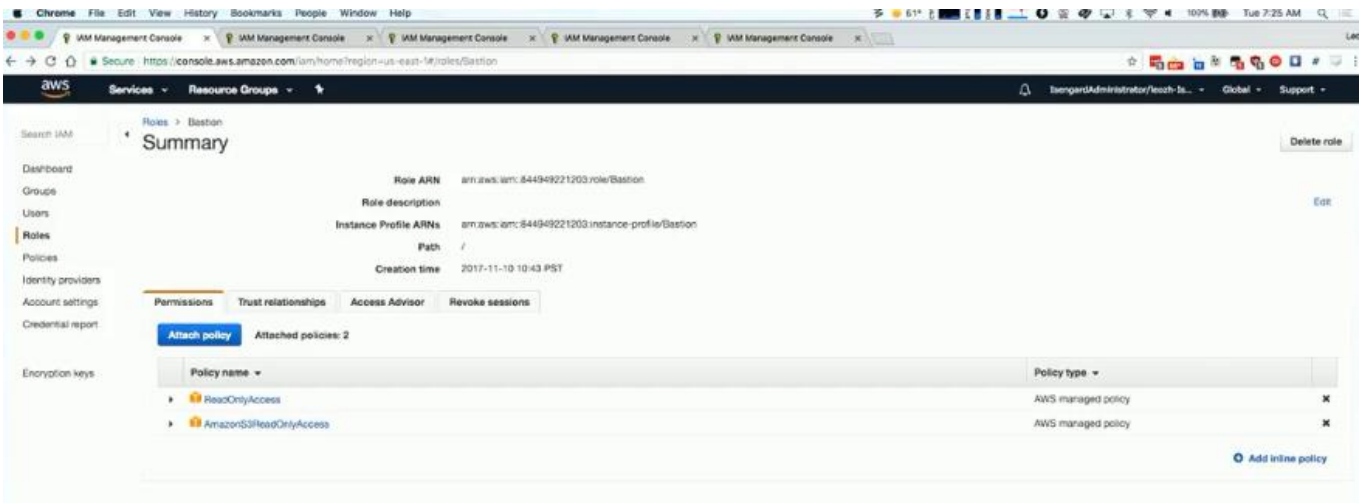


When we use the command ***$ aws ec2 describe-instances - -region us-east-1*** to get details from the EC2 service, we see that we can't do that because of the Bastion host's IAM role is for S3 read-only access.



We can now add a Read-only policy to the Bastion instance role to allow





The ***ReadOnlyAccess*** policy will give the EC2 instance read-only access to all services in this account

```
[leozh@dca904790543 reinvent]$ ssh ec2-user@54.161.244.145
Last login: Tue Nov 28 07:18:45 2017 from 72.21.198.69

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
[ec2-user@ip-172-31-57-161 ~]$ aws s3 ls
2015-10-05 01:28:08 codepipeline-us-east-1-809260747628
2015-10-05 00:30:56 demo-wordpress-creds
2014-11-11 06:38:58 elasticbeanstalk-us-east-1-844949221203
2014-11-14 16:12:56 leozh-config-bucket
2014-11-10 21:41:47 leozh-reinvent-cloudtrail
2014-11-10 22:50:24 leozh-reinvent-loadtestingtools
2017-11-10 19:22:50 leozh-reinvent-testbucket
2015-10-13 15:16:40 ta-test-bucket-greshock
[ec2-user@ip-172-31-57-161 ~]$ aws ec2 describe-instances
You must specify a region. You can also configure your region by running "aws co
nfigure".
[ec2-user@ip-172-31-57-161 ~]$ aws ec2 describe-instances --region us-east-1

An error occurred (UnauthorizedOperation) when calling the DescribeInstances ope
ration: You are not authorized to perform this operation.
[ec2-user@ip-172-31-57-161 ~]$ aws ec2 describe-instances --region us-east-1
```

We then run the previous command ***$ aws ec2 describe-instances - -region us-east-1*** to get details from the EC2 service again and it should work as below

```
                "RootDeviceType": "ebs",
                "IamInstanceProfile": {
                    "Id": "AIPAJ6KAMB3K7M5IZDA46",
                    "Arn": "arn:aws:iam::844949221203:instance-profile/demo-
wordpress-iam-role"
                },
                "RootDeviceName": "/dev/xvda",
                "VirtualizationType": "hvm",
                "Tags": [
                    {
                        "Value": "demo-wordpress",
                        "Key": "aws:autoscaling:groupName"
                    }
                ],
                "AmiLaunchIndex": 0
            }
        ],
        "ReservationId": "r-05cb575d5e8f788de",
        "RequesterId": "226008221399",
        "Groups": [],
        "OwnerId": "844949221203"
    }
]
}
[ec2-user@ip-172-31-57-161 ~]$
```

You can then exit the SSH EC2 instance using the ***exit*** command

MFA

- Require MFA for role assumption:

```
{
 "Version": "2012-10-17",
 "Statement": [
   {
     "Sid": "",
     "Effect": "Allow",
     "Principal": { "AWS": "arn:aws:iam::179442201234:user/leozh" },
     "Action": "sts:AssumeRole",
     "Condition": { "Bool": { "aws:MultiFactorAuthPresent": true } }
   }
 ]
}
```

- ~/.aws/config

```
[profile billing]
role_arn = arn:aws:iam::179442201234:role/billing
source_profile = default
mfa_serial = arn:aws:iam::179442201234:mfa/leozh
```

Where do I find my MFA ARN?

Sign-in credentials

Console password	Enabled Manage password
Console login link	https://[redacted]signin.aws.amazon[redacted]m/console
Last login	2017-11-06 19:15 EST
Assigned MFA device	arn:aws:iam:[redacted]:mfa/leozh
Signing certificates	None

or

```
$ aws iam list-mfa-devices --user-name leozh
{
    "MFADevices": [
        {
            "UserName": "leozh",
            "SerialNumber": "arn:aws:iam::179442201234:mfa/leozh",
            "EnableDate": "2016-09-15T00:27:33Z"
        }
    ]
}
```

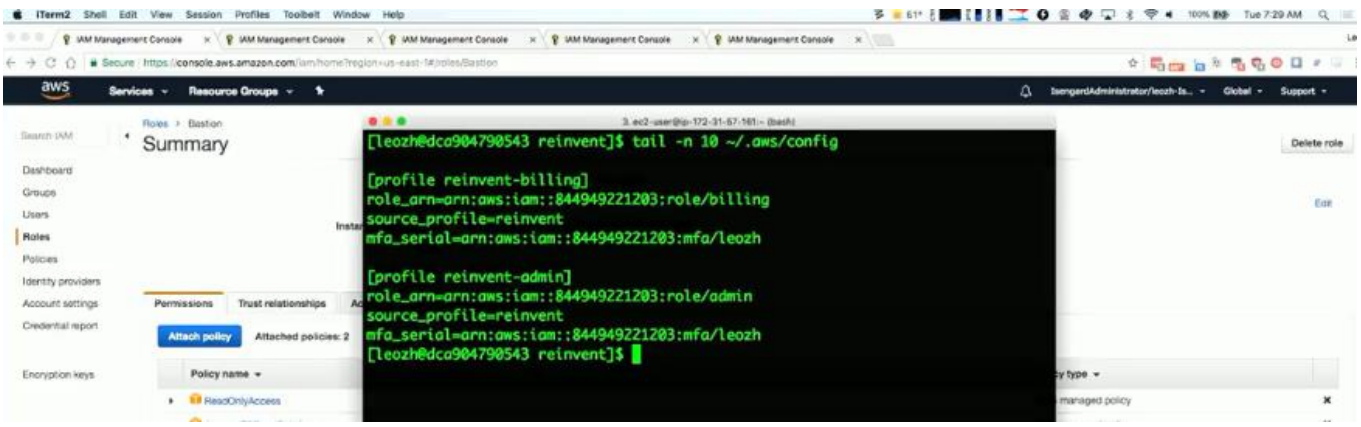
AWS re:Invent

aws

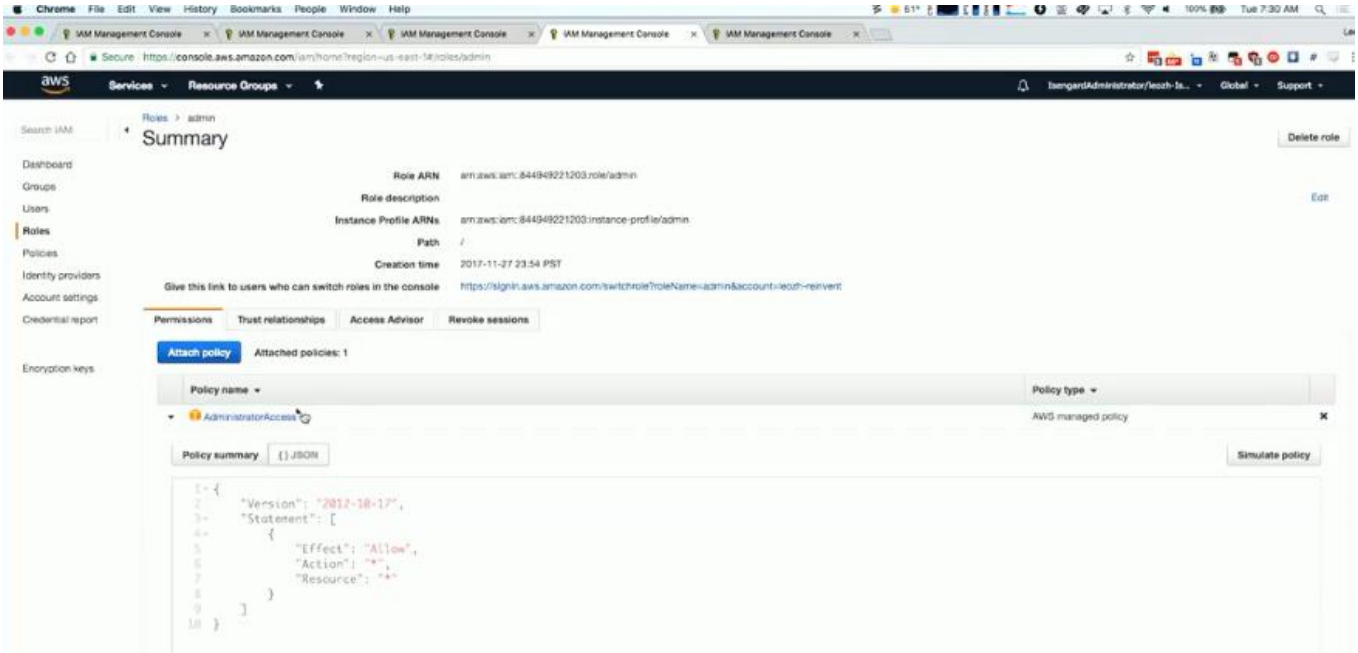
You can enable MFA through the CLI, and set up a policy for your trust relationship. Above is assuming a billing role, you add the serial number of your MFA into the billing profile as above. You can get the ***mfa_serial*** value as above in one of 2 ways using the web console or the CLI.

MFA

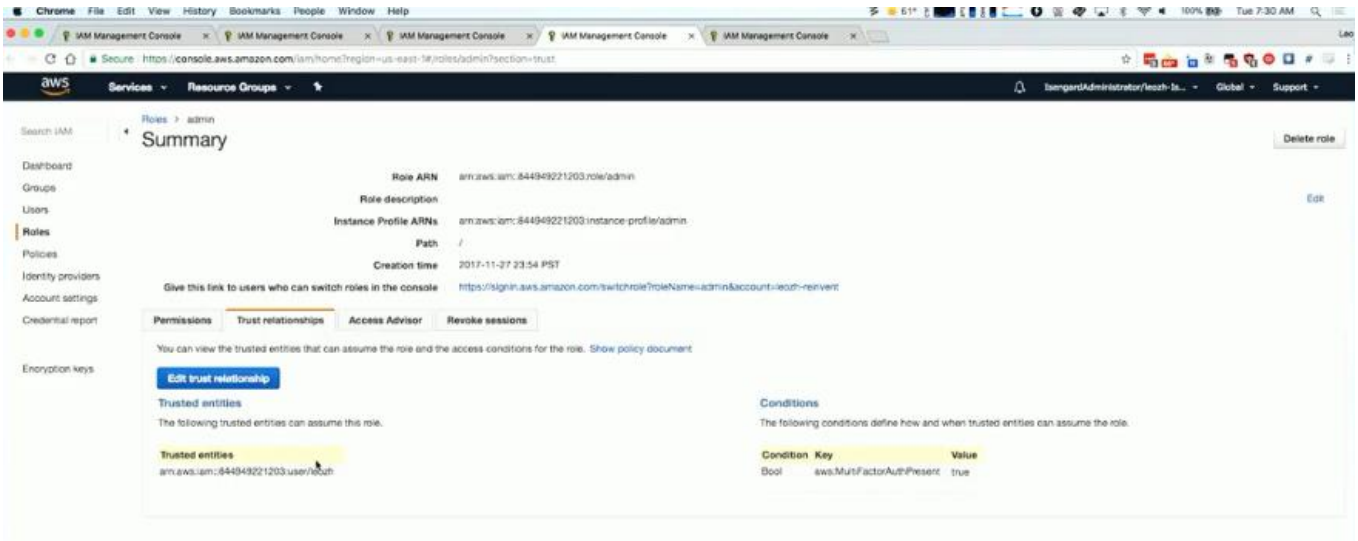
Demo



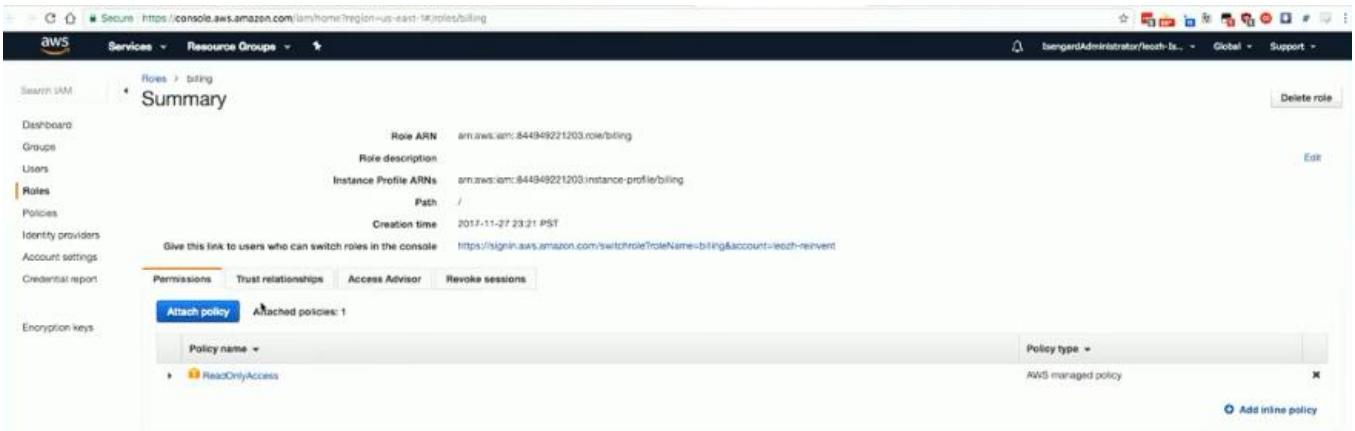
Let us tail our config file using the ***$ tail -n 10 ~/.aws/config*** command as above



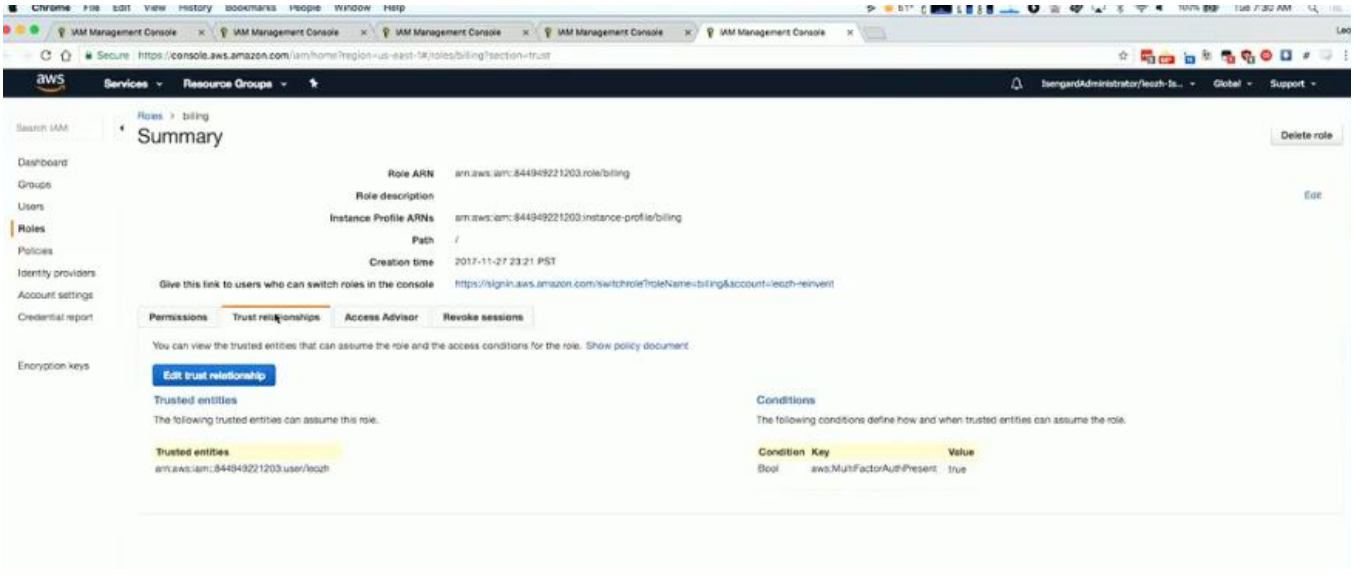
We have 2 roles, billing and admin. The billing role has read-only access but the admin role can do anything within the account. Above is our Admin role with the ***AdministratorAcess*** policy attached that gives access to everything.

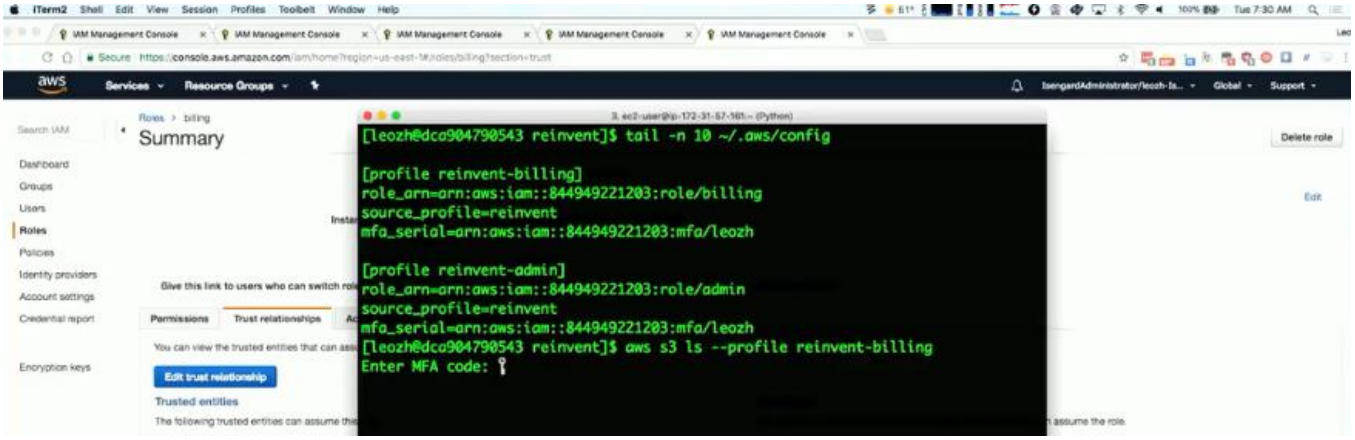


The Admin role has a trust relationship that is trusting my IAM user ***arn:aws:iam:335736757:user/leozh*** to be able assume this Admin role with the ***Bool*** condition for MFA to be present set to ***true***

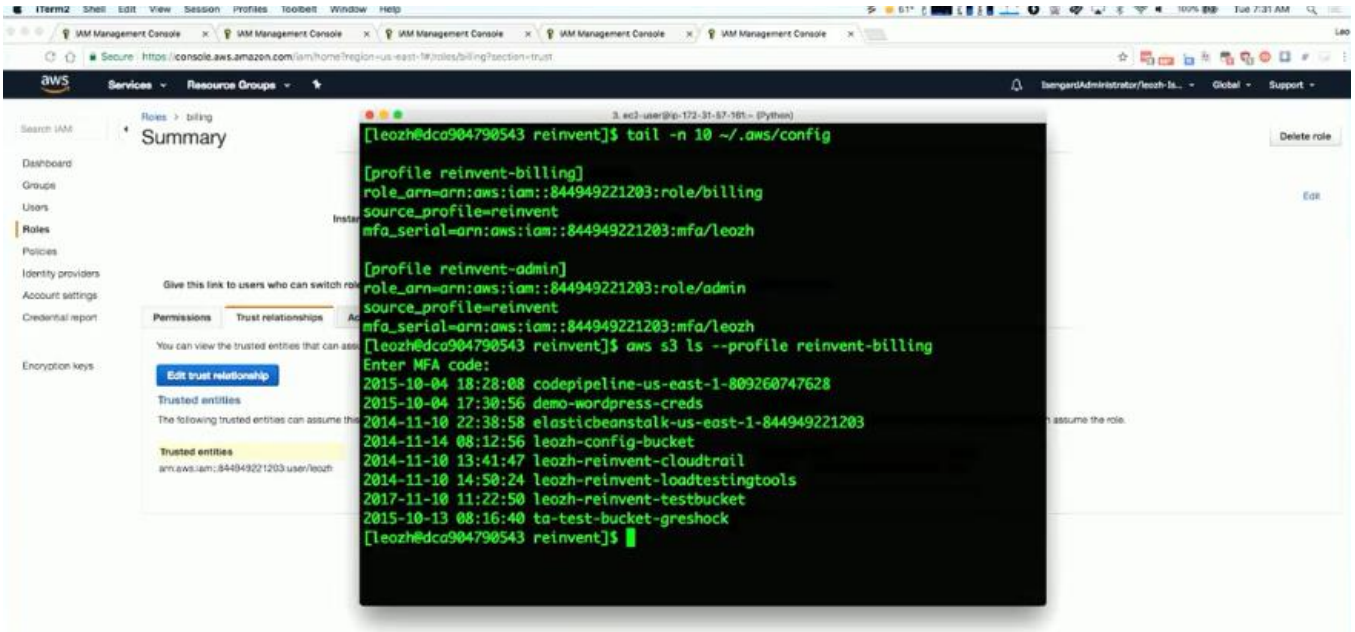


Then we have the billing role with read-only access with the same exact trust relationship for our user leozh as below

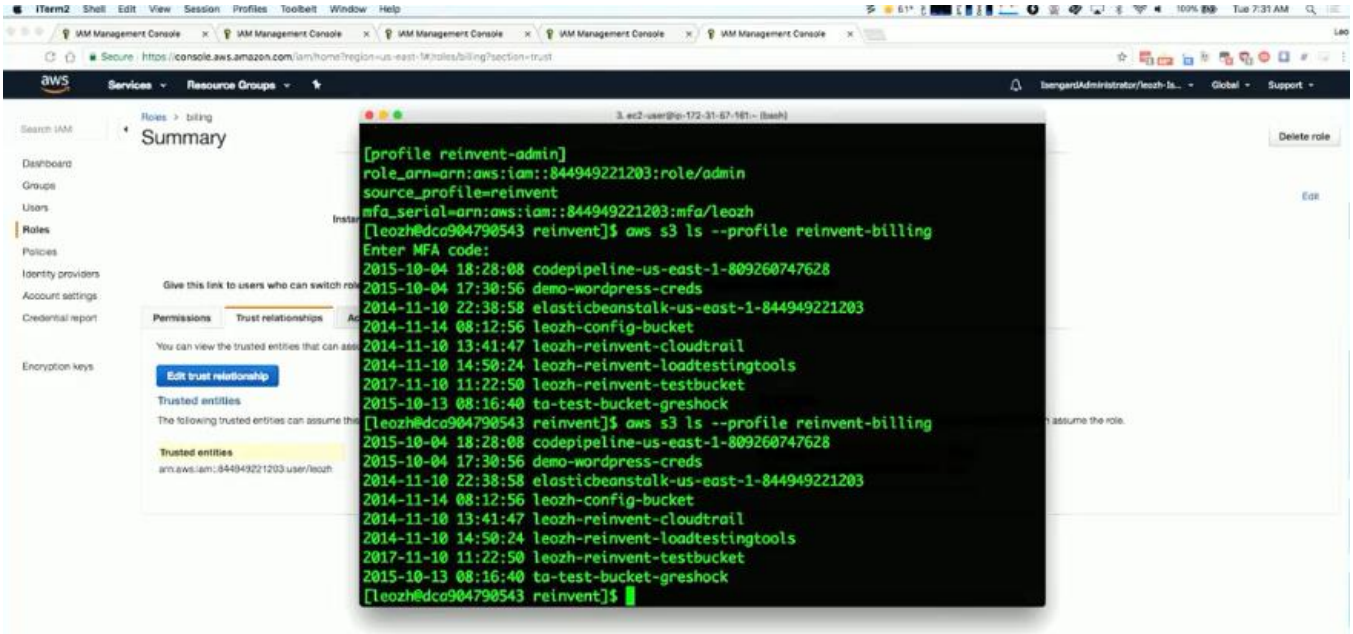




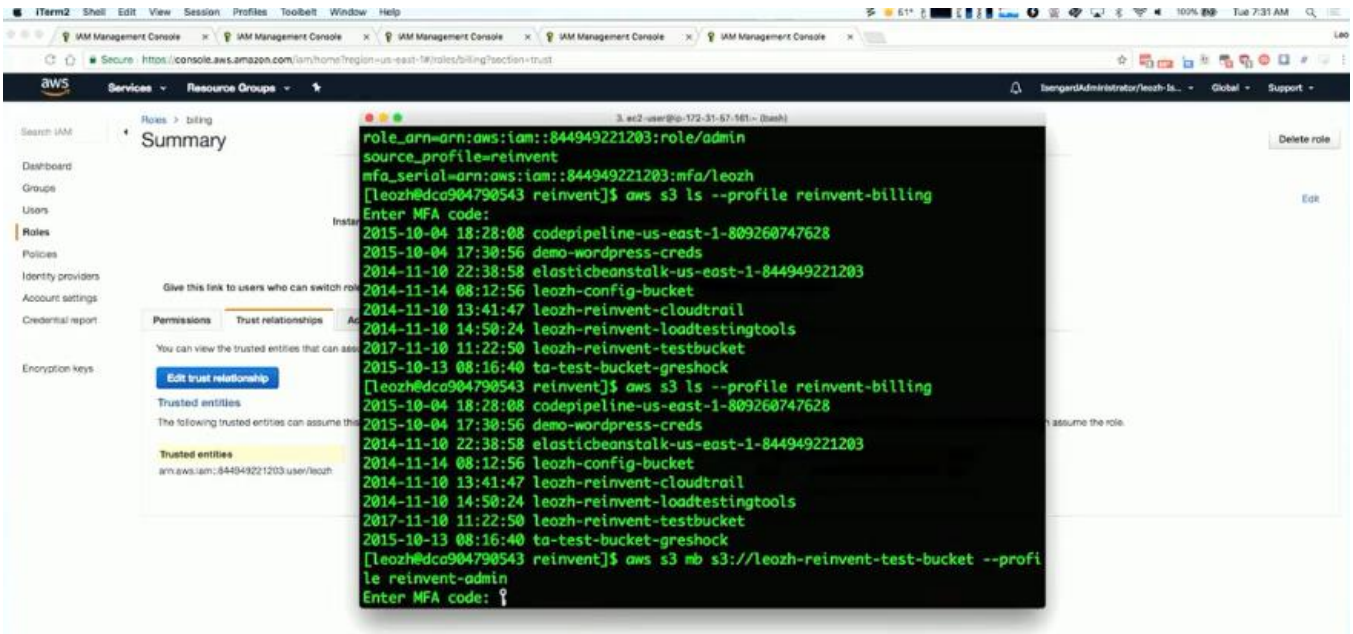
From the CLI, we can now try to assume the billing role and list all our S3 files using the command ***$ aws s3 ls –profile reinvent-billing***, it immediately asks us for the MFA code which we can get from the phone if using a software token



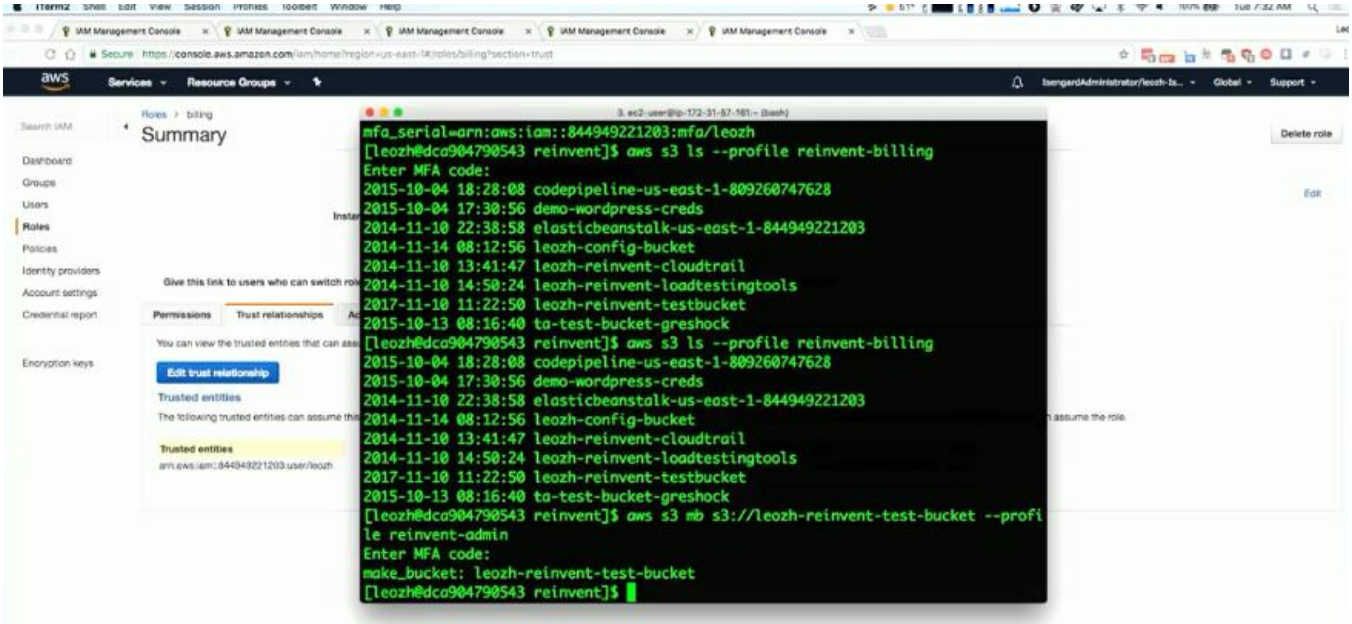
We enter the correct MFA code and the CLI can execute our command to list the S3 buckets as above.



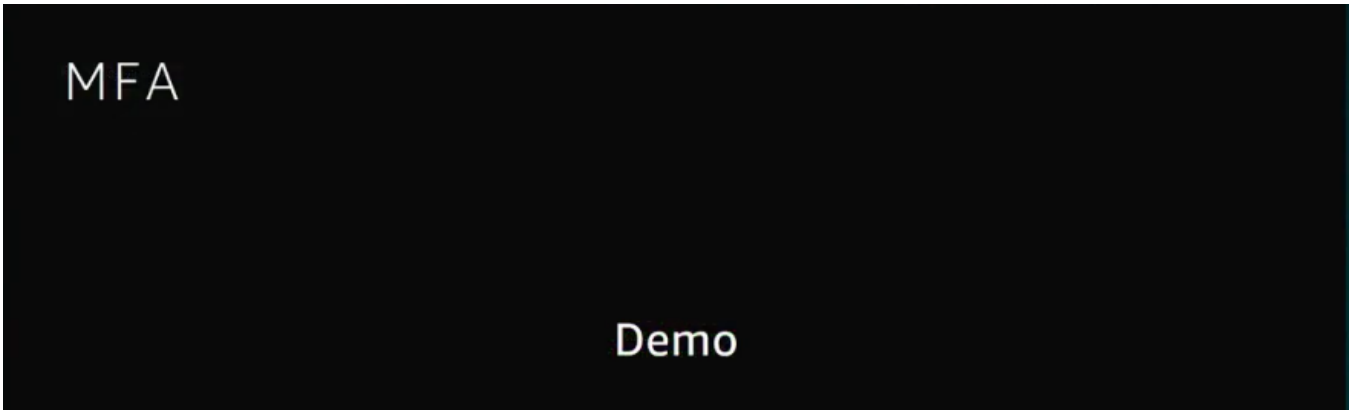
This MFA code validation remains active for a while and we don't need to put it in until it expires.

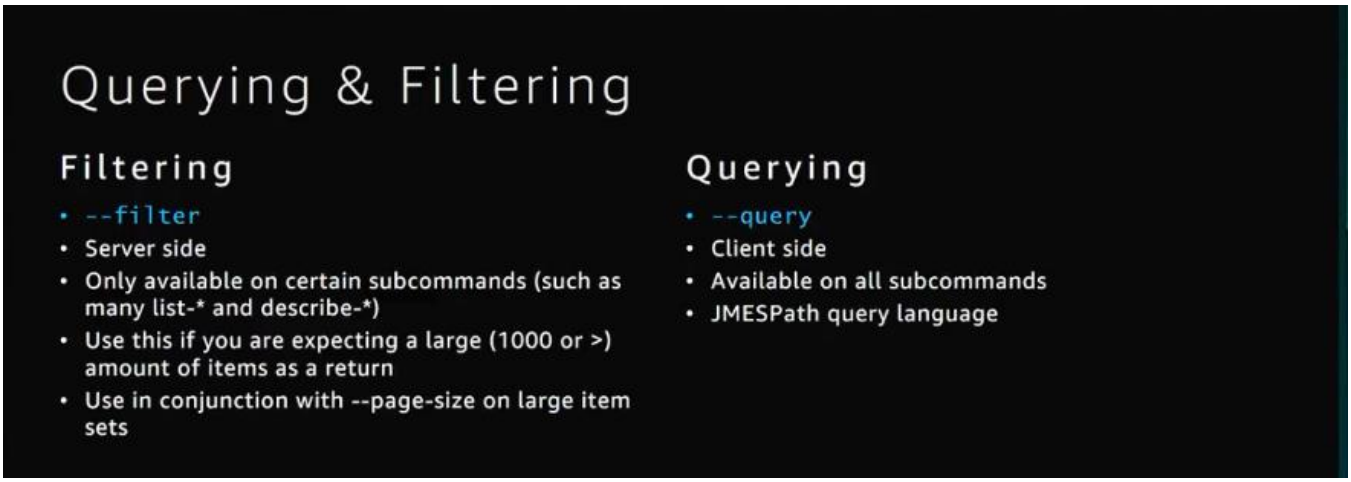


We can also switch role to the admin role and try to create a new S3 bucket using the command ***$aws s3 mb s3://leozh-reinvent-test-bucket –profile reinvent-admin***, we again have to enter the MFA code because we are switching profiles as above



We enter the MFA code and the bucket gets created as above





We can also do ***querying*** and ***filtering*** from the CLI to reduce queries that return a lot of results like listing the files in an S3 bucket.

Filtering

```
aws ec2 describe-instances --filter Name=instance-type,Values=t2.micro
{
   "Reservations": [
       {
           "Instances": [
               {
                   "Monitoring": {
                       "State": "disabled"
                   },
                   "PublicDnsName": "ec2-34-228-84-177.compute-1.amazonaws.com",
                   "State": {
                       "Code": 16,
                       "Name": "running"
                   },
                   "EbsOptimized": false,
                   "LaunchTime": "2017-07-18T23:14:07.000Z",
…
…
           },
```

We can get a list of our EC2 instances filtered by instance type of t2.micro using the command ***$ aws ec2 describe-instances - -filter Name=instance-type,Values=t2.micro*** as above.

CLI Aliases

https://github.com/awslabs/awscli-aliases

```
$ aws whoami
{
   "Account": "179442201234",
   "UserId": "AIDAIXV2V6G4AEXAMPLE",
   "Arn": "arn:aws:iam::179442201234:user/leozh"
}

$ aws amazon-linux-amis
ami-6057e21a amzn-ami-hvm-2017.09.1.20171103-x86_64-gp2 Amazon Linux AMI 2017.09.1.20171103 x86_64 HVM GP2
ami-8c1be5f6 amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 Amazon Linux AMI 2017.09.0.20170930 x86_64 HVM GP2
ami-5e8c9625 amzn-ami-hvm-2017.09.rc-0.20170913-x86_64-gp2 Amazon Linux AMI 2017.09.rc-0.20170913 x86_64 HVM
GP2
ami-4fffc834 amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2 Amazon Linux AMI 2017.03.1.20170812 x86_64 HVM GP2
…
```

You can also set up CLI Aliases using the above plugin that in the background would run more complicated commands instead of you having to remember them every time. It comes with a set of predefined aliases and also allows you to set up custom ones

Querying

```
$ aws iam list-users
{
   "Users": [
       {
           "UserName": "leozh",
           "PasswordLastUsed": "2017-11-07T00:15:03Z",
           "CreateDate": "2013-08-04T17:24:01Z",
           "UserId": "AIDAIXV2V6G4AAAV5UXYZ",
           "Path": "/",
           "Arn": "arn:aws:iam::179442201234:user/leozh"
       },
       {
           "UserName": "billing",
           "PasswordLastUsed": "2015-03-17T18:47:44Z",
           "CreateDate": "2015-03-17T03:31:45Z",
           "UserId": "AIDAIFVT4OFCCN6B5QABC",
           "Path": "/",
           "Arn": "arn:aws:iam::1794422041234:user/billing"
       }
```

AWS re:Invent

aws

We can list all our IAM users using the command ***$ aws iam list-users*** as above.

Querying

```
$ aws iam list-users --query 'Users[*].[UserName, CreateDate, PasswordLastUsed, Arn]'
[
   [
       "leozh",
       "2013-08-04T17:24:01Z",
       "2017-11-07T00:15:03Z",
       "arn:aws:iam::179442201234:user/leozh"
   ],
   [
       "Billing",
       "2015-03-17T03:31:45Z",
       "2015-03-17T18:47:44Z",
       "arn:aws:iam::179442201234:user/billing"
   ]
]
```

We can use the JMESPath query language to query for specific IAM users using the command ***$ aws iam list-users - -query 'Users[*].[Username, CreateDate, PasswordLastUsed, Arn]'*** as above

Querying

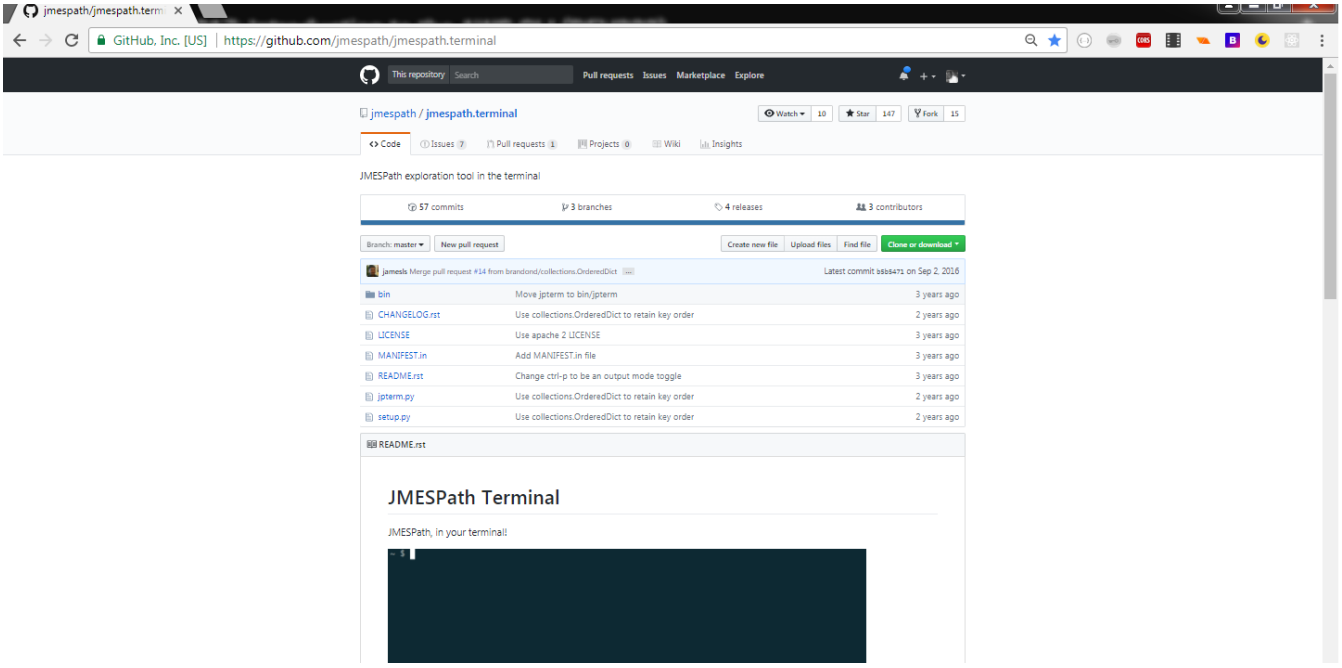
```
$ aws ec2 describe-instances --query Reservations[*].Instances[*].State.Name
[
    [
        "stopped"
    ],
    [
        "running"
    ],
    [
        "running"
    ]
]
```

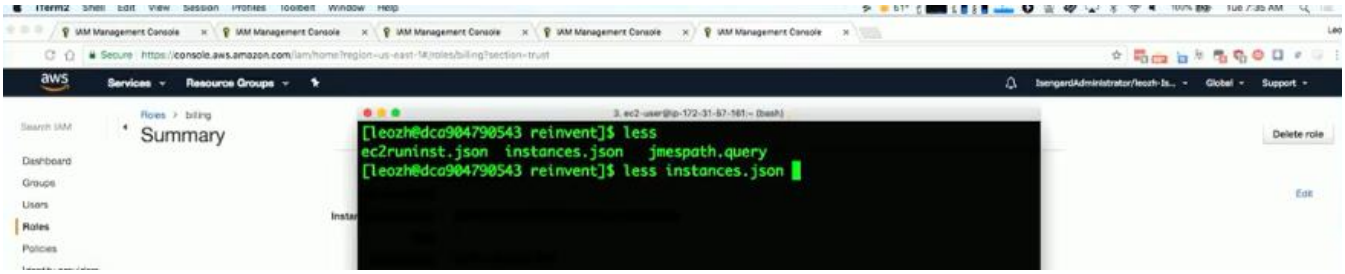
We can also list all our EC2 instances with the state they are running in using the command ***$ aws ec2 decsribe-instances - -query Reservations[*].Instances[*].State.Name*** as above

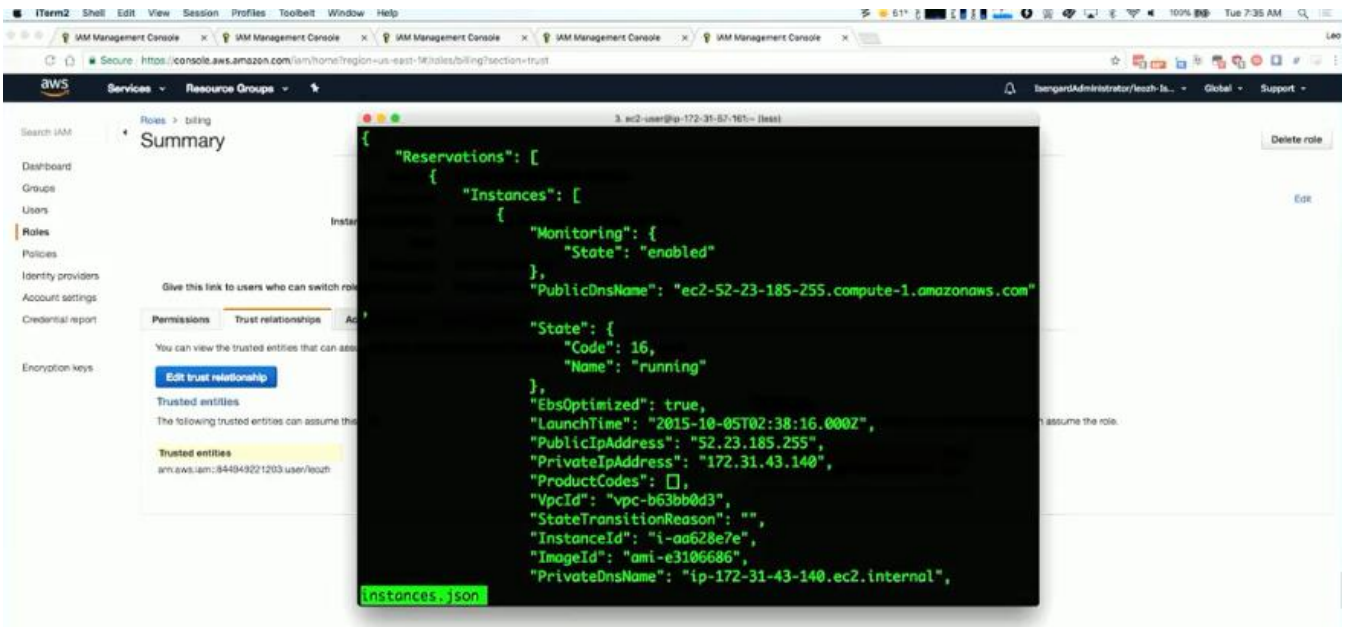
JMESPath Terminal

Demo

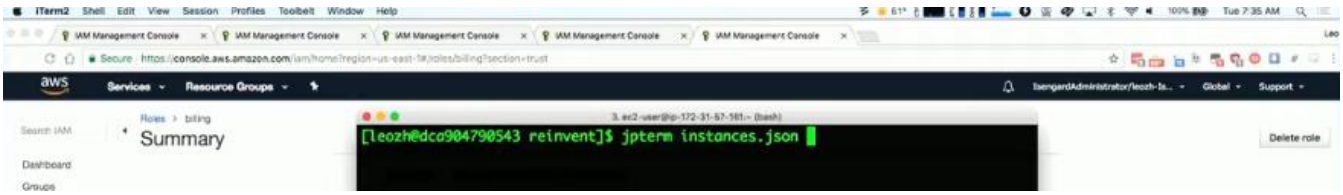
JMESPath Terminal allows you to test out the commands before issuing them in your AWS CLI



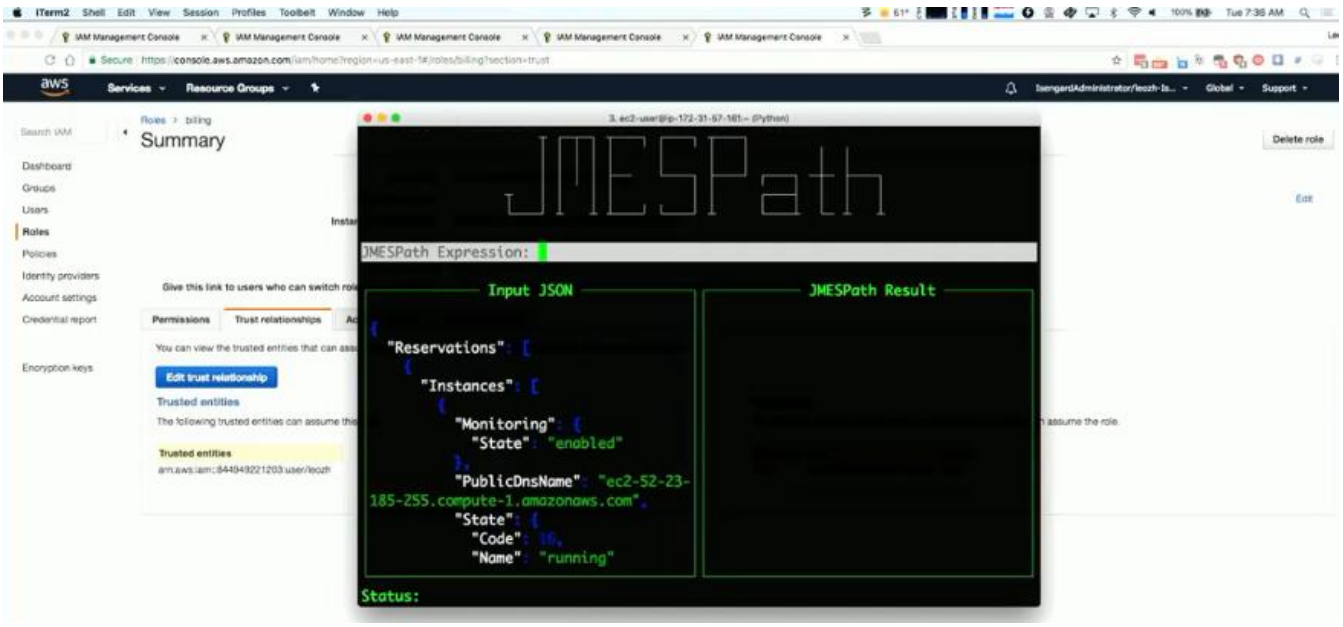


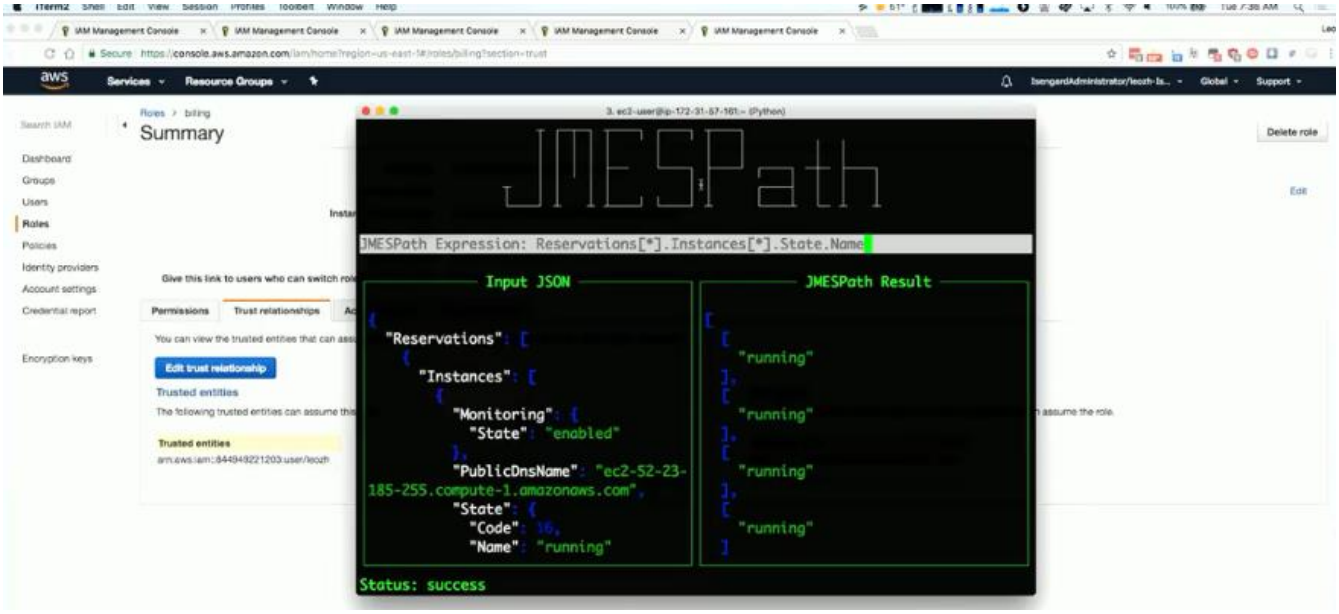


The way to use JMESPath is to do a describe-instances command and exported the output to a file on disk called instances.json.

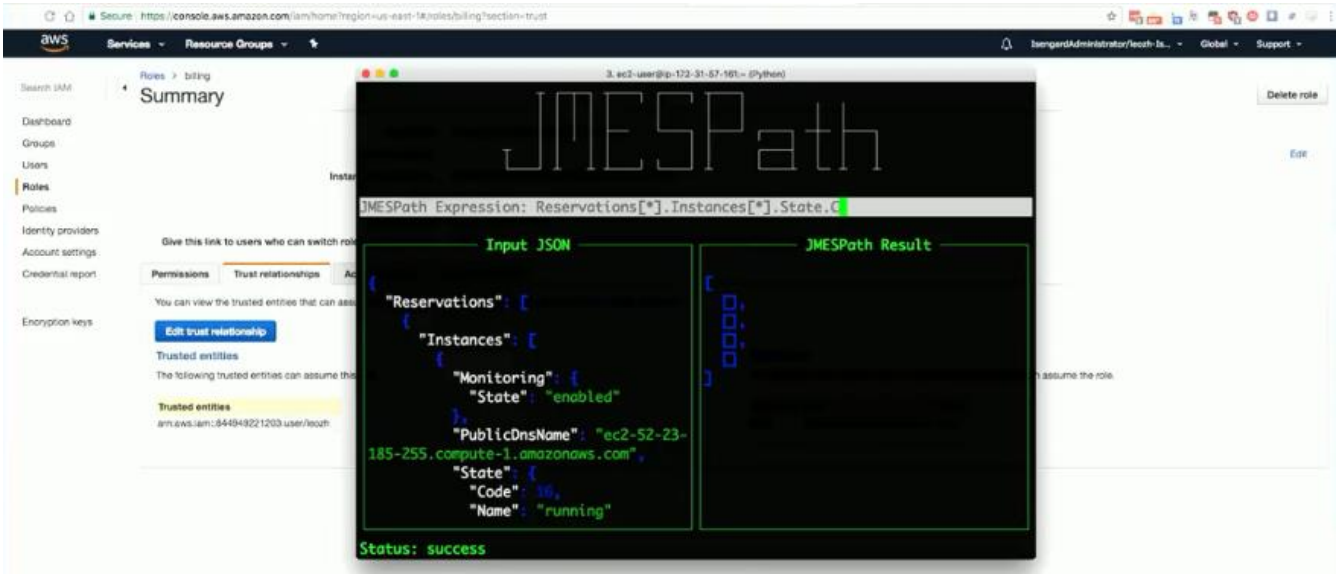


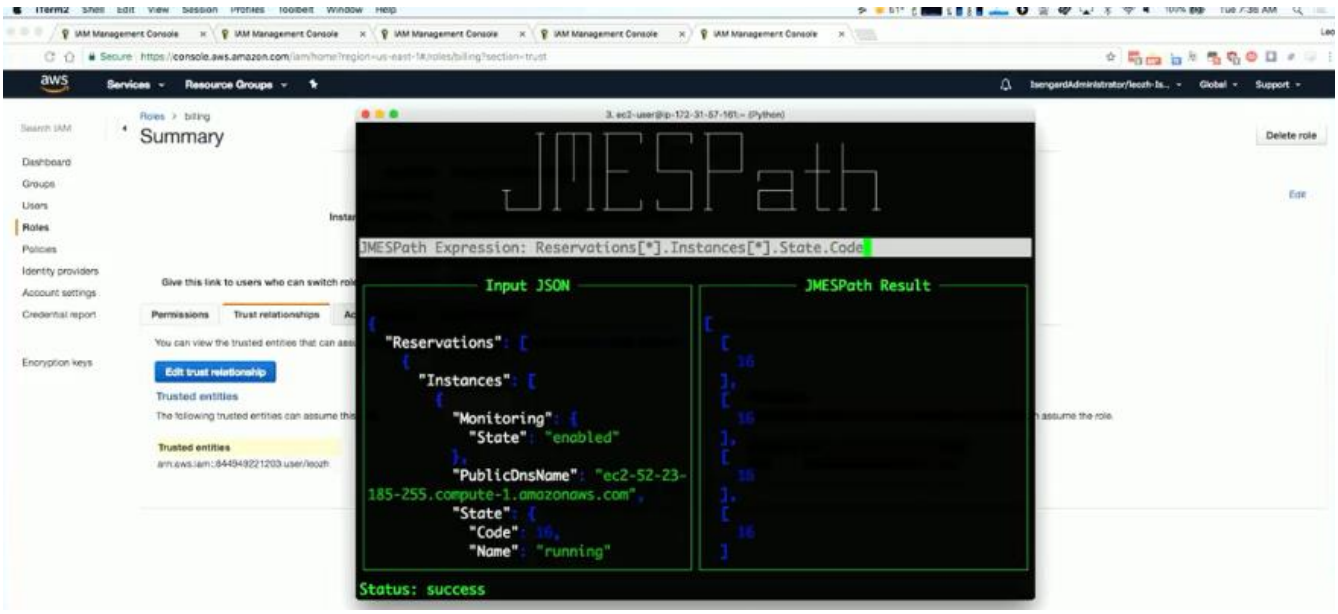
We then can run the JMESPath using the command ***$ jpterm instances.json*** as above

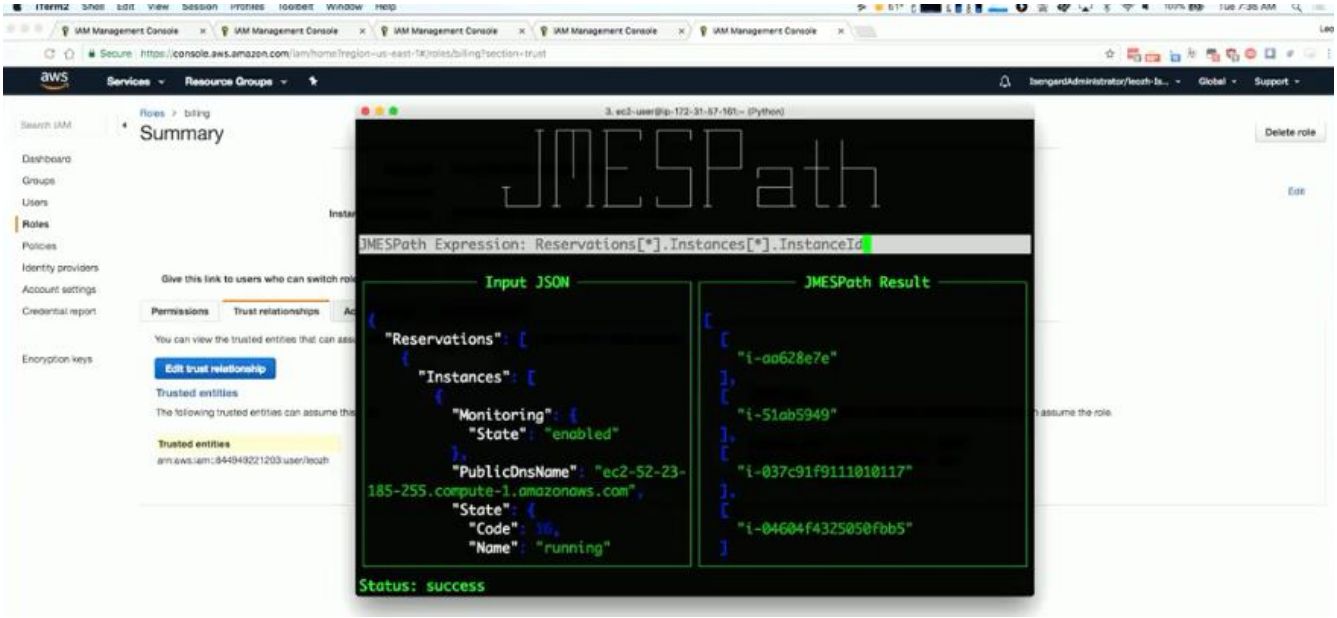




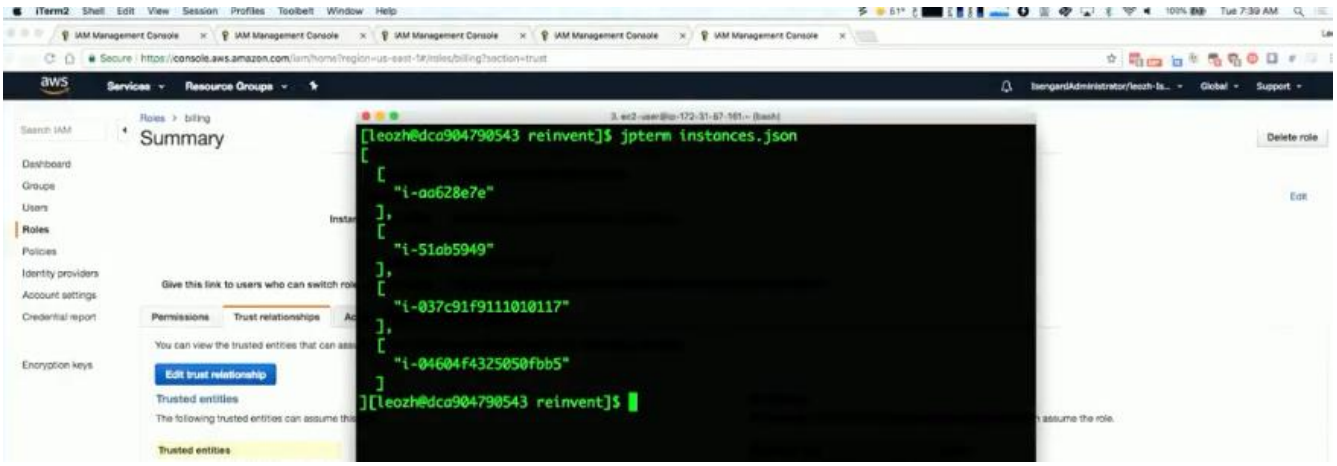
We can now create queries by typing it into the terminal as above







This is an easy way to play around with JMESPath



Querying

JMESPath

Learn more here: http://jmespath.org/tutorial.html

JMESPath Terminal

https://github.com/jmespath/jmespath.terminal

Generate CLI Skeleton

```
$ aws ec2 run-instances --generate-cli-skeleton > ec2runinst.json
{
    "DryRun": true,
    "ImageId": "",
    "MinCount": 0,
    "MaxCount": 0,
    "KeyName": "",
    "SecurityGroups": [
        ""
    ],
    "SecurityGroupIds": [
        ""
    ],
    "UserData": "",
    "InstanceType": "",
...
}
```

We can also write EC2 commands that require input values, like when we want to run an EC2 instance, we need to specify what AMI we want to use ?, what security group that EC2 instance should belong to ?, what instance type to use ?, etc. if we want to run such script without having to manually put in the needed parameters, we can ***instead export the skeleton of the command into a file*** like the command ***$ aws ec2 run-instances - -generate-cli-skeleton > ec2runinst.json*** as above

Generate CLI Skeleton

edit ec2runinst.json

```
{
    "DryRun": false,
    "ImageId": "ami-dfc39aef",
    "KeyName": "mykey",
    "SecurityGroups": [
        "my-sg"
    ],
    "InstanceType": "t2.micro",
    "Monitoring": {
        "Enabled": true
    }
}
```

We can then edit the skeleton file created for us and delete all the lines we don't want in the command we want to use as above.

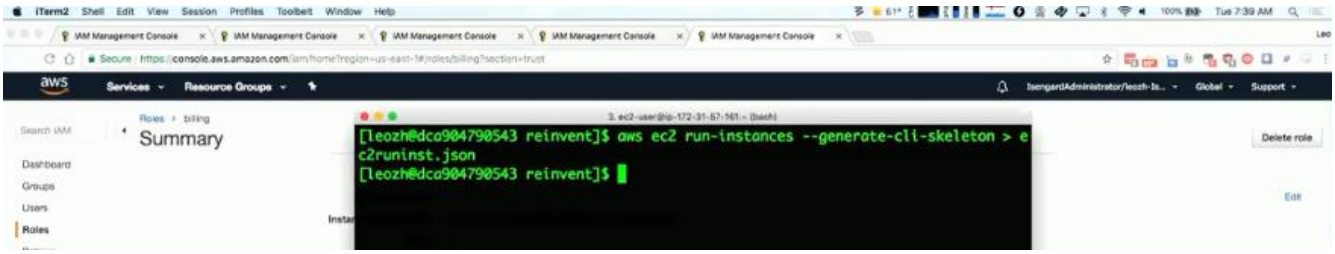
Generate CLI Skeleton

```
$ aws ec2 run-instances --cli-input-json file://ec2runinst.json
{
    "OwnerId": "123456789012",
    "ReservationId": "r-d94a2b1",
    "Groups": [],
    "Instances": [
...
```

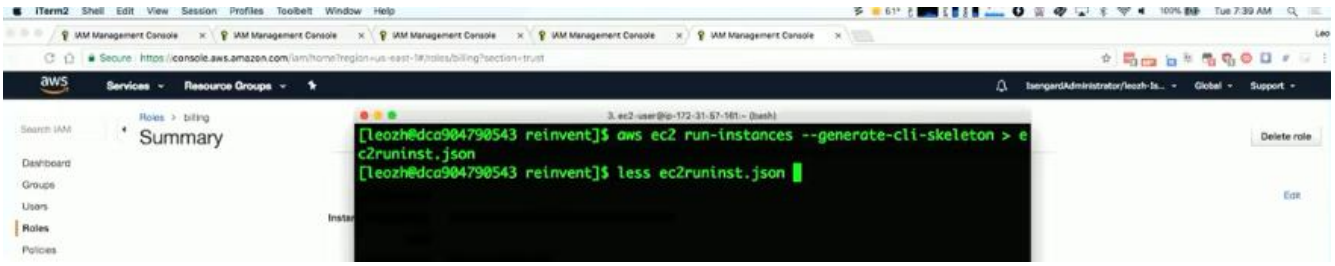
You can then issue the command with the path to the edited file using the command ***$ aws ec2 run-instances - -cli-input-json file://ec2runinst.json*** as above

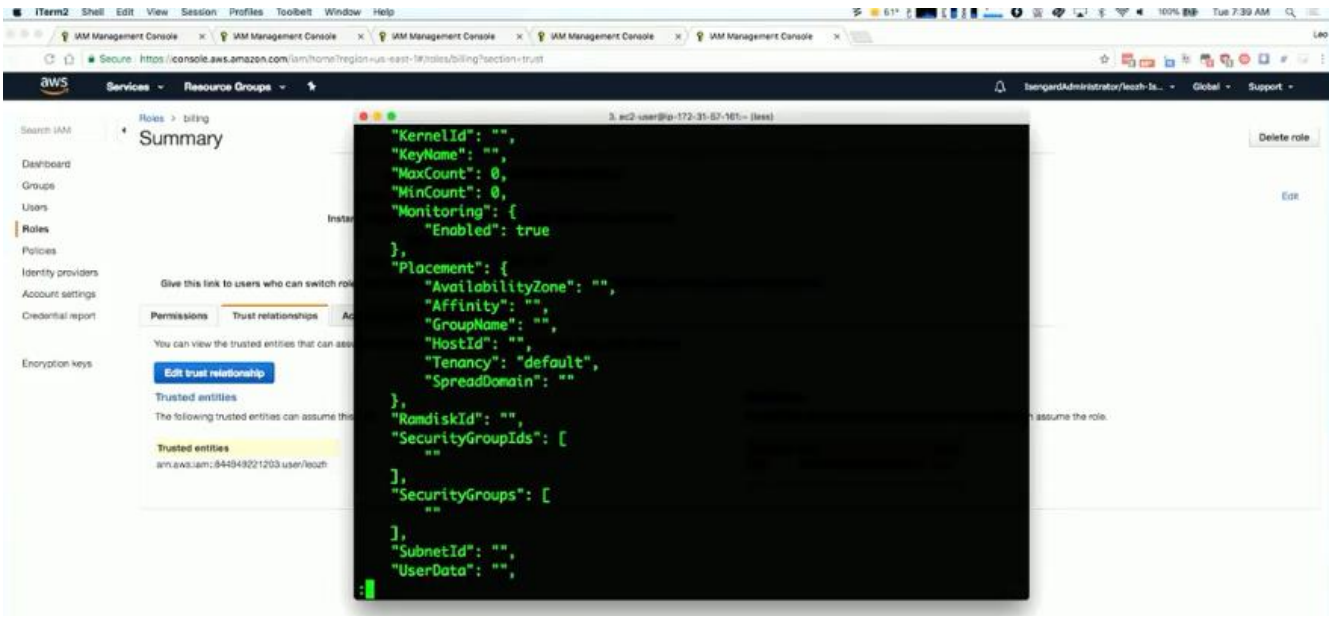
Generate CLI Skeleton

Demo

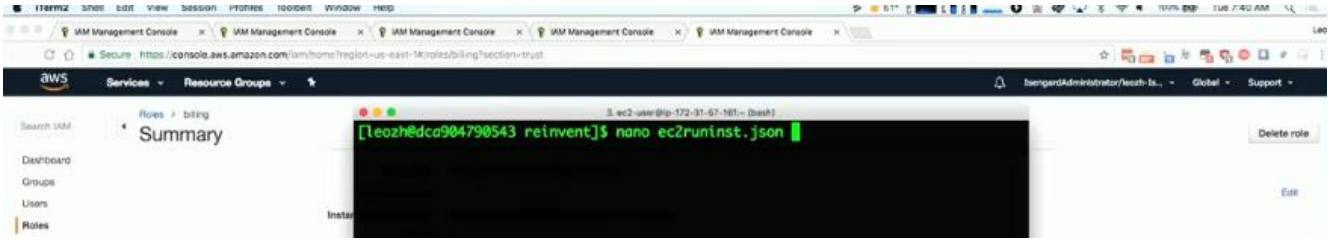


Use the ***$ aws ec2 run-instances - -generate-cli-skeleton > ec2runinst.json*** command to generate the CLI skeleton

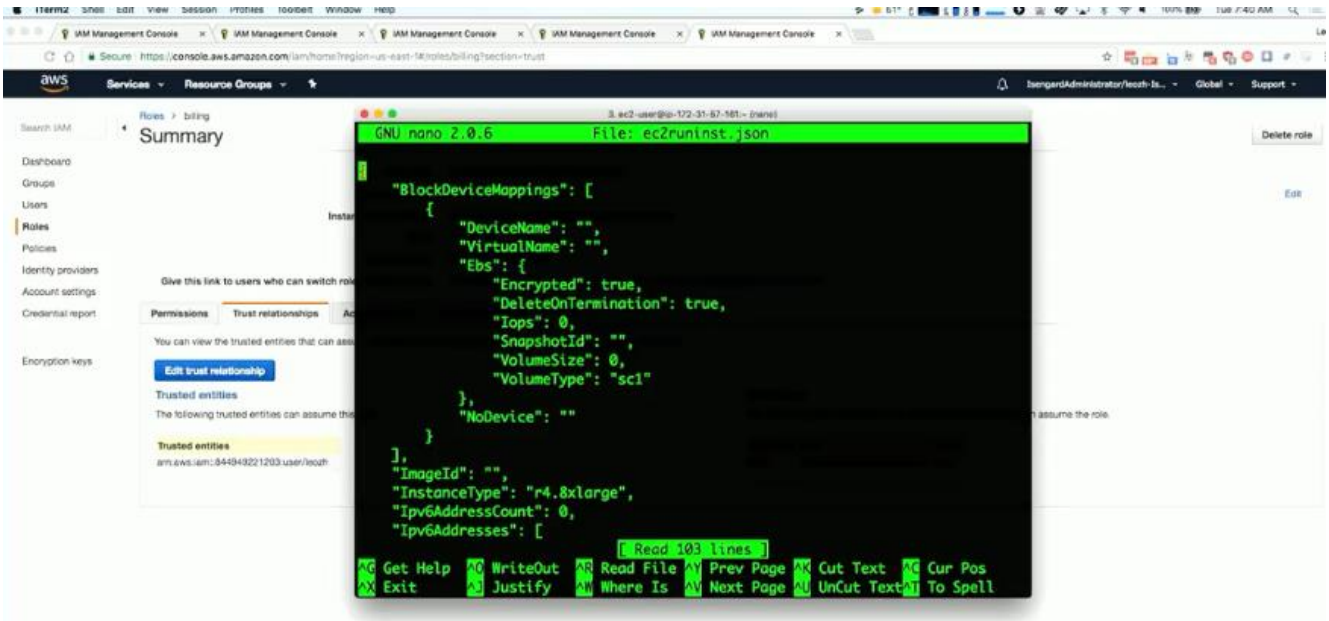


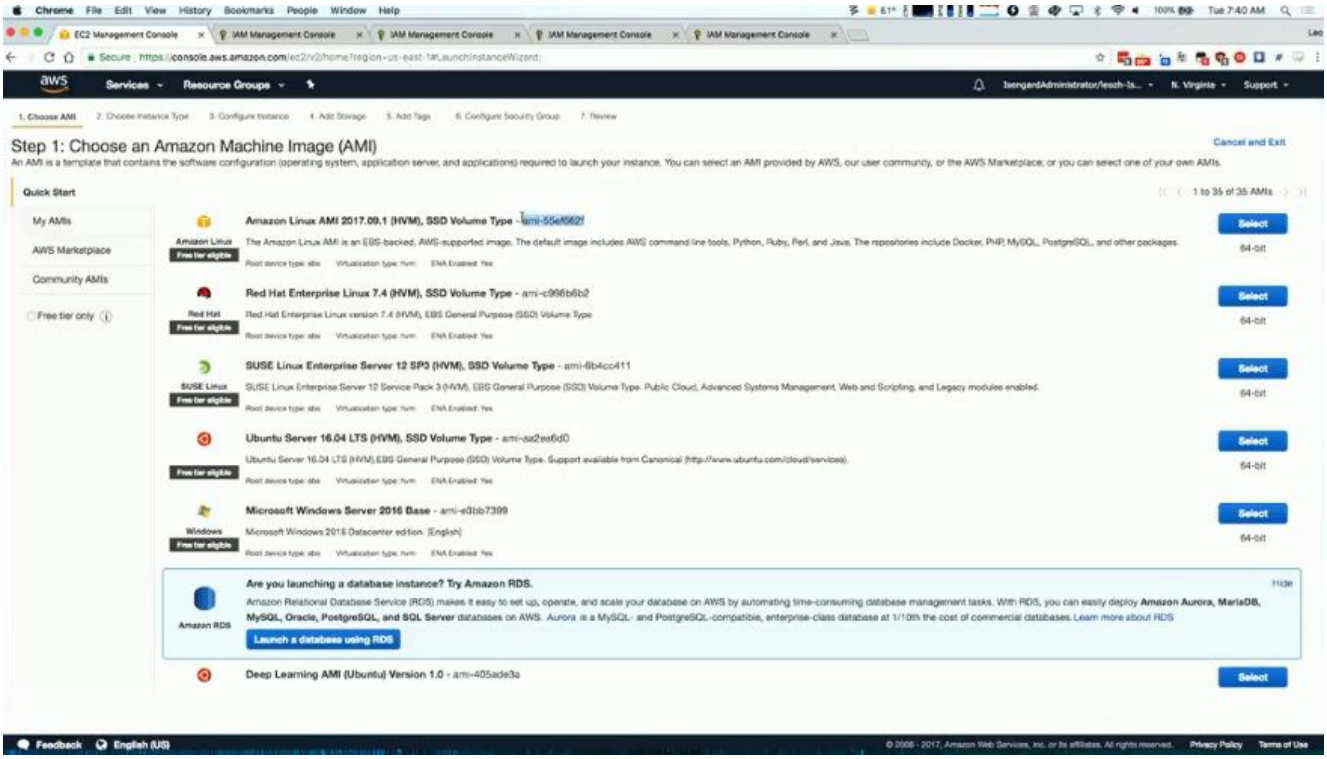


Then display the file generated that contains every option possible using the command ***$ less ec2runinst.json***

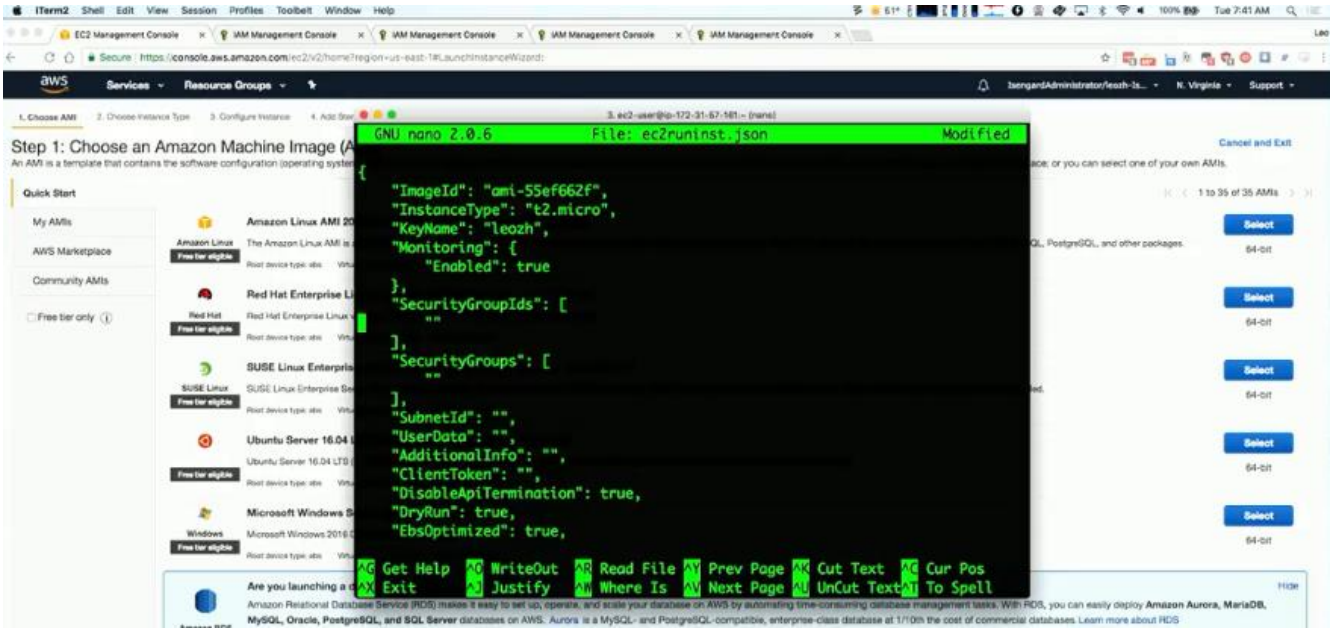


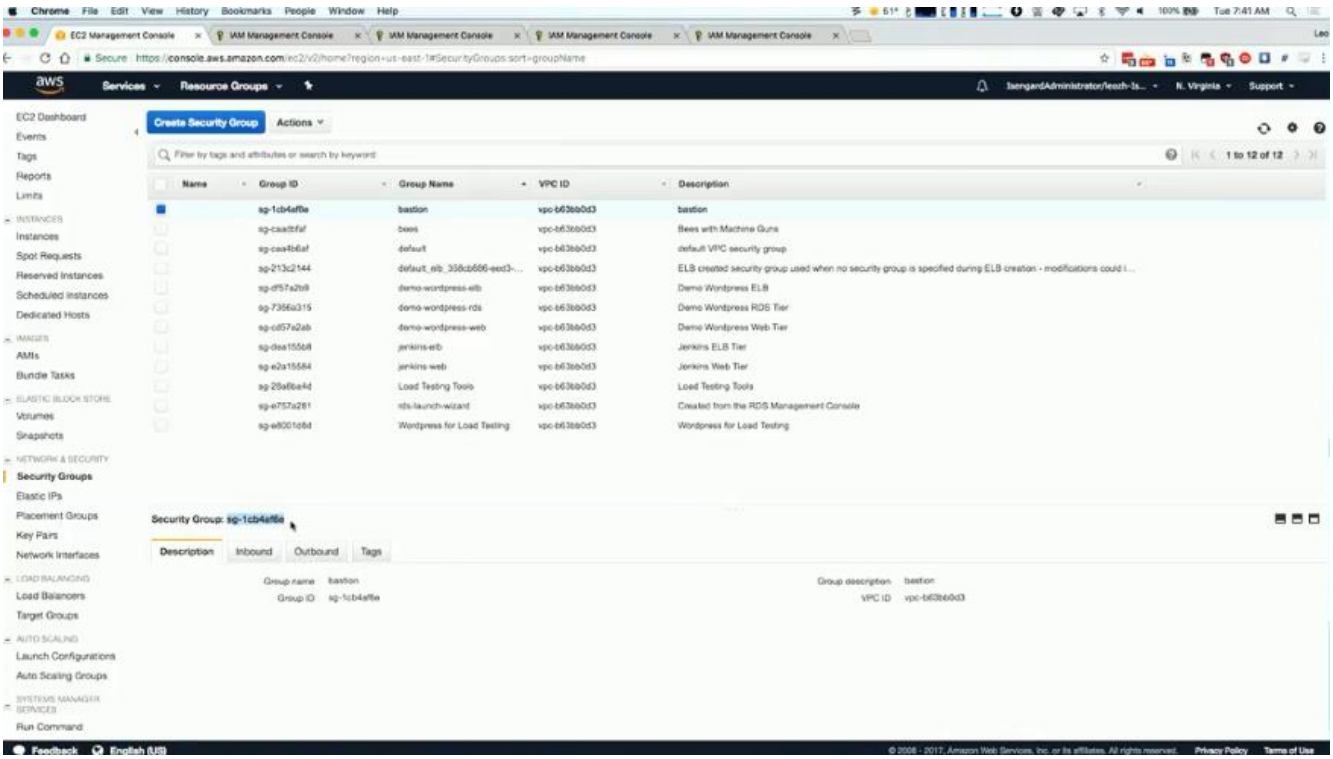
We can now edit the file using the command ***$ nano ec2runinst.json*** to leave the parameters we care about like the AMI-ID, and remove the unneeded fields



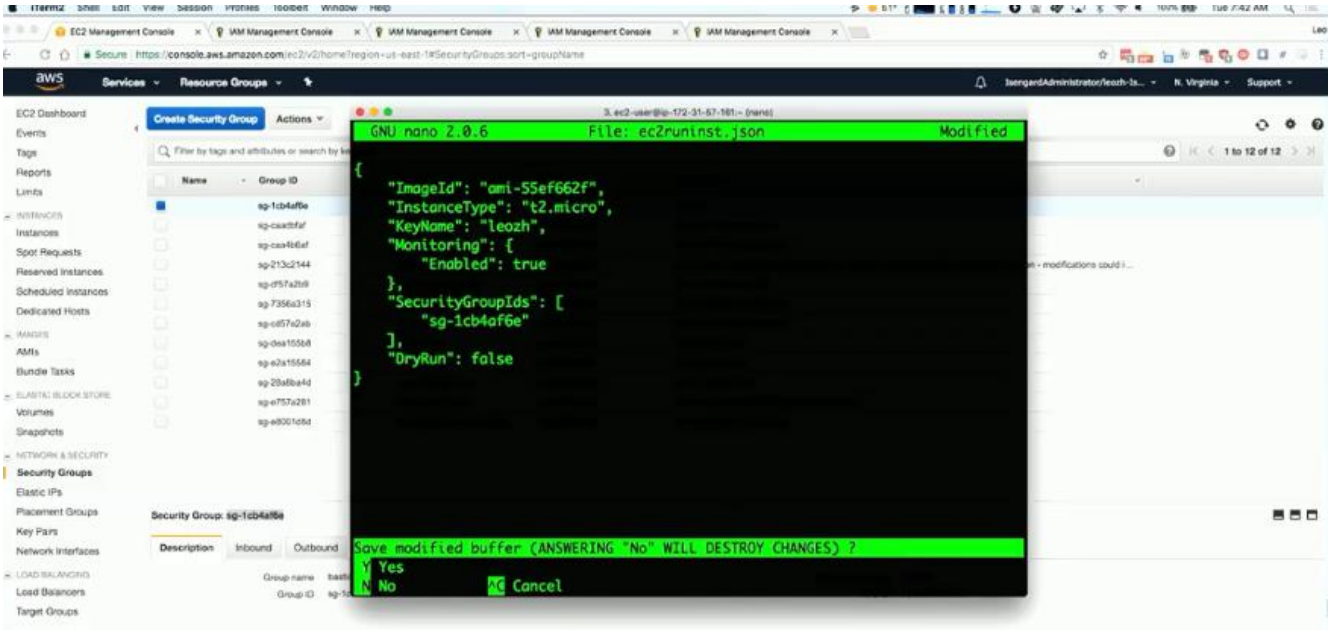


We copy the AMI-id as above

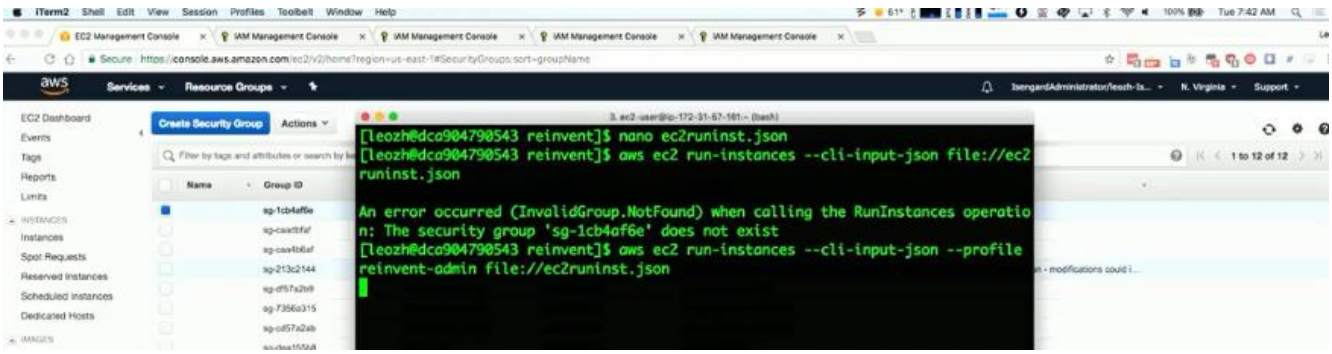




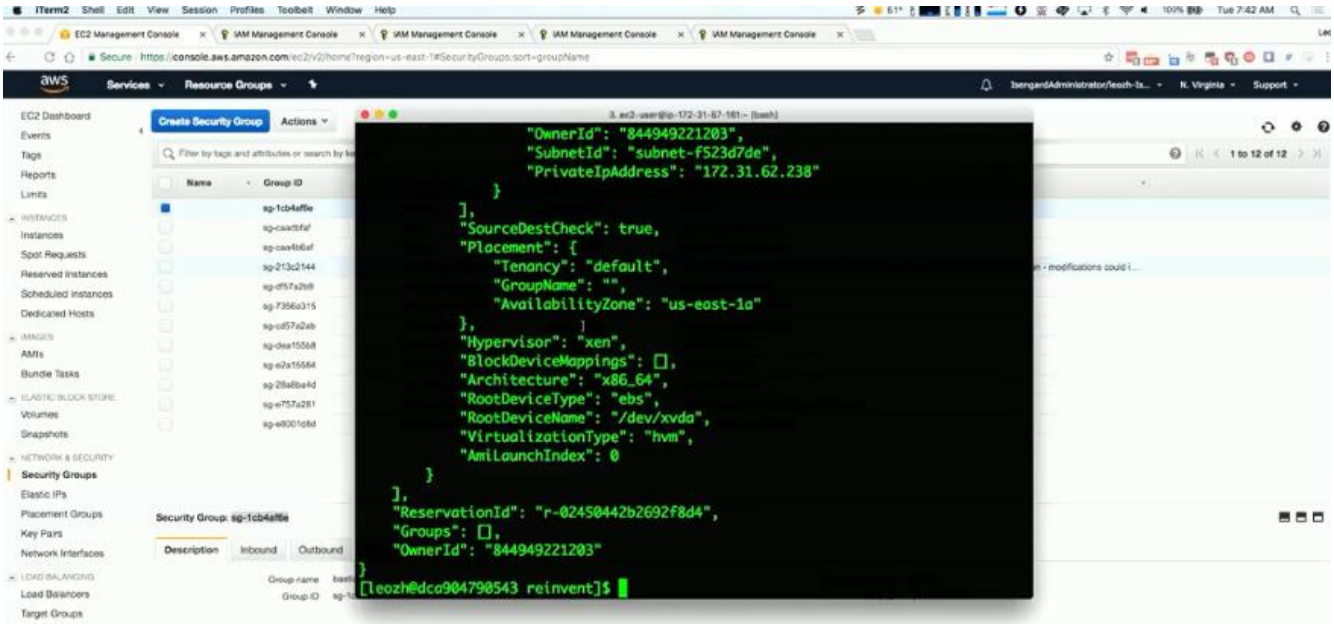
We copy the Security Group Id for our Bastion as above



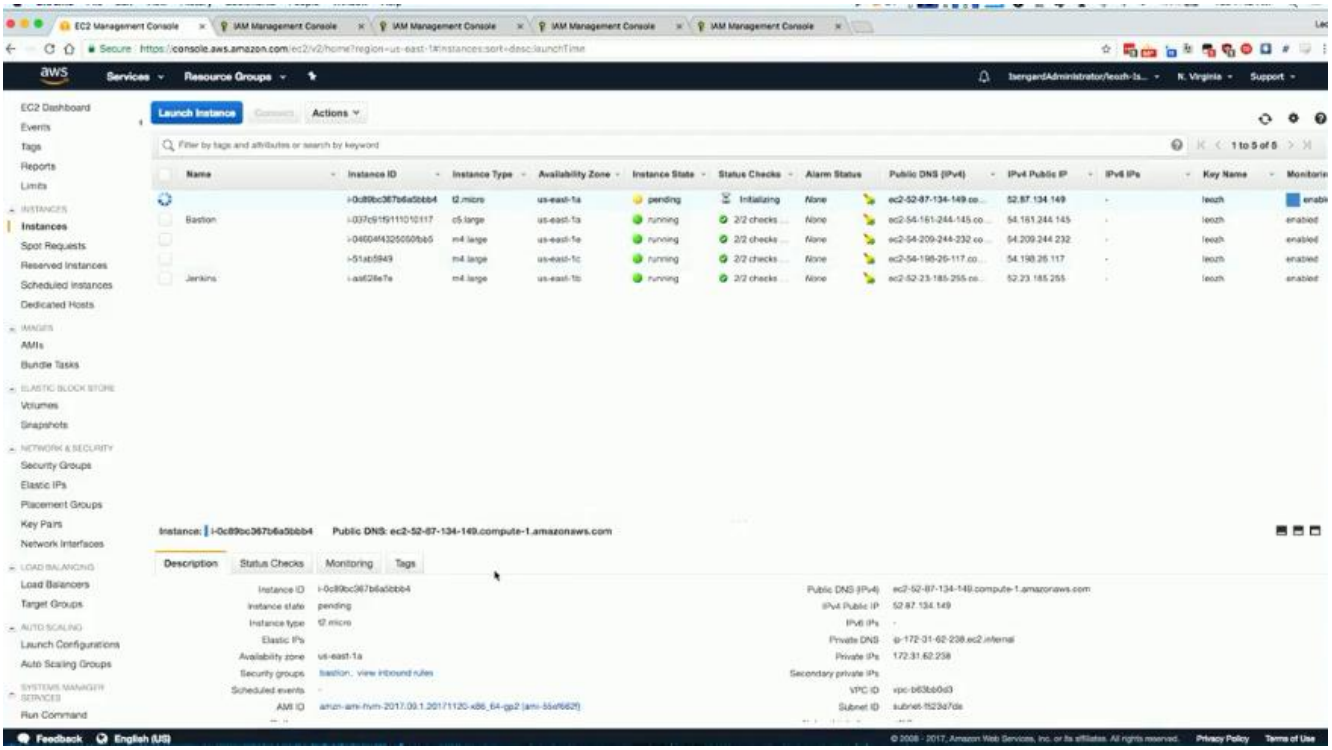
This is what we want



We then create and run the EC2 instance using the correct admin profile and the skeleton file path in the command ***$ aws ec2 run-instances - -cli-input-json - -profile reinvent-admin file://ec2runinst.json*** as above



Our instance is now running with the details we specified

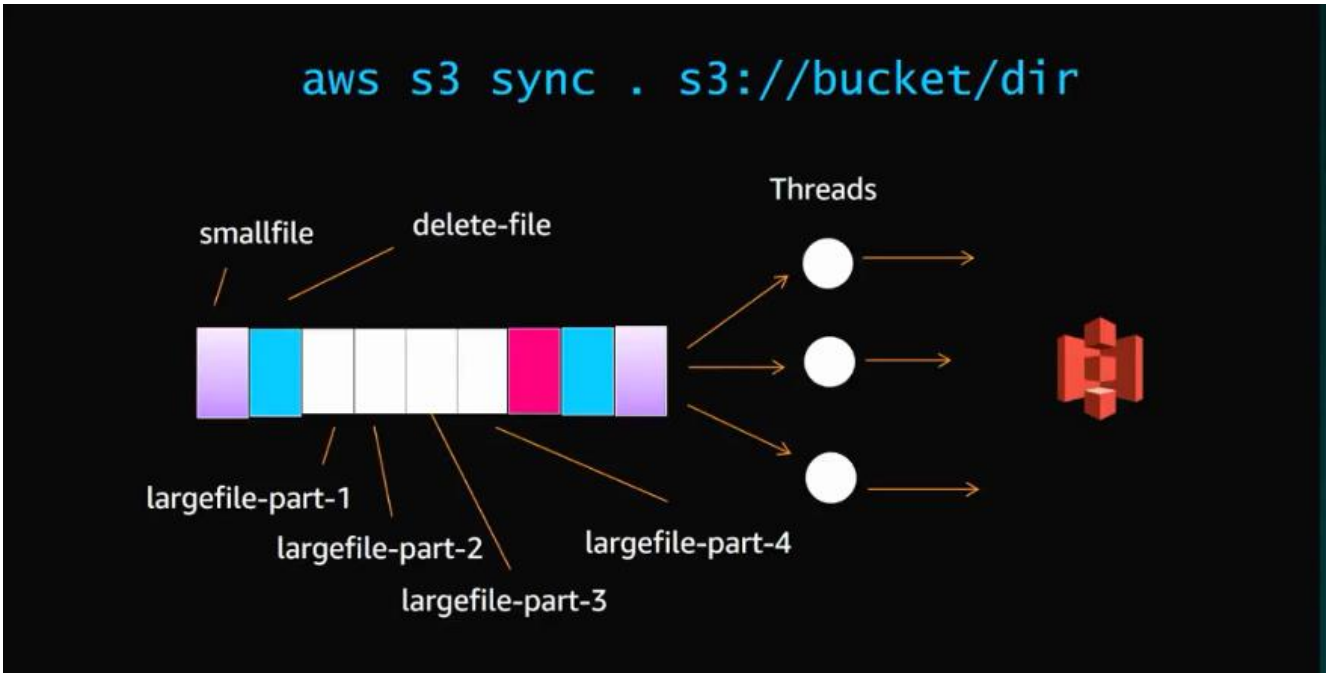



```
aws s3 cp local s3://bucket/key
aws s3 sync . s3://bucket/dir
```

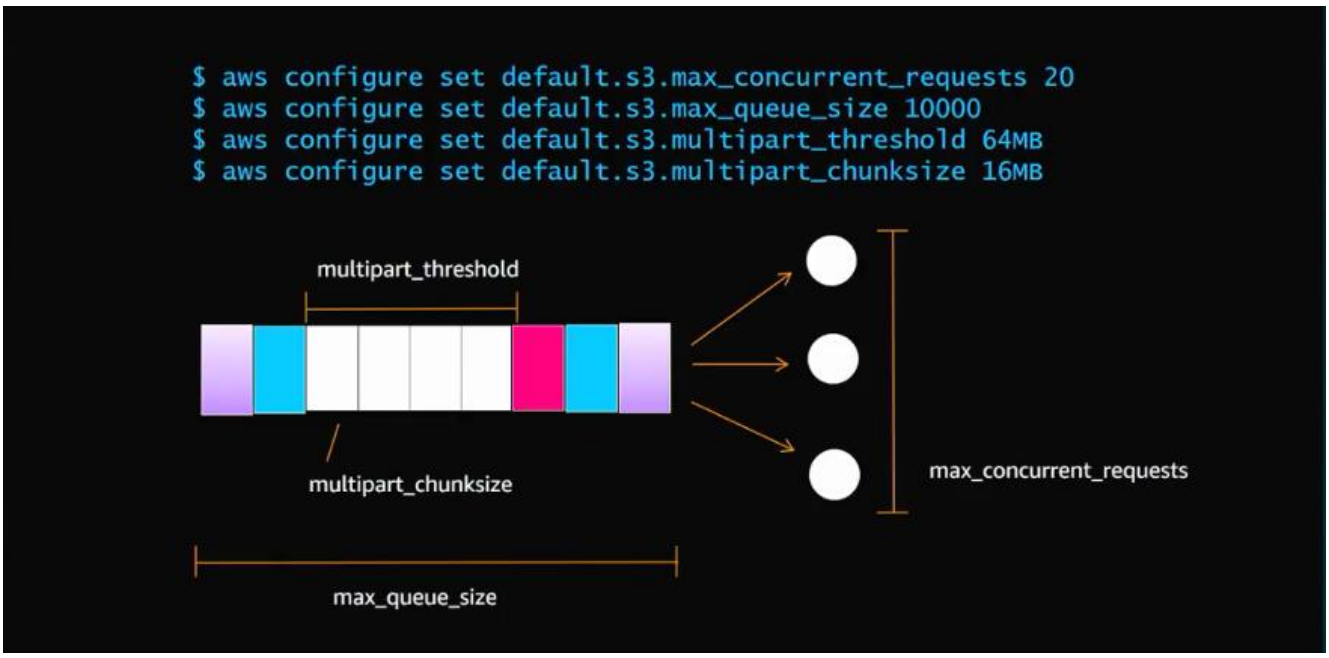
The ***$ aws s3 cp local s3://bucket/key*** command just copies a file from local destination to a remote destination like S3.
The ***$ aws s3 sync . s3://bucket/dir*** command does a sync of the contents

```
aws s3 cp s3://src-bucket/key s3://dest-bucket/key
aws s3 sync s3://src-bucket s3://dest/bucket
```

We can also copy between 2 S3 buckets



These commands are multi-part uploaded and multi-threaded by default for speed



You can also customize the commands as above

Recap

- Many ways to install on multiple platforms
- Use tab completion, help commands and aws-shell to your advantage
- Take advantage of Roles and MFA
- Powerful parsing of CLI command returns with querying, filtering & jmespath
- Generate CLI Skeleton makes scripting with the CLI easier
- Use S3 commands for data migration / backups

Learn More

- **Go to or watch on YouTube**
  - **DEV307 - AWS CLI: 2017 and Beyond**
- **Go to**
  - **https://aws.amazon.com/cli/**
- **Connect with other AWS developers on the CLI community forum**
- **Find examples and more in the User Guide**