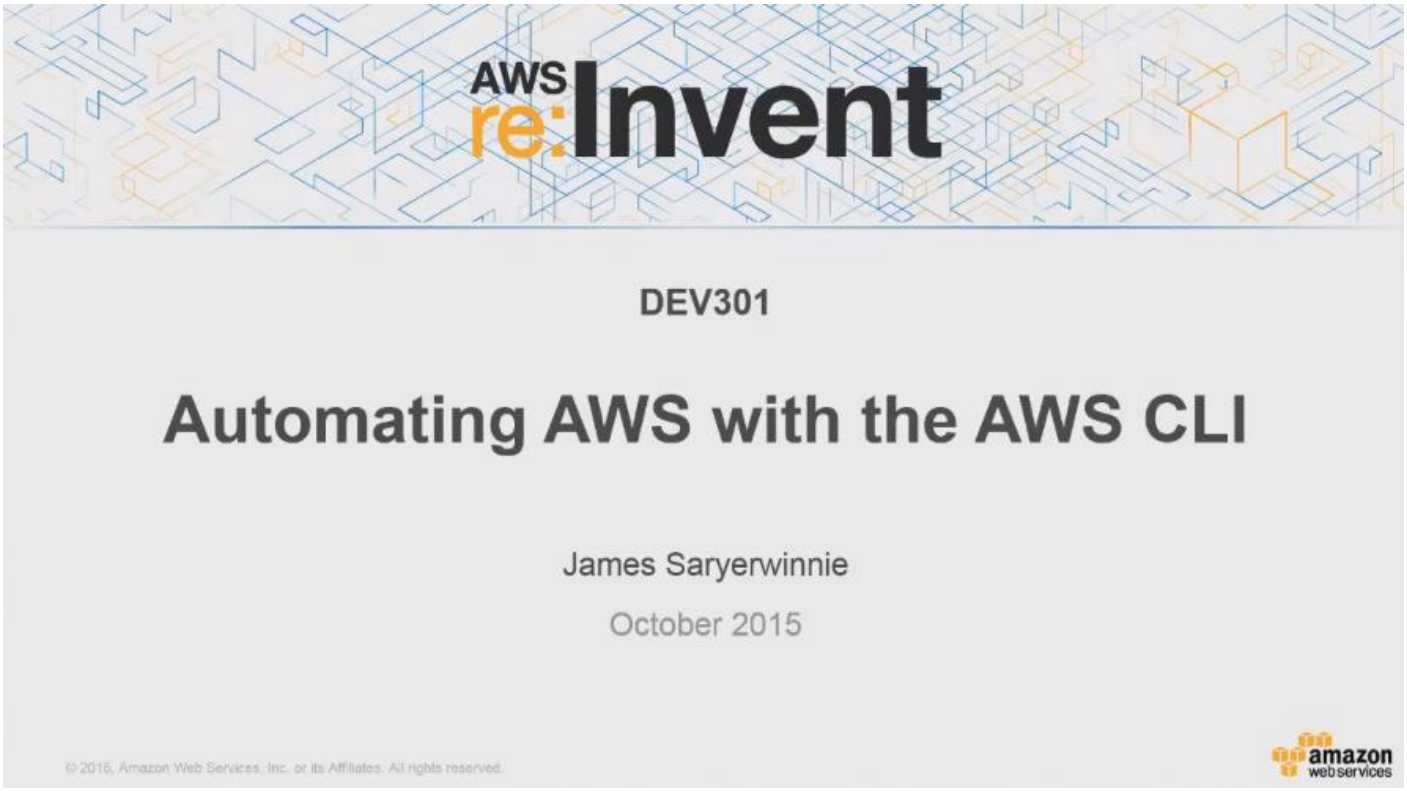
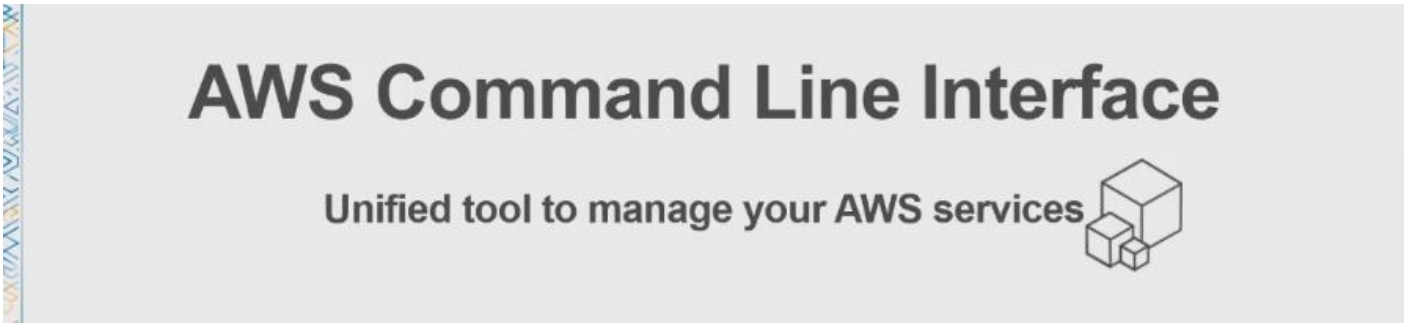
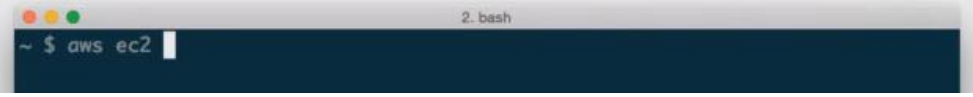


In this talk, you'll learn how you can use the AWS CLI to automate common administrative tasks in AWS. We'll cover several scenarios including EBS snapshot management and S3 backups and see how to combine AWS CLI features to create powerful tools for automation. You'll see how to develop, debug, and deploy these examples in several live, end to end examples.



```
$ git show 1.0.0
tag 1.0.0
Tagger: James Saryerwinnie <js@jamesls.com>
Date:   Mon Sep 2 18:38:51 2013 -0700

Tagging 1.0.0 release.

commit 7fbdac0b19ea9dcc0380242068af20ef425ac5c3
Merge: 6f76721 469bab6
Author: James Saryerwinnie <js@jamesls.com>
Date:   Mon Sep 2 18:38:51 2013 -0700

    Merge branch 'release-1.0.0'

    * release-1.0.0:
      Bumping version to 1.0.0
```

Interactive



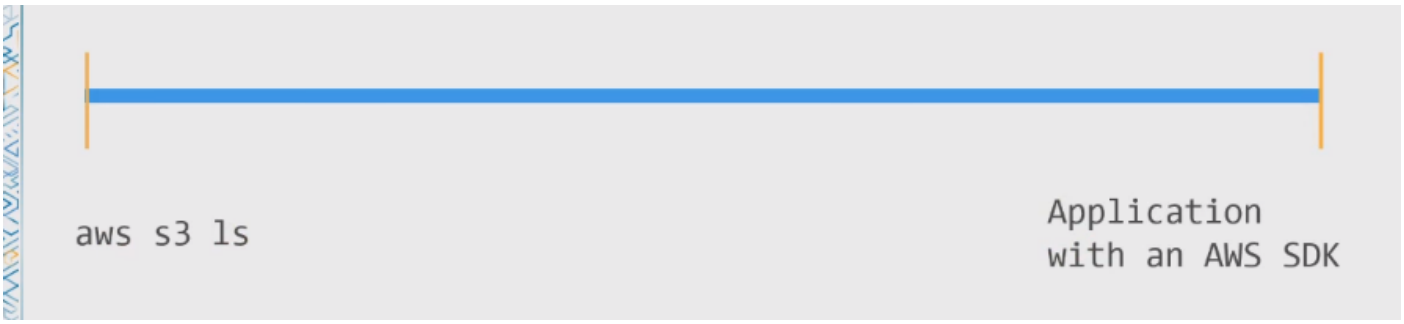
We generally use this interactive interaction with the AWS CLI for exploratory tasks of some new API or doing simple things like copying files to S3

Shell scripts

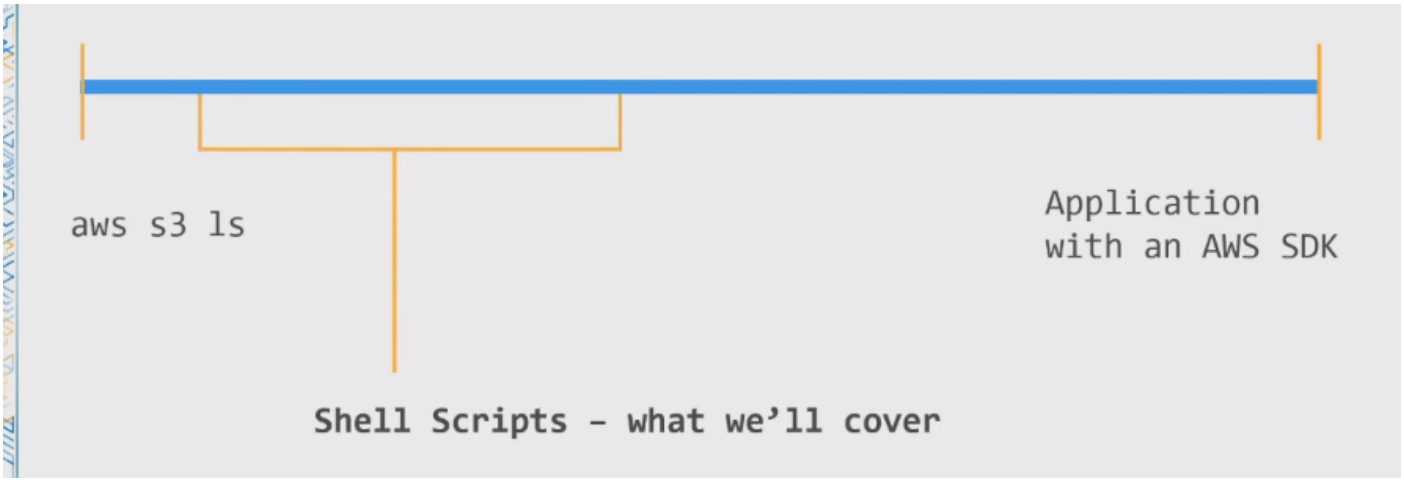
```
#!/bin/bash

import_key_pair() {
  echo -n "Would you like to import ~/.ssh/id_rsa.pub? [y/N]: "
  read confirmation
  if [[ "$confirmation" != "y" ]]
  then
    return
  fi
  aws ec2 import-key-pair \
    --key-name id_rsa \
    --public-key-material file://~/.ssh/id_rsa.pub
}

create_instance_profile() {
  echo -n "Would you like to create an IAM instance profile? [y/N]: "
  read confirmation
  if [[ "$confirmation" != "y" ]]; then
    return
  fi
  aws iam create-role --role-name dev-ec2-instance \
    --assume-role-policy-document "$TRUST_POLICY" || errexit "Could not create Role"

  # Use a managed policy
  policies=$(aws iam list-policies --scope AWS)
  admin_policy_arn=$(jp -u \
    "Policies[?PolicyName=='AdministratorAccess'].Arn | [0]" <<< "$policies")
  aws iam attach-role-policy \
    --role-name dev-ec2-instance \
    --policy-arn "$admin_policy_arn" || errexit "Could not attach role policy"

  # Then we need to create an instance profile from the role.
  aws iam create-instance-profile \
    --instance-profile-name dev-ec2-instance || \
    errexit "Could not create instance profile."
  # And add it to the role
  aws iam add-role-to-instance-profile \
    --role-name dev-ec2-instance \
    --instance-profile-name dev-ec2-instance || \
    errexit "Could not add role to instance profile."
}

compute_key_fingerprint() {
  # Computes the fingerprint of a public SSH key given a private
  # RSA key.  This can be used to compare against the output given
  # from aws ec2 describe-key-pair.
  openssl pkey -in ~/.ssh/id_rsa -pubout -outform DER | \
    openssl md5 -c | \
    cut -d = -f 2 | \
    tr -d '[:space:]'
}
```

```
do_setup() {
  echo "Checking for required resources..."
  echo ""
  # 1. Check if a security group is found for
  #    both windows and non windows tags.
  #    If not, we'll eventually give the option to
  #    configure this.
  if resource_exists "aws ec2 describe-security-groups \
    --filter Name=tag:dev-ec2-instance,Values=non-windows"; then
    echo "Security groups exists."
  else
    echo "Security group not found."
  fi
  echo ""

  # 2. Make sure the keypair is imported
  if [[ ! -f ~/.ssh/id_rsa ]]; then
    echo "Missing ~/.ssh/id_rsa key pair."
  elif has_new_enough_openssl; then
    fingerprint=$(compute_key_fingerprint ~/.ssh/id_rsa)
    if resource_exists "aws ec2 describe-key-pairs \
      --filter Name=fingerprint,Values=$fingerprint"; then
      echo "Key pair exists."
    else
      echo "~/.ssh/id_rsa key pair does not appear to be imported."
      import_key_pair
    fi
  else
    echo "Can't check if SSH key has been imported."
    echo "You need at least openssl 1.0.0 that has a \"pkey\" command."
    echo "Please upgrade your version of openssl."
  fi
  echo ""

  # 3. Check that they have an IAM role called dev-ec2-instance.
  #    There is no server side filter for this like we have with the EC2
  #    APIs, so we have to manually use a --query option for us.
  if resource_exists "aws iam list-instance-profiles" \
    "length(InstanceProfiles[?InstanceProfileName=='dev-ec2-instance'])"; then
    echo "Instance profile exists."
  else
    echo "Missing IAM instance profile 'dev-ec2-instance'"
    create_instance_profile
  fi
  echo ""
}

do_setup
```

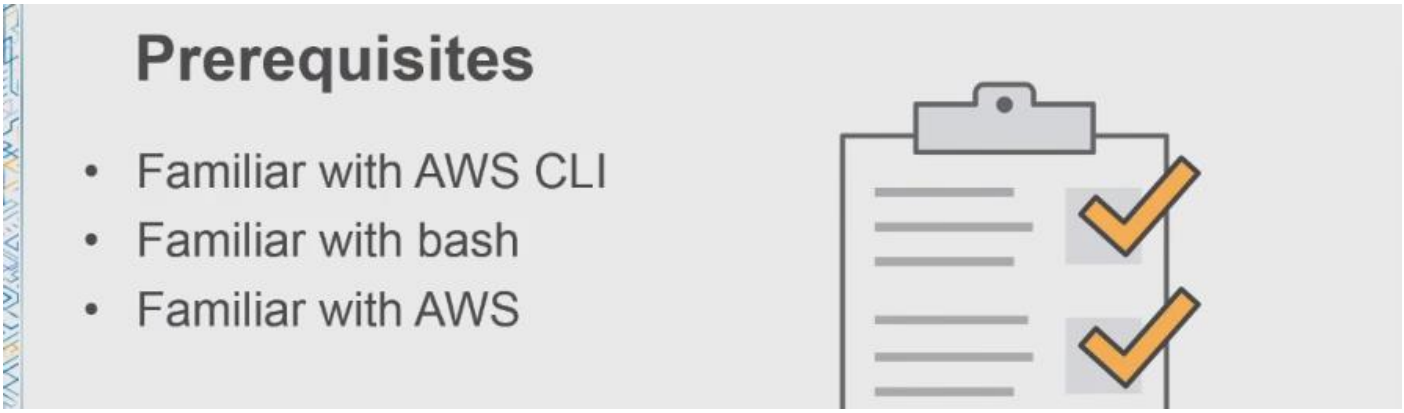
Customers are also taking the AWS CLI commands and binding them together just like a toolkit or SDK to build higher level abstractions to get some complicated task done programmatically. This approach combines local commands with AWS CLI commands to provide some useful functionality.



We can use the CLI to make a single S3 command to list all of our buckets, we can write full web apps that has many components that integrate with many AWS services and has a lot of domain specific logic for our business and application



This is for things like 'if resource exists or command fails, then take action A, otherwise take action B' tasks.



Prerequisites

```
$ aws ec2 describe-instances --filters Name=architecture,Values=x86_64

$ du $(ls | grep cli) | sort -nr | tail -2
```

Amazon EC2

Amazon S3

The 2nd command is looking for the 2 largest files in our current directory.

Getting up to speed

- Becoming a Command Line Expert with the AWS CLI
- Advanced Usage of the AWS CLI
- Installation
- Configuration

We will be in the terminal…

Common patterns

```
/tmp/demos $ aws ec2 describe-regions
```

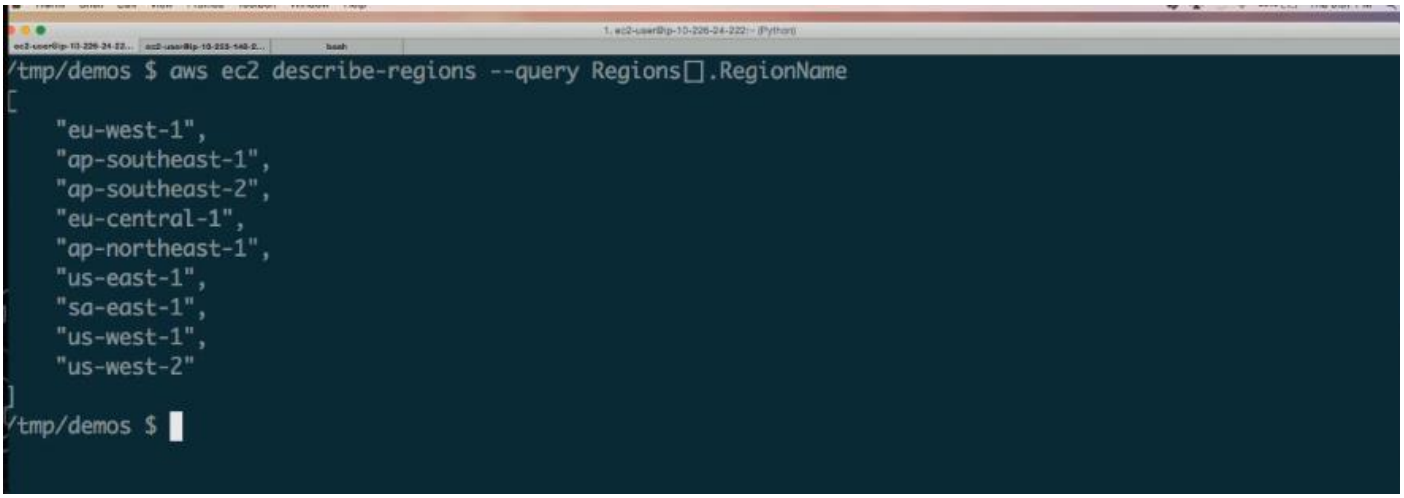
We can use the ***$ aws ec2 decsribe-regions*** command as above

```
        },
        {
            "Endpoint": "ec2.ap-northeast-1.amazonaws.com",
            "RegionName": "ap-northeast-1"
        },
        {
            "Endpoint": "ec2.us-east-1.amazonaws.com",
            "RegionName": "us-east-1"
        },
        {
            "Endpoint": "ec2.sa-east-1.amazonaws.com",
            "RegionName": "sa-east-1"
        },
        {
            "Endpoint": "ec2.us-west-1.amazonaws.com",
            "RegionName": "us-west-1"
        },
        {
            "Endpoint": "ec2.us-west-2.amazonaws.com",
            "RegionName": "us-west-2"
        }
    ]
}
/tmp/demos $
```

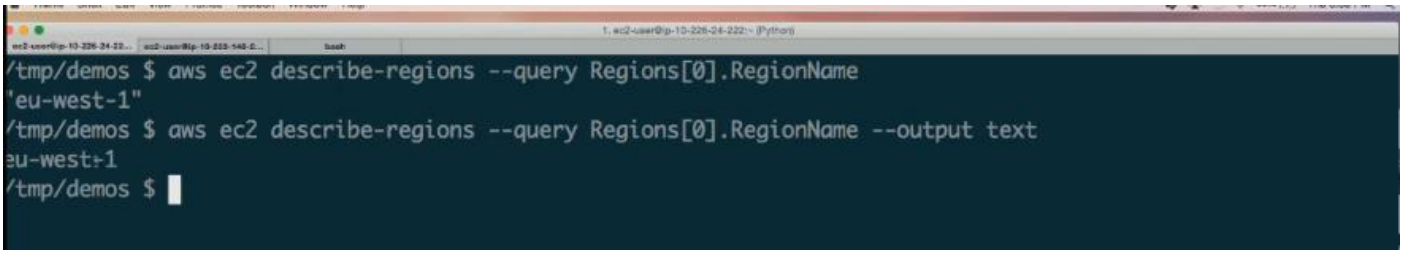
We will be using output text

```
/tmp/demos $ aws ec2 describe-regions --output text
REGIONS ec2.eu-west-1.amazonaws.com      eu-west-1
REGIONS ec2.ap-southeast-1.amazonaws.com        ap-southeast-1
REGIONS ec2.ap-southeast-2.amazonaws.com        ap-southeast-2
REGIONS ec2.eu-central-1.amazonaws.com  eu-central-1
REGIONS ec2.ap-northeast-1.amazonaws.com        ap-northeast-1
REGIONS ec2.us-east-1.amazonaws.com     us-east-1
REGIONS ec2.sa-east-1.amazonaws.com     sa-east-1
REGIONS ec2.us-west-1.amazonaws.com     us-west-1
REGIONS ec2.us-west-2.amazonaws.com     us-west-2
/tmp/demos $
```

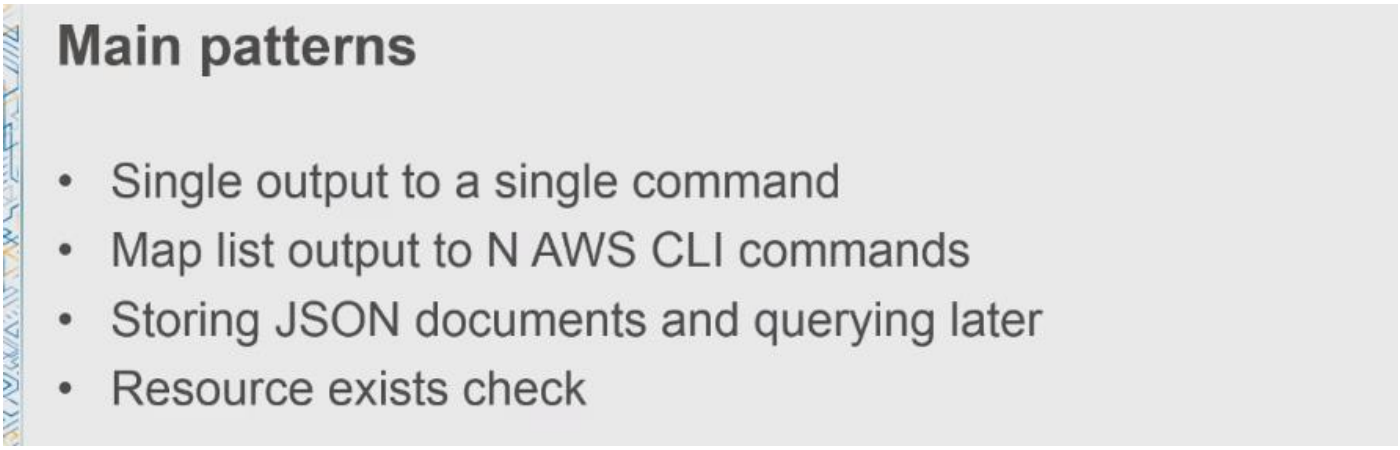
We can add the ***- -output text*** flag the ***$ aws ec2 decsribe-regions*** command to see the result in the tabular layout with tab separated data as above



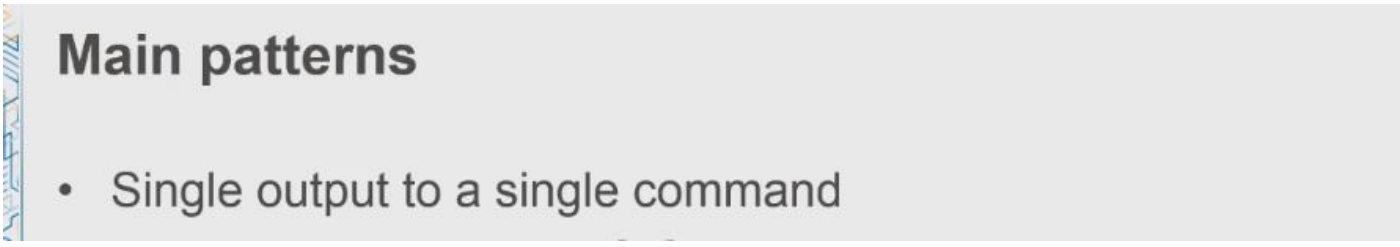
We can use the ***$ aws ec2 dsecribe-regions - -query Regions[].RegionName*** command above to do a client side filter on the query for a subset of the data that we want

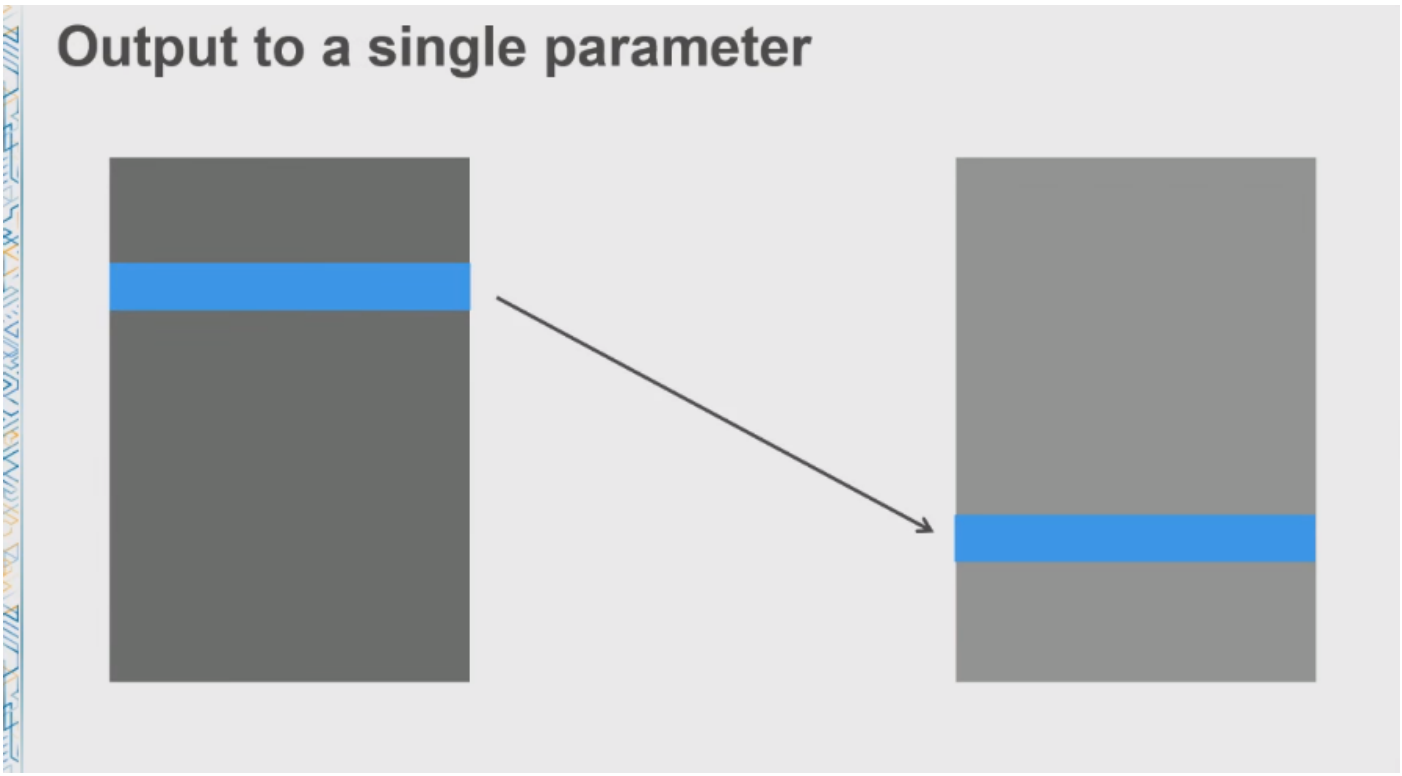


We can also grab the first element in the query result using ***$ aws ec2 dsecribe-regions - -query Regions[0].RegionName*** command, notice that the result is in JSON. We can specify the ***- -output text*** flag to get a text string output to feed into the input of some other downstream command.

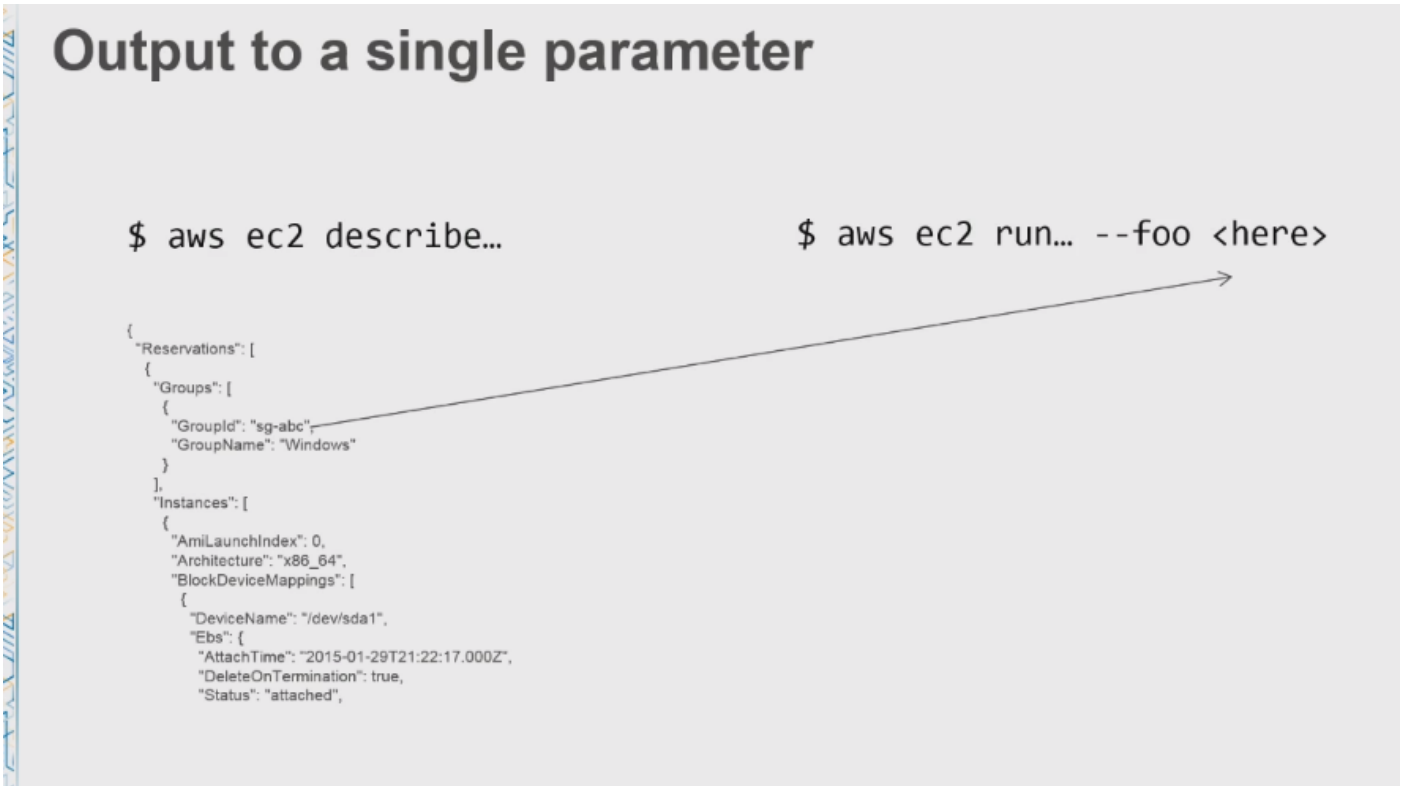


These 4 patterns are frequently used in the CLI, they allow you to automate most of the AWS tasks that you might have.

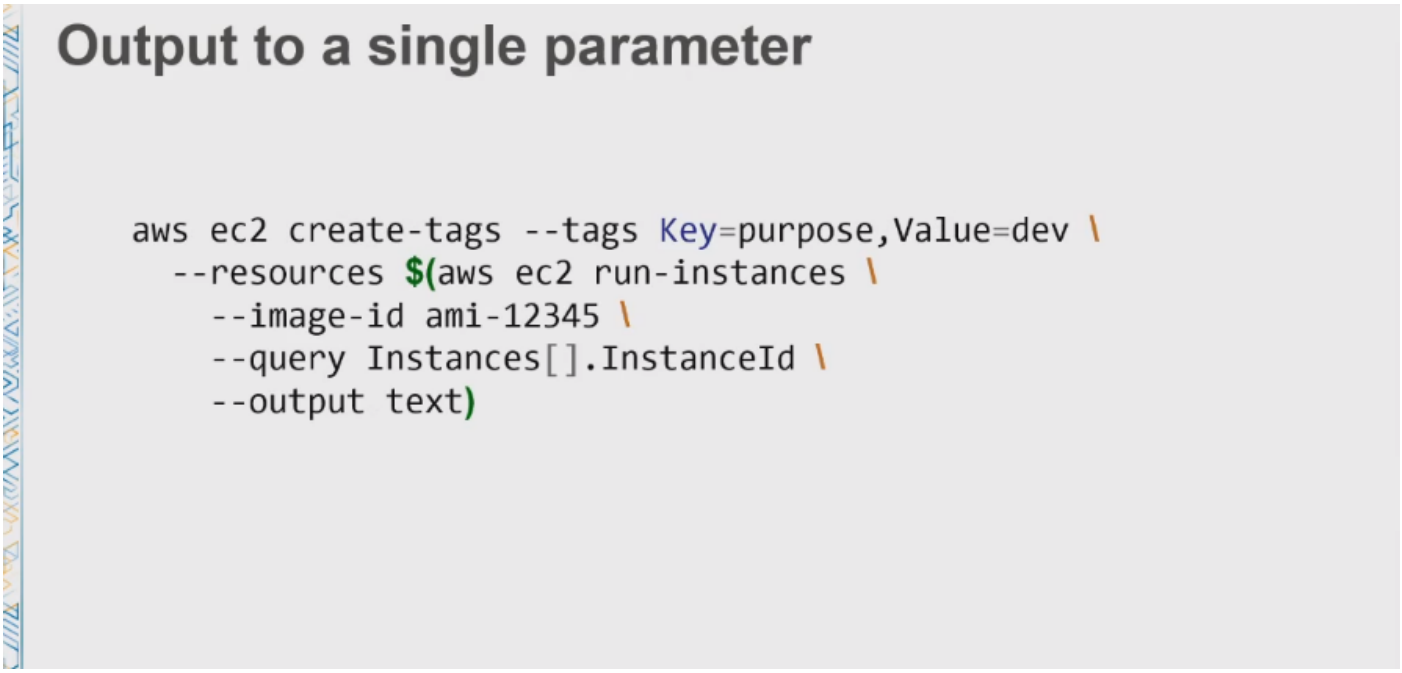




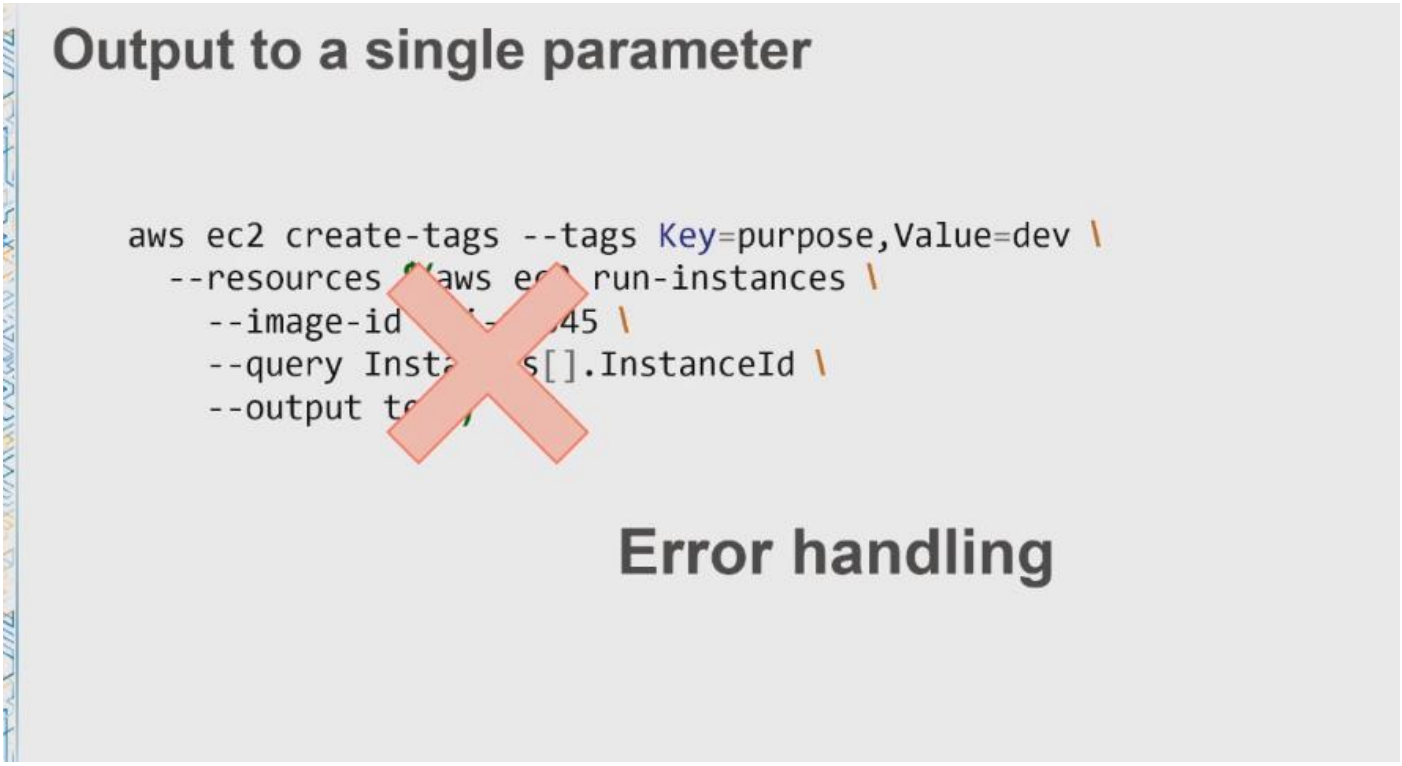
This is when we want to take output from one command and feed it as the input to another command.



We might run a ***$ aws ec2 describe*** command and feed the result into a run or delete command as above



You can try the above method that uses command substitution, the intent above is to run an EC2 instance and then tag it as soon as we start running it



This is not good because our code has o way of knowing if something went wrong

Output to a single parameter

```
instance_ids=$(aws ec2 run-instances \
  --image-id ami-12345 \
  --query Instances[].InstanceId \
  --output text) || errexit "Could not run instance"
aws ec2 create-tags \
  --tags Key=purpose,Value=dev \
  --resources "$(instance_ids)" || errexit "<errmsg>"
```

Above is the best way to do this pattern, you explicitly call out and name (like ***instance_ids***) each thing you want to do like looking for the InstanceId's from all instances, then you use that value as ***$(instance_ids)*** in the ***create-tags*** command. The ***errexit*** is just a helper function that prints the error message that occurs and then exits. The above code will give you an explicit error if something goes wrong.

Output to a single parameter

```
instance_ids=$(aws ec2 run-instances \
  --image-id ami-12345 \
  --query Instances[].InstanceId \
  --output text) || errexit "Could not run instance"
aws ec2 create-tags \
  --tags Key=purpose,Value=dev \
  --resources "$(instance_ids)" || errexit "<errmsg>"
```


We are running the ***- -query*** flag to grab the ***InstanceIds***, we then use the ***- -output text*** flag so that we can feed the output to another command downstream as an input

Output to a single parameter

```
instance_ids=$(aws ec2 run-instances \
  --image-id ami-12345 \
  --query Instances[].InstanceId \
  --output text) || errexit "Could not run instance"
aws ec2 create-tags \
  --tags Key=purpose,Value=dev \
  --resources "$(instance_ids)" || errexit "<errmsg>"
```


We then use the result in the next command to create-tags as above

Output to a single parameter

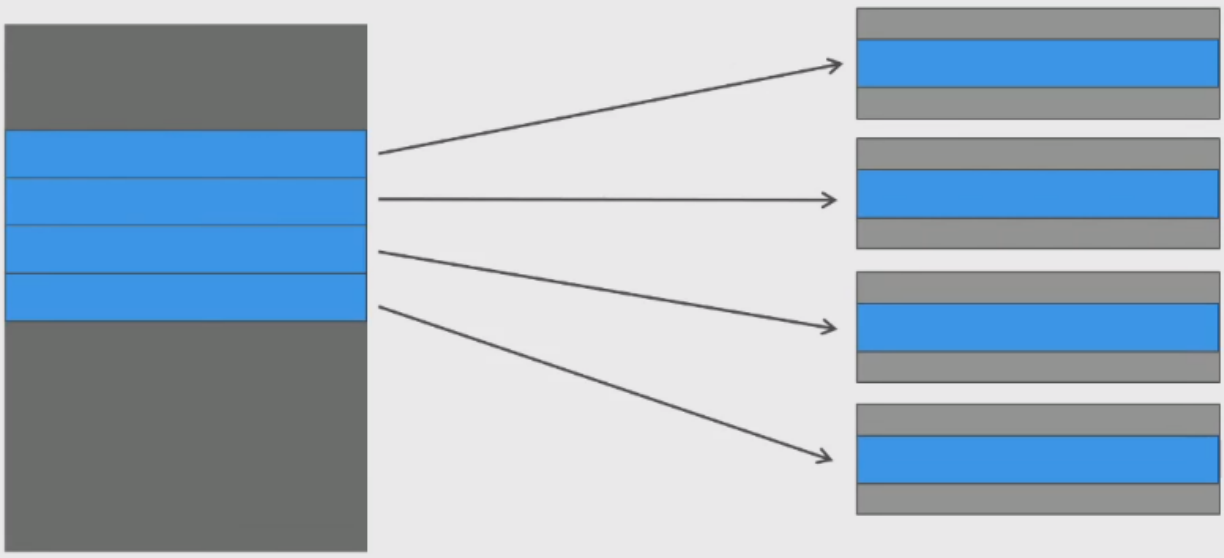
- Used to map one part of the output to one part of the input
- Use --query and --output text
- Make sure to handle the error case before using the value

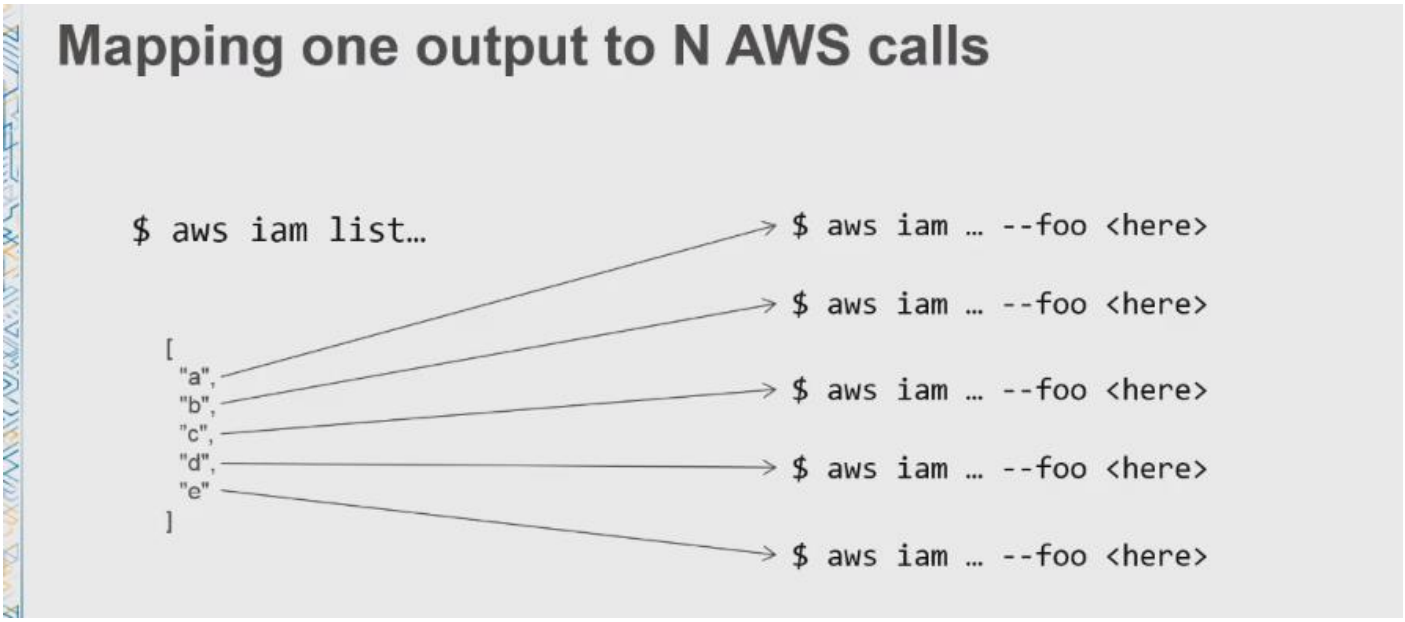
Main Patterns

- Single output to a single command
- Map list output to N AWS CLI commands
- Storing JSON documents and querying later
- Resource exists check

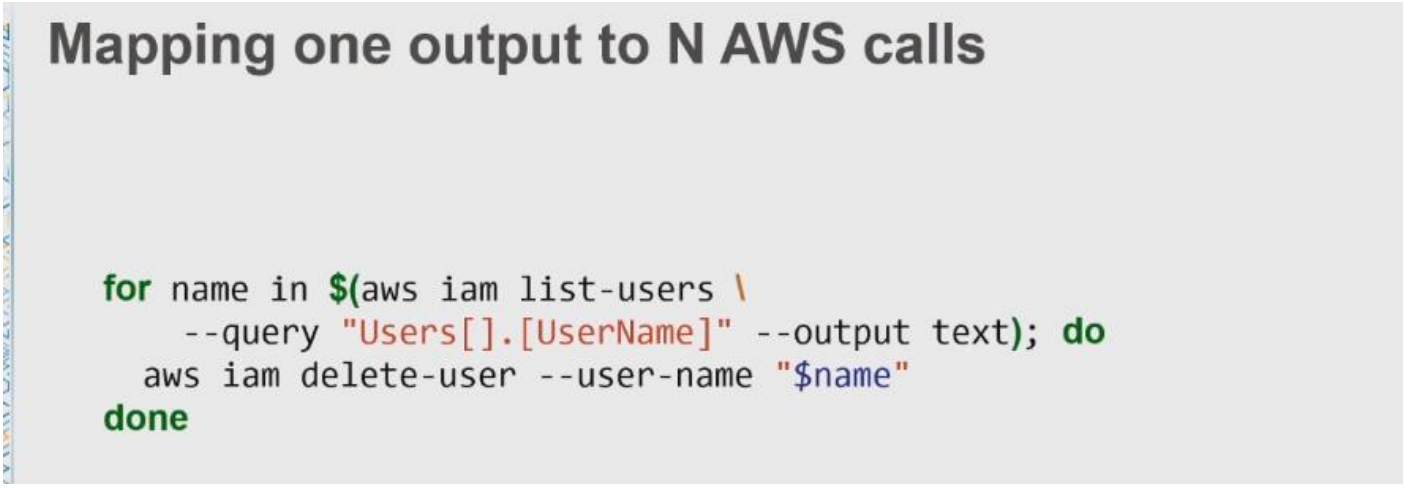
This is when we want to map the results of a command to N number of further commands

Mapping one output to N AWS calls





We can use this approach to list all our IAM users and for each of them we can call some command like add-policy or delete-user. There are 2 ways to do this.



You can use a ***for…do…done*** loop in bash as above,

Mapping one output to N AWS calls

```
for name in $(aws iam list-users \
    --query "Users[].[UserName]" --output text); do
  aws iam delete-user --user-name "$name"
done
```

```
User1
User2
User3
User4
User5
```

The first command is going to create line-separated output as above,

Mapping one output to N AWS calls

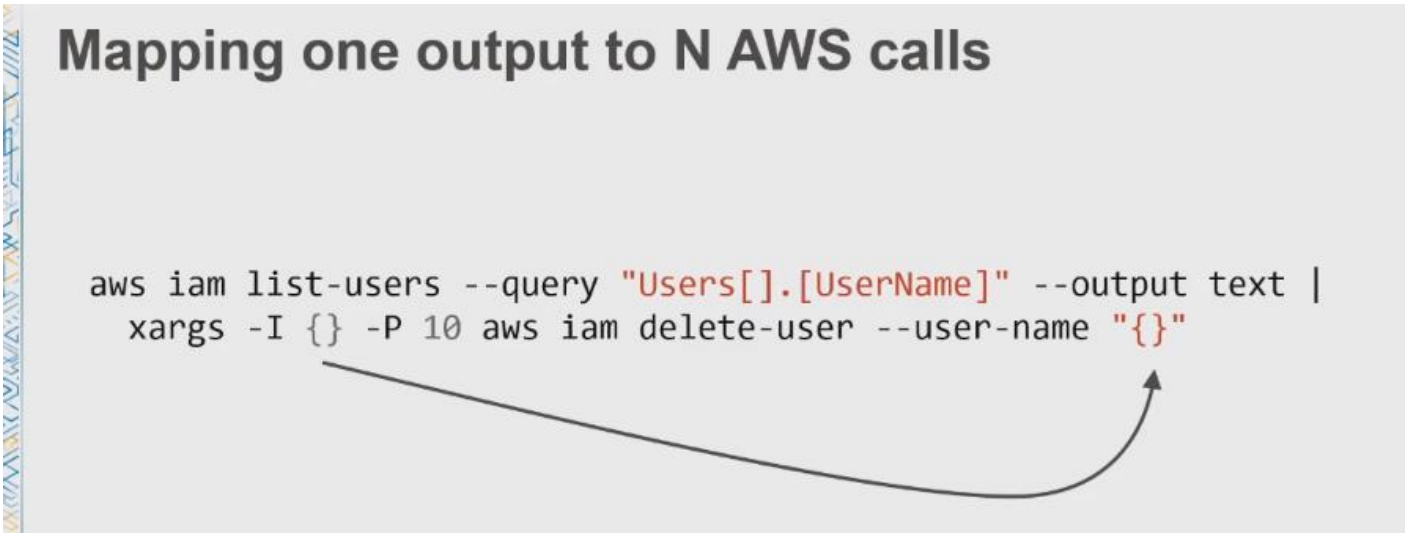
```
for name in $(aws iam list-users \
    --query "Users[].[UserName]" --output text); do
  aws iam delete-user --user-name "$name"
done
```

From that for…do…done loop, we are then going to call delete user using the second command as above

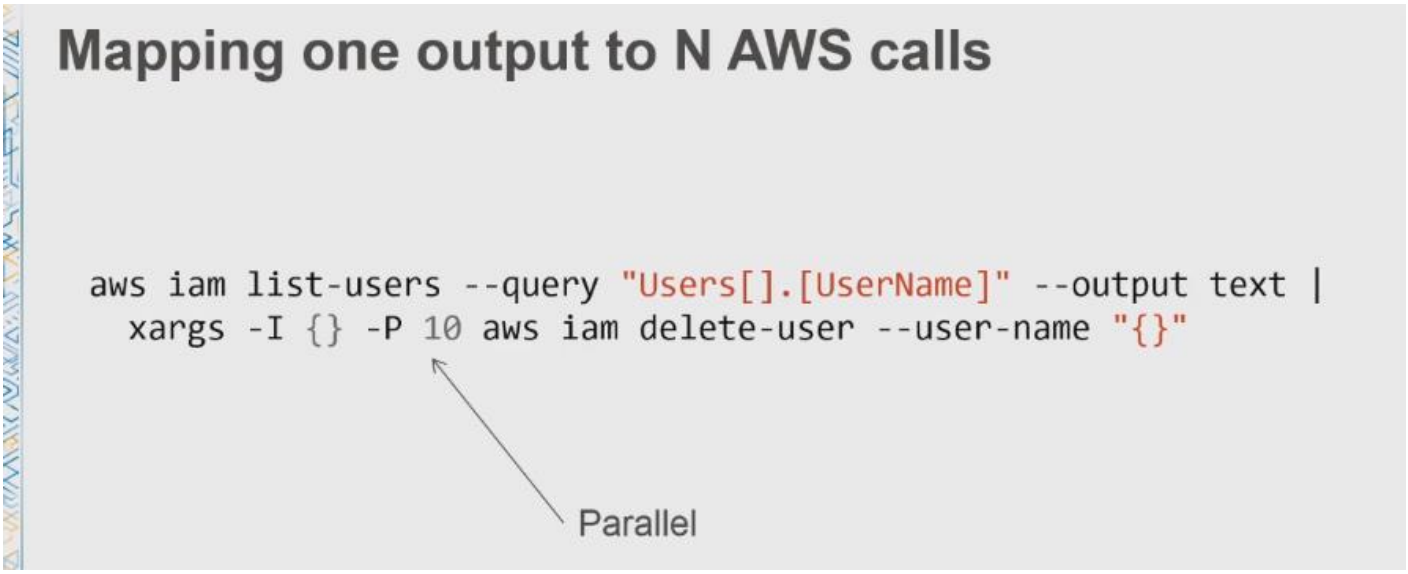
Mapping one output to N AWS calls

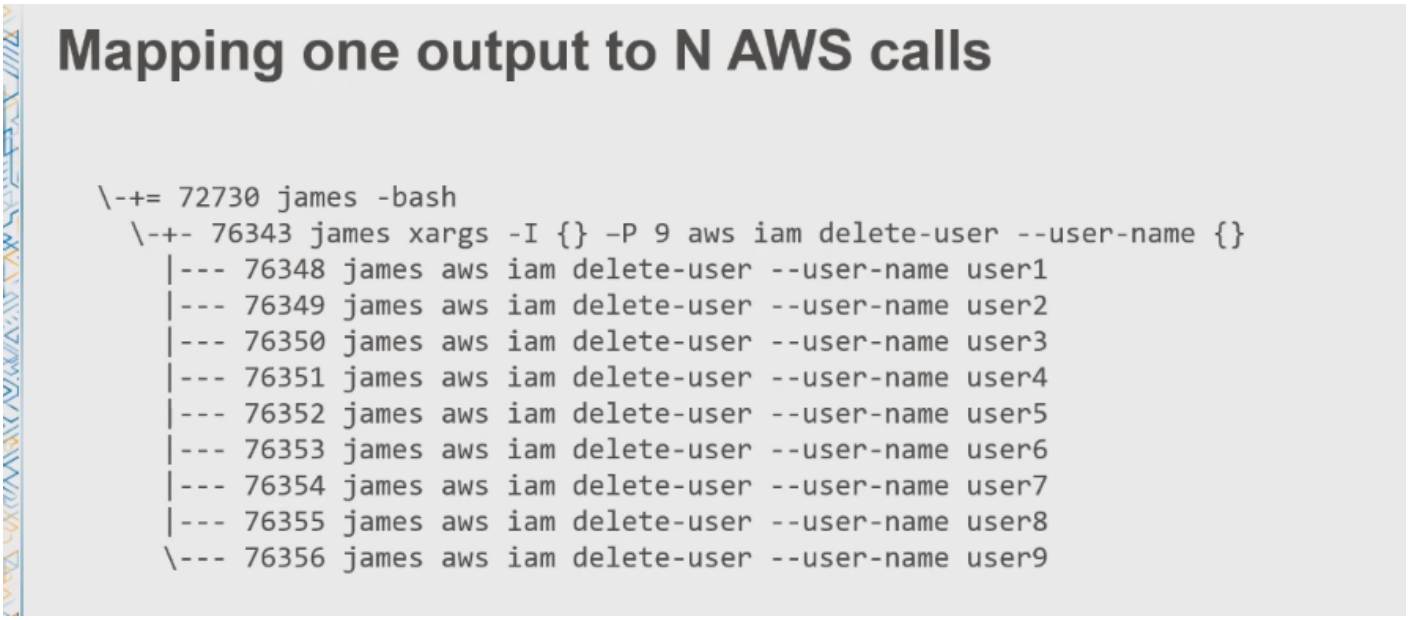
```
aws iam list-users --query "Users[].[UserName]" --output text |
  xargs -I {} -P 10 aws iam delete-user --user-name "{}"
```

The 2nd approach to doing N calls from the result of a first call is to use ***xargs***, xargs are a way to read from the standard input of some process that is producing outputs and map those to command line parameters.



This is good for when we have independent tasks where the N downstream calls don't need to combine their results into a single result, we can run this command the -I flag and also to run in parallel use the -P flag



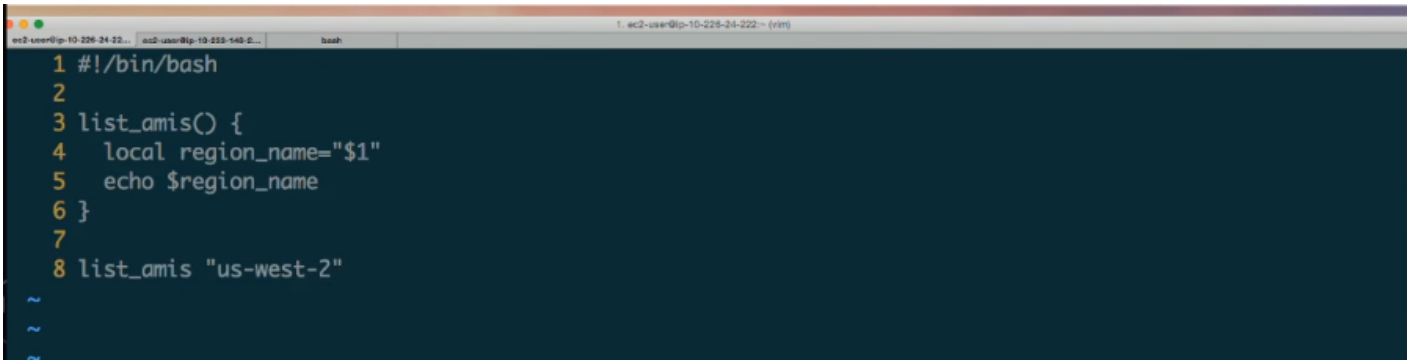


What we will see are 10 child processes running at the same time with the username filled in from the line output

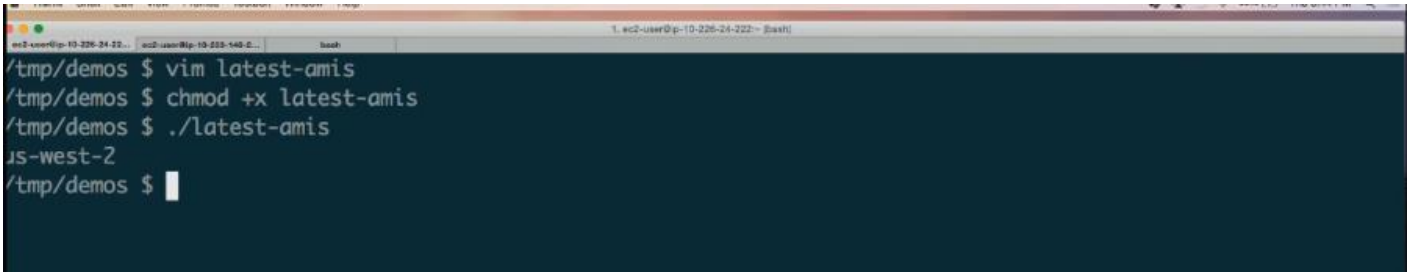
Mapping one output to N AWS calls

- Use a for loop for functions
- If you're using xargs, use `-I {}`
- Use `xargs -P N` parallel execution
- Use ".[UserName]" instead of ".UserName" to get newline separated output..
- This works because nothing is written to stdout on error

Demo



We can have a development EC2 instance up and running using the latest Amazon Linux AMI image, we will want it to spin up, verify SSH access is available. First, we need to write a script that gives us the latest Amazon Linux AMI image,



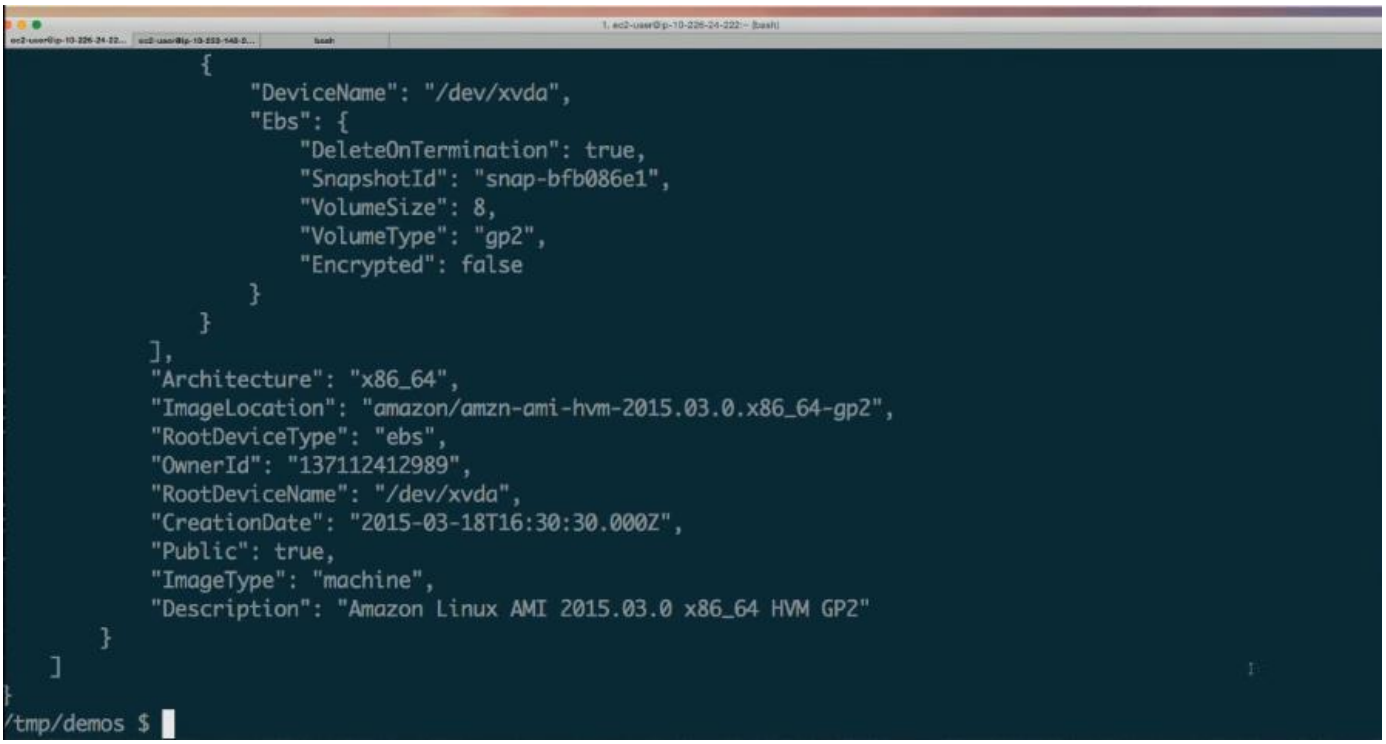
We can see that it indeed see the echo out as expected. Now we can start working on script below

```
#!/bin/bash

list_amis() {
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name"
}

list_amis "us-west-2"
~
```

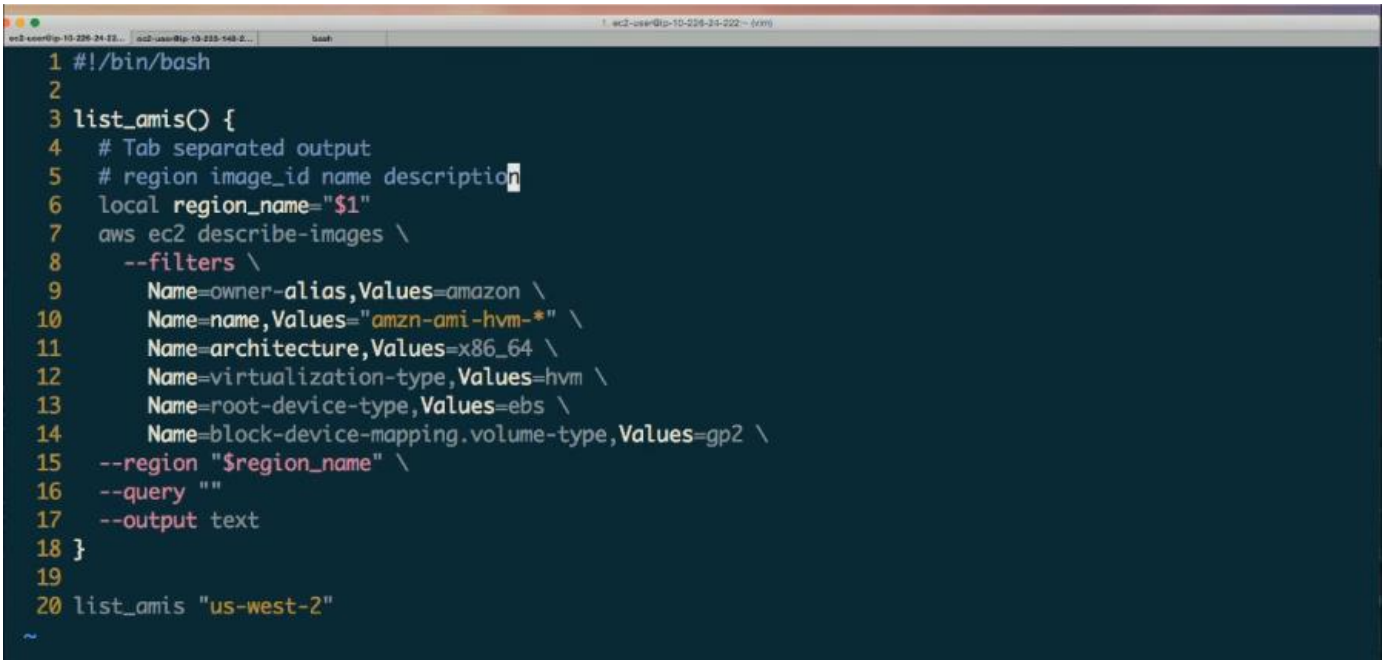
The call we want is the ***describe-images*** command, we then use the - -filter flag to get the specific AMI's we want using the parameters, then we use the region name that was passed in to this script as the output we want from this script.

```
/tmp/demos $ vim latest-amis
/tmp/demos $ chmod +x latest-amis
/tmp/demos $ ./latest-amis
us-west-2
/tmp/demos $ vim latest-amis
/tmp/demos $ ./latest-amis
```


```
            "Public": true,
            "ImageType": "machine",
            "Description": "Amazon Linux AMI 2015.03.1 x86_64 HVM GP2"
        },
        {
            "VirtualizationType": "hvm",
            "Name": "amzn-ami-hvm-2015.03.0.x86_64-gp2",
            "Hypervisor": "xen",
            "ImageOwnerAlias": "amazon",
            "SriovNetSupport": "simple",
            "ImageId": "ami-e7527ed7",
            "State": "available",
            "BlockDeviceMappings": [
                {
                    "DeviceName": "/dev/xvda",
                    "Ebs": {
                        "DeleteOnTermination": true,
                        "SnapshotId": "snap-bfb086e1",
                        "VolumeSize": 8,
                        "VolumeType": "gp2",
                        "Encrypted": false
                    }
                }
            ],
```



We get the requested result that matches all the parameters we passed in. now, we can start seeing how to make this more useful as input into other commands like EC2 instance commands



We can now add the - -query and - -output text flags as above. First, we need to figure out the correct query command we want to use. We want ***tab separated output*** with ***four fields*** of the ***region***, ***image_id***, ***name***, ***description***.

```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
    --region "$region_name" \
    --query "" \
    --output text
}

list_amis "us-west-2"
~
```

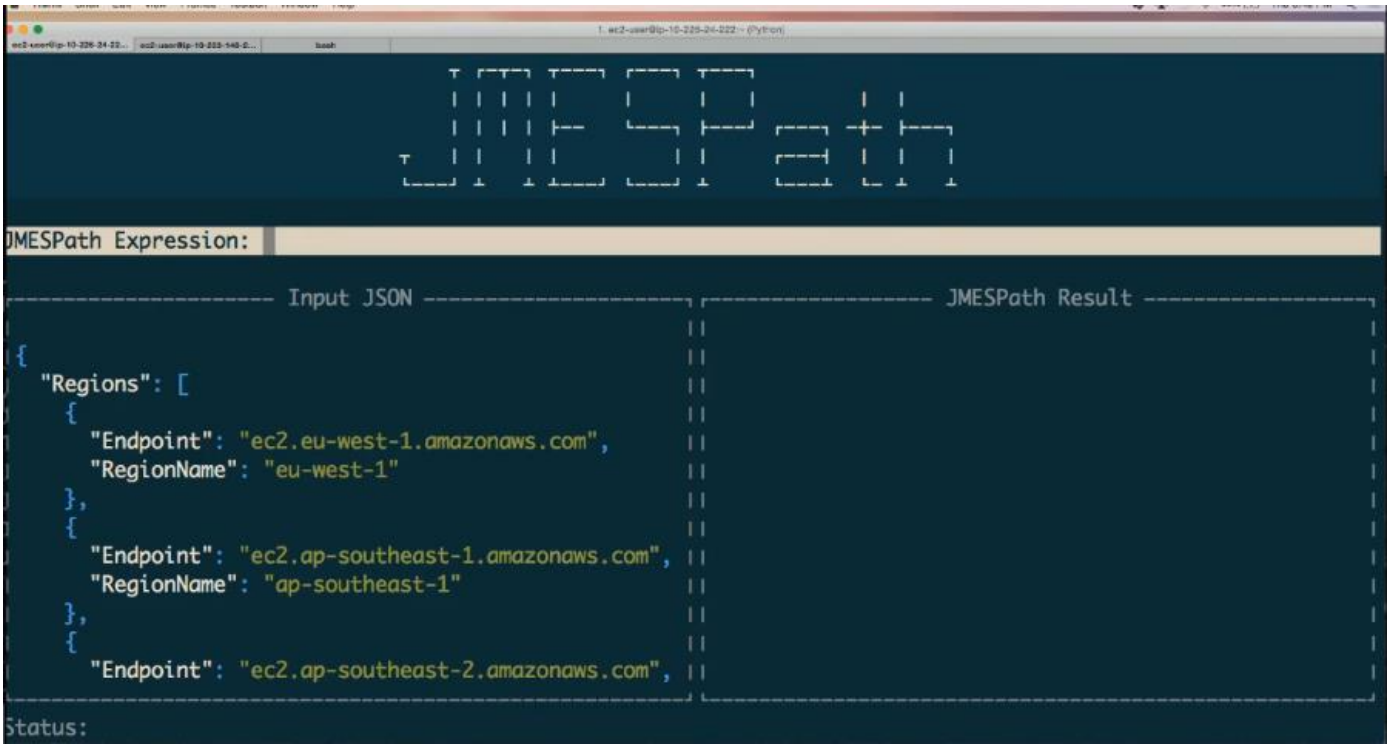
We then need to create a query expression that matches the above value using some tool

```
/tmp/demos $ pip install jmespath-terminal
Requirement already satisfied (use --upgrade to upgrade): jmespath-terminal in /usr/local/lib/python2.7/site-pa
ckages
Requirement already satisfied (use --upgrade to upgrade): urwid==1.2.2 in /usr/local/lib/python2.7/site-package
s (from jmespath-terminal)
Requirement already satisfied (use --upgrade to upgrade): jmespath<=1.0.0,>=0.4.1 in /usr/local/lib/python2.7/s
ite-packages (from jmespath-terminal)
Requirement already satisfied (use --upgrade to upgrade): Pygments<3.0,>=2.0 in /usr/local/lib/python2.7/site-p
ackages (from jmespath-terminal)
/tmp/demos $
```

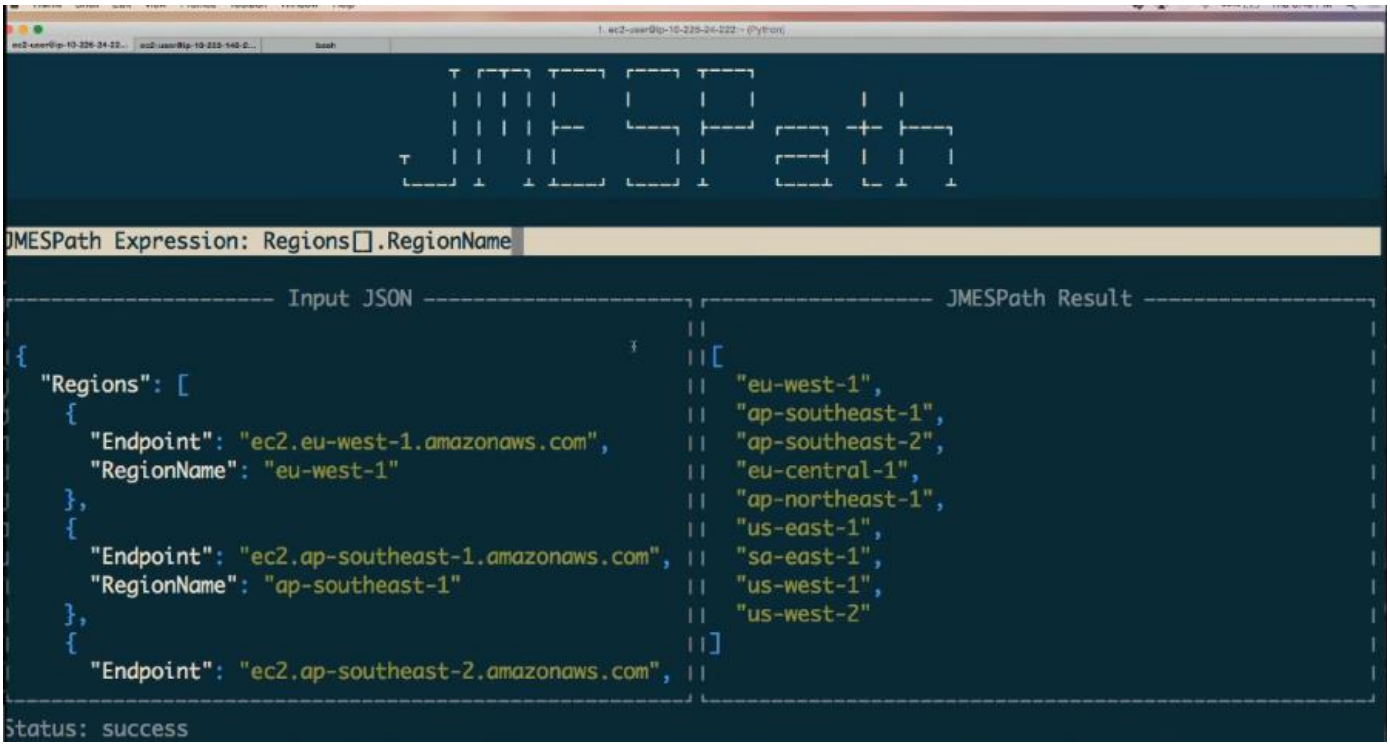
JMESPath is a tool that allows you to take JSON input, manipulate it to fit your query expression. It will give use a terminal where we can manipulate JSON data

```
/tmp/demos $ pip install jmespath-terminal
Requirement already satisfied (use --upgrade to upgrade): jmespath-terminal in /usr/local/lib/python2.7/site-pa
ckages
Requirement already satisfied (use --upgrade to upgrade): urwid==1.2.2 in /usr/local/lib/python2.7/site-package
s (from jmespath-terminal)
Requirement already satisfied (use --upgrade to upgrade): jmespath<=1.0.0,>=0.4.1 in /usr/local/lib/python2.7/s
ite-packages (from jmespath-terminal)
Requirement already satisfied (use --upgrade to upgrade): Pygments<3.0,>=2.0 in /usr/local/lib/python2.7/site-p
ackages (from jmespath-terminal)
/tmp/demos $ aws ec2 describe-regions | jpterm
```

We can pipe the result of an AWS CLI command into jpterm like the command ***$ aws ec2 describe-regions | jpterm*** above



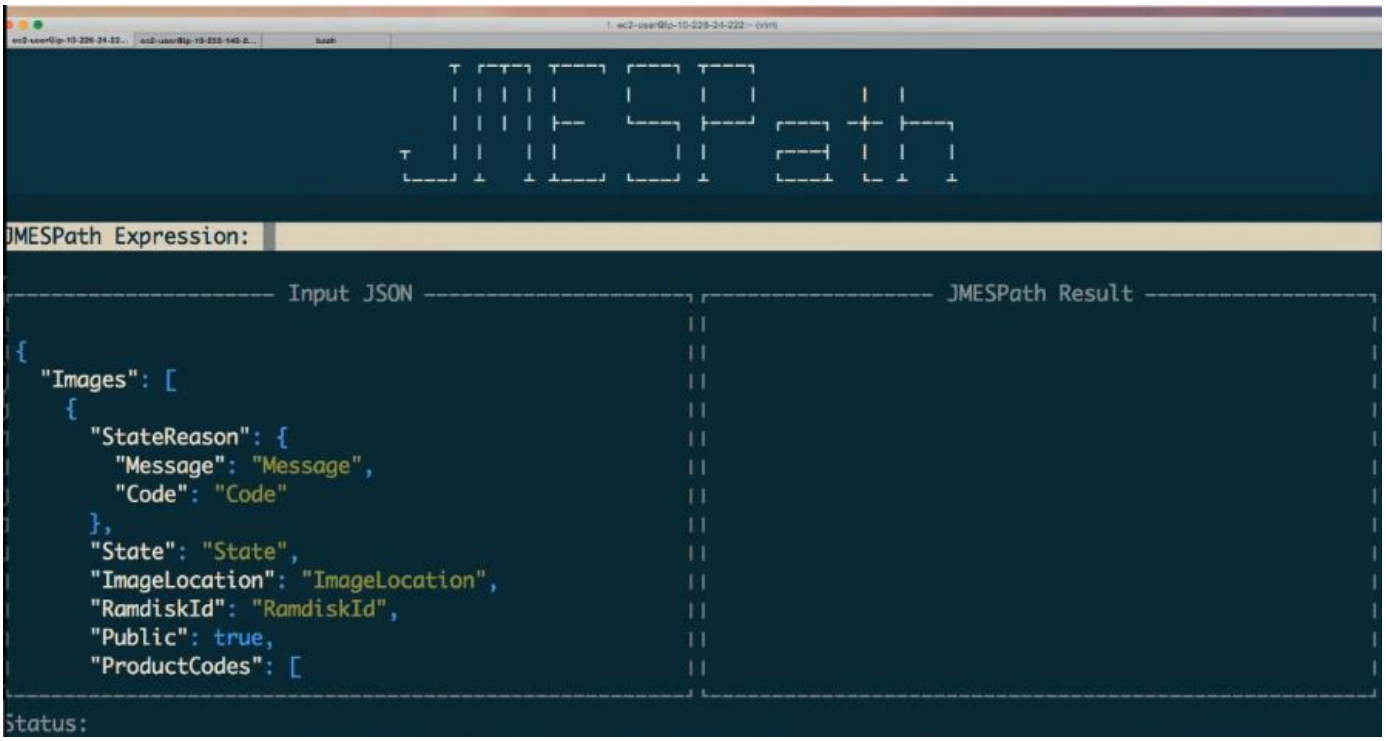
We can now start working with the expression to get the right data shape we want



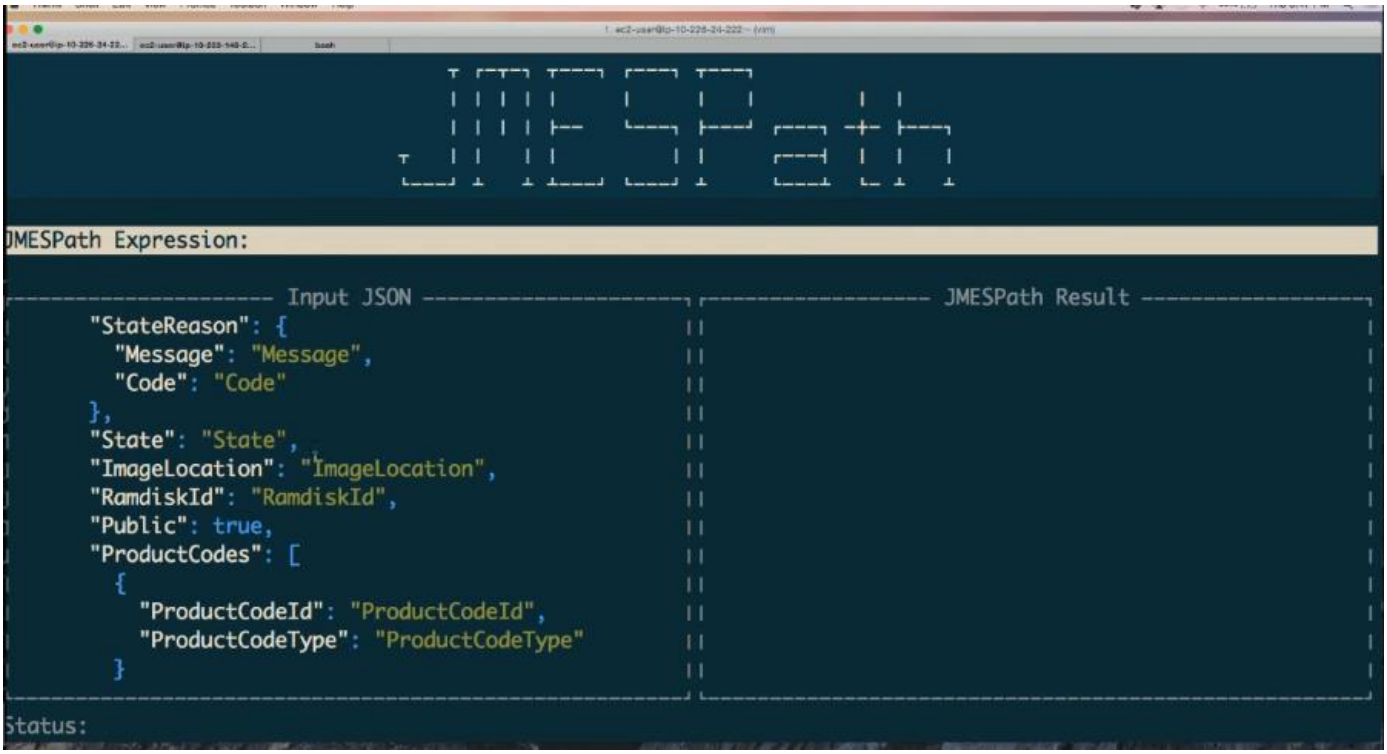
We can write an expression to filter the JSON on the right as above

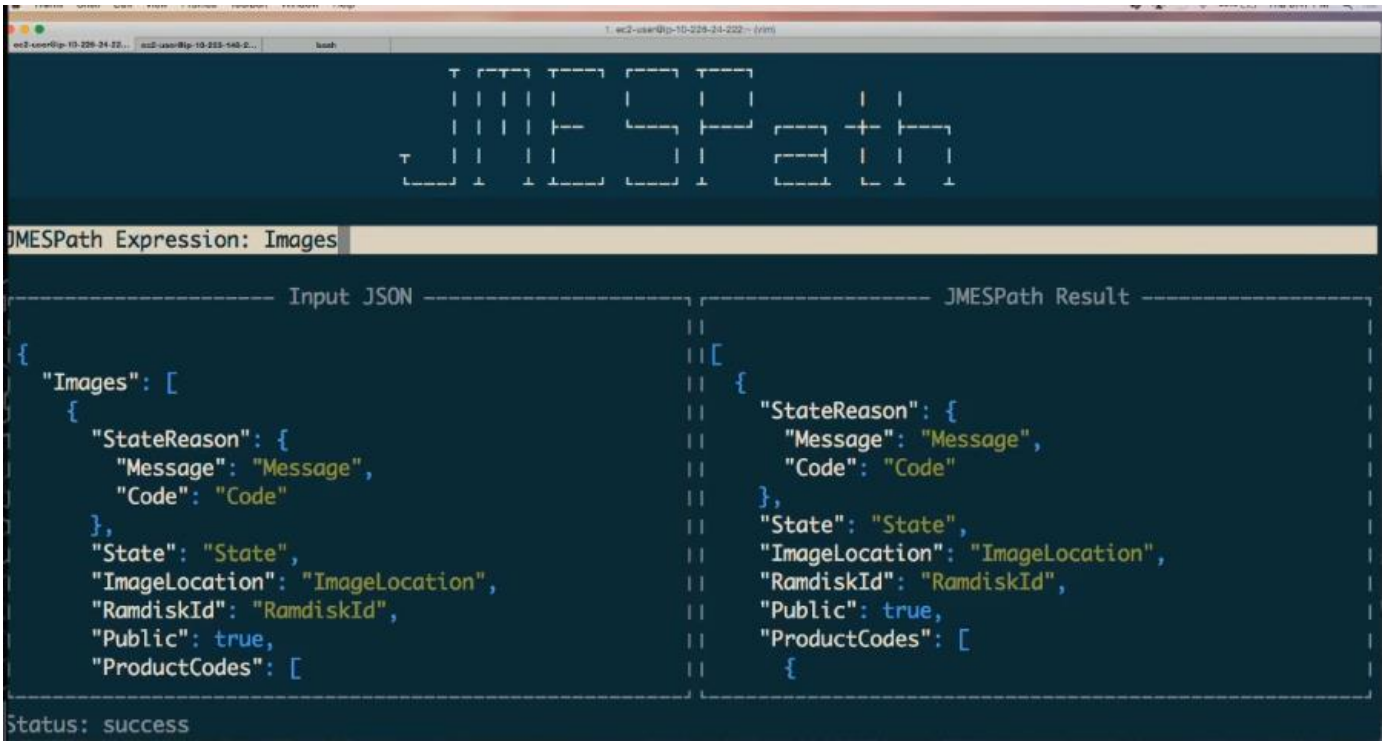
```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "" \
  --output text
}

list_amis "us-west-2"
~
~
```

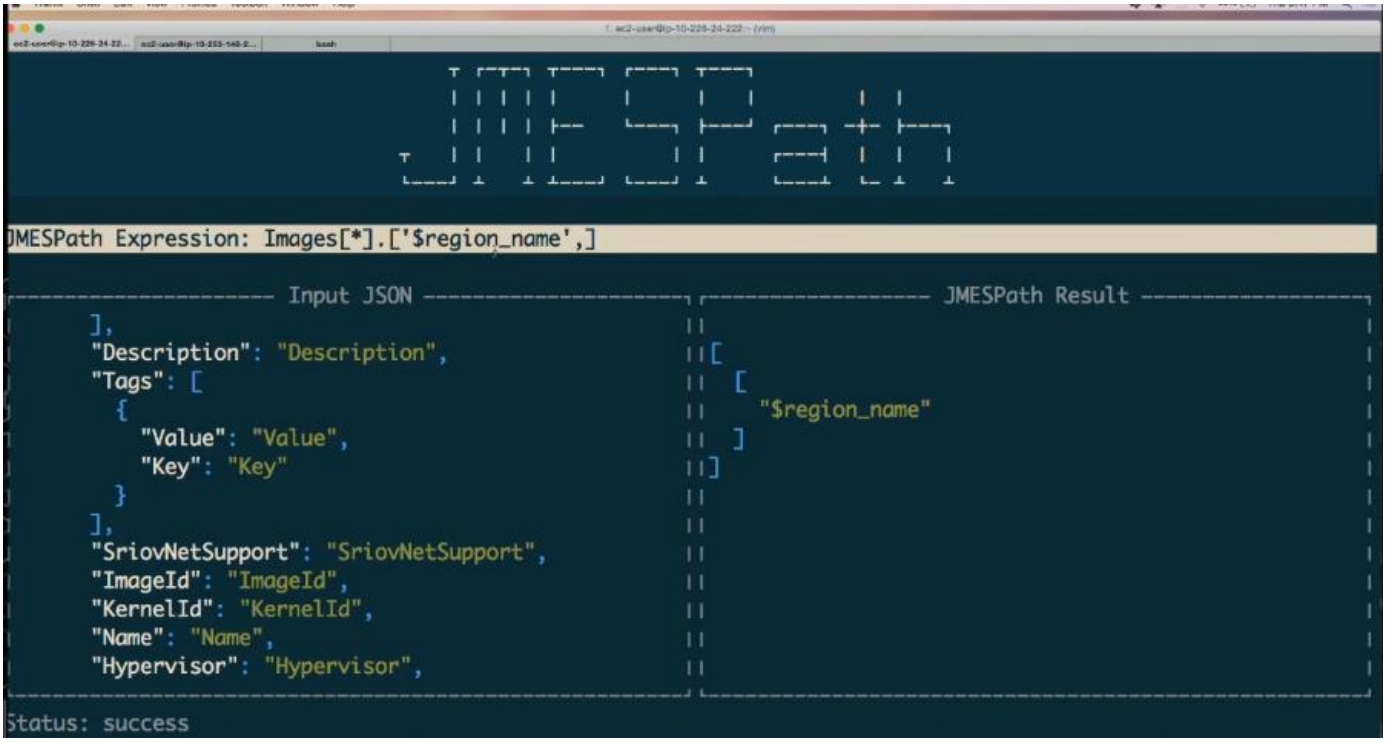
We can use jpterm to generate a sample response for this query by highlighting the query part as above

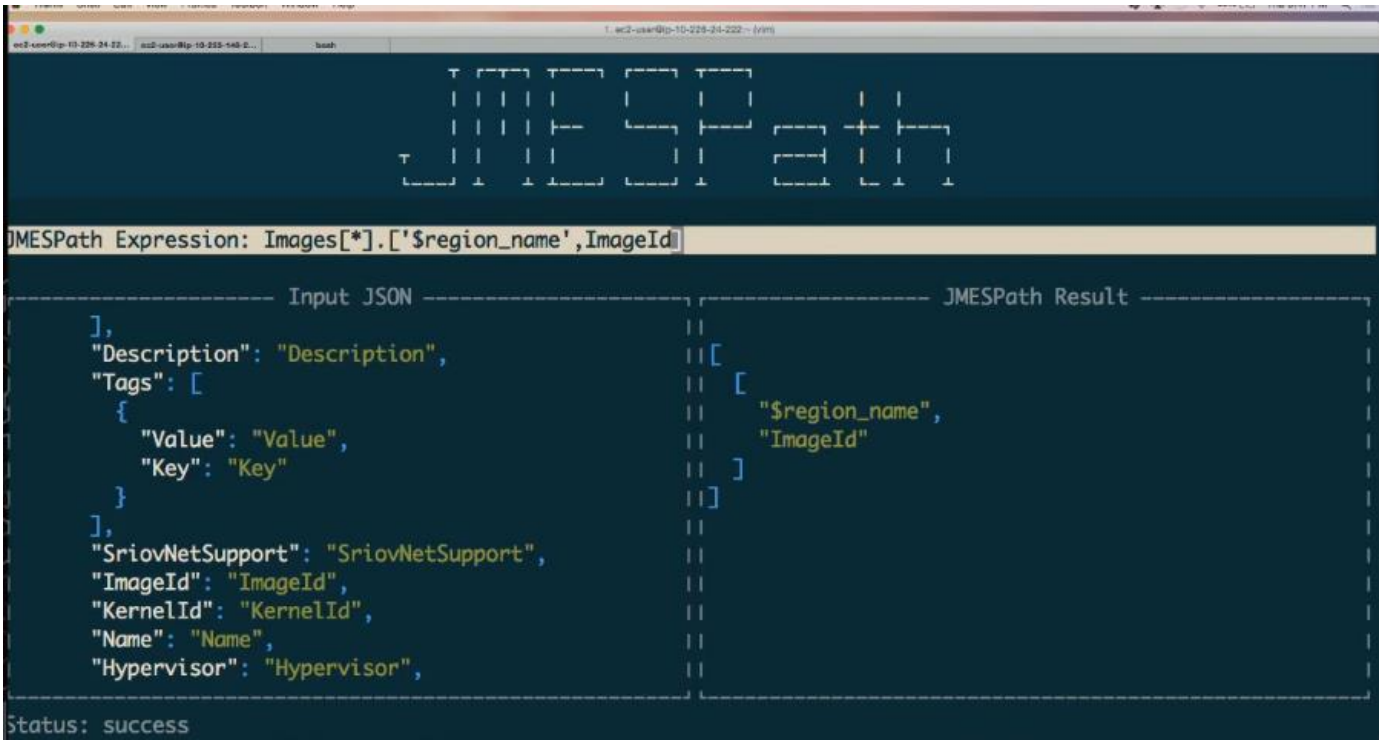


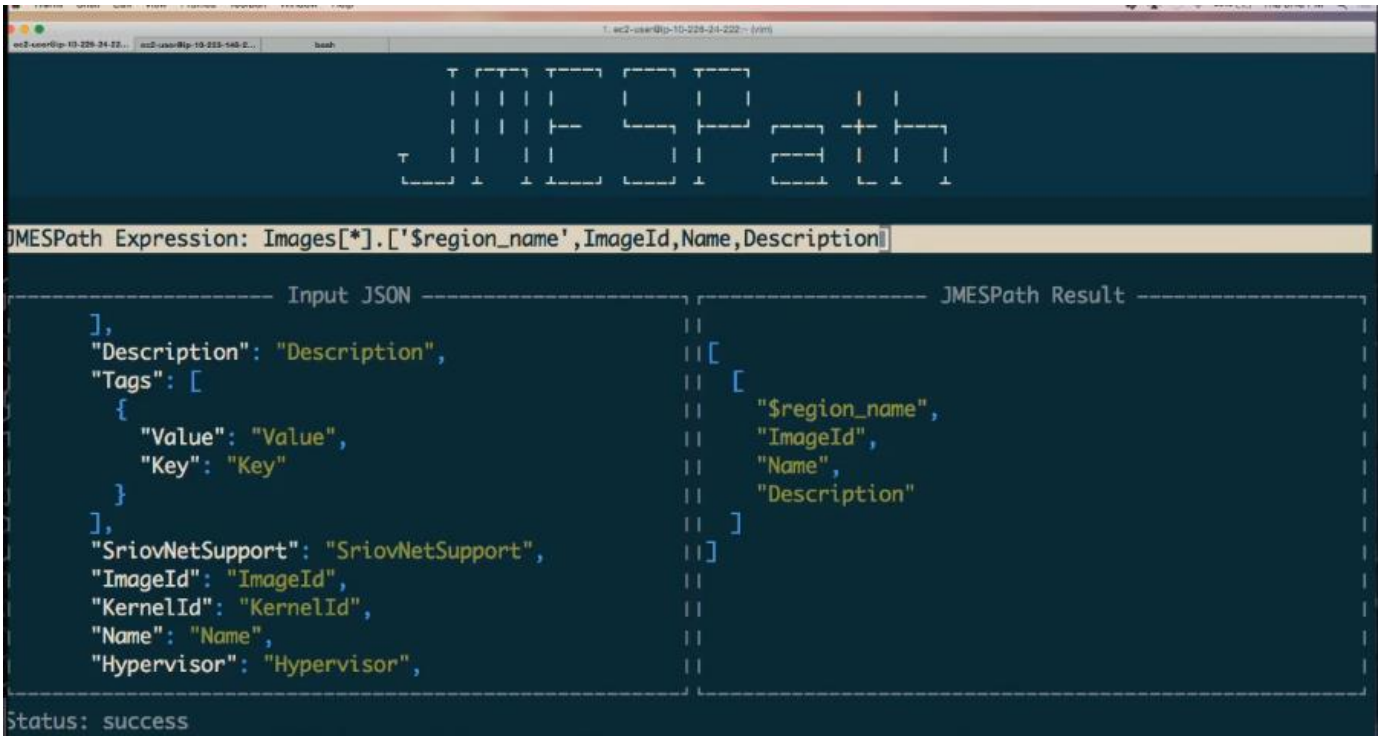
Jpterm will then figure out what the query should be for the highlighted text and generate a JSON that defines all the inputs and outputs as above. Based on that, it will then create what the sample response should be


JMESPath Expression:
Input JSON
"StateReason": {
"Message": "Message",
"Code": "Code"
},
"State": "State",
"ImageLocation": "ImageLocation",
"RamdiskId": "RamdiskId",
"Public": true,
"ProductCodes": [
{
"ProductCodeId": "ProductCodeId",
"ProductCodeType": "ProductCodeType"
}
JMESPath Result
Status:

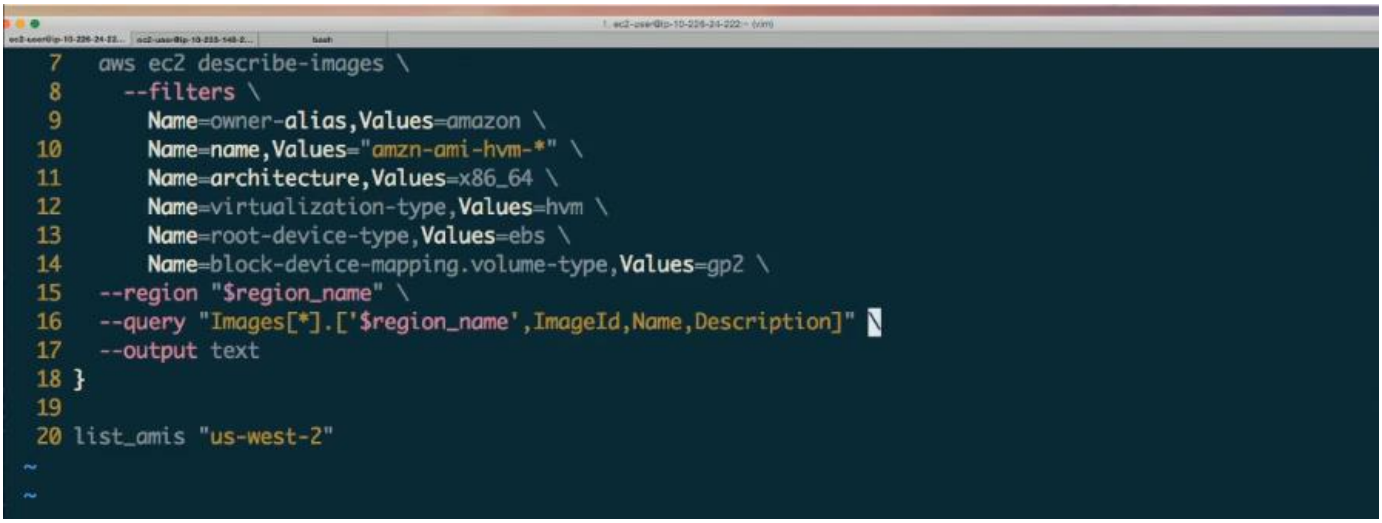

JMESPath Expression: Images
Input JSON
{
"Images": [
{
"StateReason": {
"Message": "Message",
"Code": "Code"
},
"State": "State",
"ImageLocation": "ImageLocation",
"RamdiskId": "RamdiskId",
"Public": true,
"ProductCodes": [
JMESPath Result
[
{
"StateReason": {
"Message": "Message",
"Code": "Code"
},
"State": "State",
"ImageLocation": "ImageLocation",
"RamdiskId": "RamdiskId",
"Public": true,
"ProductCodes": [
{
Status: success


JMESPath Expression: Images[*].['$region_name',]
Input JSON
JMESPath Result
"Description": "Description",
"Tags": [
{
"Value": "Value",
"Key": "Key"
}
],
"SriovNetSupport": "SriovNetSupport",
"ImageId": "ImageId",
"KernelId": "KernelId",
"Name": "Name",
"Hypervisor": "Hypervisor",
[
[
"$region_name"
]
]
Status: success


JMESPath Expression: Images[*].['$region_name',ImageId]
Input JSON
JMESPath Result
"Description": "Description",
"Tags": [
{
"Value": "Value",
"Key": "Key"
}
],
"SriovNetSupport": "SriovNetSupport",
"ImageId": "ImageId",
"KernelId": "KernelId",
"Name": "Name",
"Hypervisor": "Hypervisor",
[
[
"$region_name",
"ImageId"
]
]
Status: success



We now have the data that we want displayed on the right side as a list of list,



Once we exit out of ***jpterm***, it fills in the query for us in the shell script that was highlighted. This is a way to create those shell scripts that we need without making any AWS API calls.

```
/tmp/demos $ ./latest-amis
us-west-2       ami-35d5c805    amzn-ami-hvm-2015.09.rc-1.x86_64-gp2    Amazon Linux AMI 2015.09.rc-1 x86_64 HV
M GP2
us-west-2       ami-61a6b151    amzn-ami-hvm-2015.09.rc-0.x86_64-gp2    Amazon Linux AMI 2015.09.rc-0 x86_64 HV
M GP2
us-west-2       ami-8586c6b5    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM G
P2
us-west-2       ami-8f6815bf    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM G
P2
us-west-2       ami-9ff7e8af    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM G
P2
us-west-2       ami-b1a7ea81    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM G
P2
us-west-2       ami-c1c39af1    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM G
P2
us-west-2       ami-d5c5d1e5    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM G
P2
us-west-2       ami-e7527ed7    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM G
P2
/tmp/demos $
```

We get the tab separated outputs that we wanted from the query

```
[1]+  Stopped                 vim latest-amis
/tmp/demos $ clear
/tmp/demos $ ./latest-amis
us-west-2       ami-35d5c805    amzn-ami-hvm-2015.09.rc-1.x86_64-gp2    Amazon Linux AMI 2015.09.rc-1 x86_64 HVM GP2
us-west-2       ami-61a6b151    amzn-ami-hvm-2015.09.rc-0.x86_64-gp2    Amazon Linux AMI 2015.09.rc-0 x86_64 HVM GP2
us-west-2       ami-8586c6b5    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
us-west-2       ami-8f6815bf    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
us-west-2       ami-9ff7e8af    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
us-west-2       ami-b1a7ea81    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
us-west-2       ami-c1c39af1    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
us-west-2       ami-d5c5d1e5    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
us-west-2       ami-e7527ed7    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
/tmp/demos $ ./latest-amis
us-west-2       ami-35d5c805    amzn-ami-hvm-2015.09.rc-1.x86_64-gp2    Amazon Linux AMI 2015.09.rc-1 x86_64 HVM GP2
us-west-2       ami-61a6b151    amzn-ami-hvm-2015.09.rc-0.x86_64-gp2    Amazon Linux AMI 2015.09.rc-0 x86_64 HVM GP2
us-west-2       ami-8586c6b5    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
us-west-2       ami-8f6815bf    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
us-west-2       ami-9ff7e8af    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
us-west-2       ami-b1a7ea81    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
us-west-2       ami-c1c39af1    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
us-west-2       ami-d5c5d1e5    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
us-west-2       ami-e7527ed7    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
/tmp/demos $
```

We can sort this list using the year, filter out the rc-* ones, then get the latest version of the AMIs to use later

```
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "Images[*].['$region_name',ImageId,Name,Description]" \
  --output text
}

list_amis "us-west-2"
~
~
```

```
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "sort_by(Images, &CreationDate)[*].['$region_name',ImageId,Name,Description]" \
  --output text
}

list_amis "us-west-2"
```

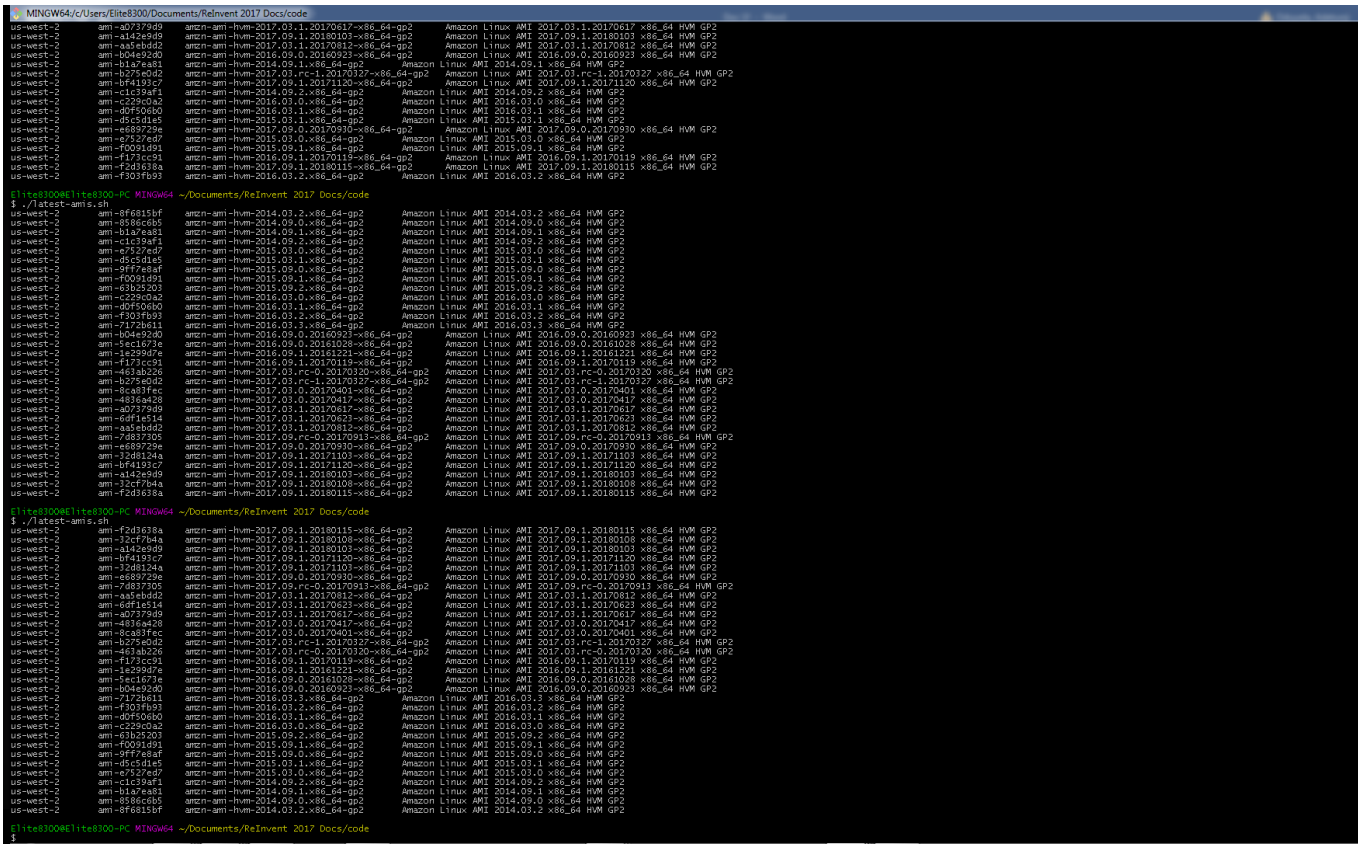
We first add the text to sort the result by the Images and the CreationDate using the query flag ***- -query "sort_by(Images, &CreationDate)[*].['$region_name',ImageId,Name,Description]"***

```
Elite8300@Elite8300-PC MINGW64 ~/Documents/ReInvent 2017 Docs/code
$ ./latest-amis.sh
us-west-2       ami-1e299d7e    amzn-ami-hvm-2016.09.1.20161221-x86_64-gp2      Amazon Linux AMI 2016.09.1.20161221 x86_64 HVM GP2
us-west-2       ami-32cf7b4a    amzn-ami-hvm-2017.09.1.20180108-x86_64-gp2      Amazon Linux AMI 2017.09.1.20180108 x86_64 HVM GP2
us-west-2       ami-32d8124a    amzn-ami-hvm-2017.09.1.20171103-x86_64-gp2      Amazon Linux AMI 2017.09.1.20171103 x86_64 HVM GP2
us-west-2       ami-463ab226    amzn-ami-hvm-2017.03.rc-0.20170320-x86_64-gp2   Amazon Linux AMI 2017.03.rc-0.20170320 x86_64 HVM GP2
us-west-2       ami-4836a428    amzn-ami-hvm-2017.03.0.20170417-x86_64-gp2      Amazon Linux AMI 2017.03.0.20170417 x86_64 HVM GP2
us-west-2       ami-5ec1673e    amzn-ami-hvm-2016.09.0.20161028-x86_64-gp2      Amazon Linux AMI 2016.09.0.20161028 x86_64 HVM GP2
us-west-2       ami-63b25203    amzn-ami-hvm-2015.09.2.x86_64-gp2       Amazon Linux AMI 2015.09.2 x86_64 HVM GP2
us-west-2       ami-6df1e514    amzn-ami-hvm-2017.03.1.20170623-x86_64-gp2      Amazon Linux AMI 2017.03.1.20170623 x86_64 HVM GP2
us-west-2       ami-7172b611    amzn-ami-hvm-2016.03.3.x86_64-gp2       Amazon Linux AMI 2016.03.3 x86_64 HVM GP2
us-west-2       ami-7d837305    amzn-ami-hvm-2017.09.rc-0.20170913-x86_64-gp2   Amazon Linux AMI 2017.09.rc-0.20170913 x86_64 HVM GP2
us-west-2       ami-8586c6b5    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
us-west-2       ami-8ca83fec    amzn-ami-hvm-2017.03.0.20170401-x86_64-gp2      Amazon Linux AMI 2017.03.0.20170401 x86_64 HVM GP2
us-west-2       ami-8f6815bf    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
us-west-2       ami-9ff7e8af    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
us-west-2       ami-a07379d9    amzn-ami-hvm-2017.03.1.20170617-x86_64-gp2      Amazon Linux AMI 2017.03.1.20170617 x86_64 HVM GP2
us-west-2       ami-a142e9d9    amzn-ami-hvm-2017.09.1.20180103-x86_64-gp2      Amazon Linux AMI 2017.09.1.20180103 x86_64 HVM GP2
us-west-2       ami-aa5ebdd2    amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2      Amazon Linux AMI 2017.03.1.20170812 x86_64 HVM GP2
us-west-2       ami-b04e92d0    amzn-ami-hvm-2016.09.0.20160923-x86_64-gp2      Amazon Linux AMI 2016.09.0.20160923 x86_64 HVM GP2
us-west-2       ami-b1a7ea81    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
us-west-2       ami-b27e0d2     amzn-ami-hvm-2017.03.rc-1.20170327-x86_64-gp2   Amazon Linux AMI 2017.03.rc-1.20170327 x86_64 HVM GP2
us-west-2       ami-bf4193c7    amzn-ami-hvm-2017.09.1.20171120-x86_64-gp2      Amazon Linux AMI 2017.09.1.20171120 x86_64 HVM GP2
us-west-2       ami-c1c39af1    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
us-west-2       ami-c229c0a2    amzn-ami-hvm-2016.03.0.x86_64-gp2       Amazon Linux AMI 2016.03.0 x86_64 HVM GP2
us-west-2       ami-d0f506b0    amzn-ami-hvm-2016.03.1.x86_64-gp2       Amazon Linux AMI 2016.03.1 x86_64 HVM GP2
us-west-2       ami-d5c5d1e5    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
us-west-2       ami-e689729e    amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2      Amazon Linux AMI 2017.09.0.20170930 x86_64 HVM GP2
us-west-2       ami-e7527ed7    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
us-west-2       ami-f0091d91    amzn-ami-hvm-2015.09.1.x86_64-gp2       Amazon Linux AMI 2015.09.1 x86_64 HVM GP2
us-west-2       ami-f173cc91    amzn-ami-hvm-2016.09.1.20170119-x86_64-gp2      Amazon Linux AMI 2016.09.1.20170119 x86_64 HVM GP2
us-west-2       ami-f2d3638a    amzn-ami-hvm-2017.09.1.20180115-x86_64-gp2      Amazon Linux AMI 2017.09.1.20180115 x86_64 HVM GP2
us-west-2       ami-f303fb93    amzn-ami-hvm-2016.03.2.x86_64-gp2       Amazon Linux AMI 2016.03.2 x86_64 HVM GP2

Elite8300@Elite8300-PC MINGW64 ~/Documents/ReInvent 2017 Docs/code
$ ./latest-amis.sh
us-west-2       ami-8f6815bf    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
us-west-2       ami-8586c6b5    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
us-west-2       ami-b1a7ea81    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
us-west-2       ami-c1c39af1    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
us-west-2       ami-e7527ed7    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
us-west-2       ami-d5c5d1e5    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
us-west-2       ami-9ff7e8af    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
us-west-2       ami-f0091d91    amzn-ami-hvm-2015.09.1.x86_64-gp2       Amazon Linux AMI 2015.09.1 x86_64 HVM GP2
us-west-2       ami-63b25203    amzn-ami-hvm-2015.09.2.x86_64-gp2       Amazon Linux AMI 2015.09.2 x86_64 HVM GP2
us-west-2       ami-c229c0a2    amzn-ami-hvm-2016.03.0.x86_64-gp2       Amazon Linux AMI 2016.03.0 x86_64 HVM GP2
us-west-2       ami-d0f506b0    amzn-ami-hvm-2016.03.1.x86_64-gp2       Amazon Linux AMI 2016.03.1 x86_64 HVM GP2
us-west-2       ami-f303fb93    amzn-ami-hvm-2016.03.2.x86_64-gp2       Amazon Linux AMI 2016.03.2 x86_64 HVM GP2
us-west-2       ami-7172b611    amzn-ami-hvm-2016.03.3.x86_64-gp2       Amazon Linux AMI 2016.03.3 x86_64 HVM GP2
us-west-2       ami-b04e92d0    amzn-ami-hvm-2016.09.0.20160923-x86_64-gp2      Amazon Linux AMI 2016.09.0.20160923 x86_64 HVM GP2
us-west-2       ami-5ec1673e    amzn-ami-hvm-2016.09.0.20161028-x86_64-gp2      Amazon Linux AMI 2016.09.0.20161028 x86_64 HVM GP2
us-west-2       ami-1e299d7e    amzn-ami-hvm-2016.09.1.20161221-x86_64-gp2      Amazon Linux AMI 2016.09.1.20161221 x86_64 HVM GP2
us-west-2       ami-f173cc91    amzn-ami-hvm-2016.09.1.20170119-x86_64-gp2      Amazon Linux AMI 2016.09.1.20170119 x86_64 HVM GP2
us-west-2       ami-463ab226    amzn-ami-hvm-2017.03.rc-0.20170320-x86_64-gp2   Amazon Linux AMI 2017.03.rc-0.20170320 x86_64 HVM GP2
us-west-2       ami-b27e0d2     amzn-ami-hvm-2017.03.rc-1.20170327-x86_64-gp2   Amazon Linux AMI 2017.03.rc-1.20170327 x86_64 HVM GP2
us-west-2       ami-8ca83fec    amzn-ami-hvm-2017.03.0.20170401-x86_64-gp2      Amazon Linux AMI 2017.03.0.20170401 x86_64 HVM GP2
us-west-2       ami-4836a428    amzn-ami-hvm-2017.03.0.20170417-x86_64-gp2      Amazon Linux AMI 2017.03.0.20170417 x86_64 HVM GP2
us-west-2       ami-a07379d9    amzn-ami-hvm-2017.03.1.20170617-x86_64-gp2      Amazon Linux AMI 2017.03.1.20170617 x86_64 HVM GP2
us-west-2       ami-6df1e514    amzn-ami-hvm-2017.03.1.20170623-x86_64-gp2      Amazon Linux AMI 2017.03.1.20170623 x86_64 HVM GP2
us-west-2       ami-aa5ebdd2    amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2      Amazon Linux AMI 2017.03.1.20170812 x86_64 HVM GP2
us-west-2       ami-7d837305    amzn-ami-hvm-2017.09.rc-0.20170913-x86_64-gp2   Amazon Linux AMI 2017.09.rc-0.20170913 x86_64 HVM GP2
us-west-2       ami-e689729e    amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2      Amazon Linux AMI 2017.09.0.20170930 x86_64 HVM GP2
us-west-2       ami-32d8124a    amzn-ami-hvm-2017.09.1.20171103-x86_64-gp2      Amazon Linux AMI 2017.09.1.20171103 x86_64 HVM GP2
us-west-2       ami-bf4193c7    amzn-ami-hvm-2017.09.1.20171120-x86_64-gp2      Amazon Linux AMI 2017.09.1.20171120 x86_64 HVM GP2
us-west-2       ami-a142e9d9    amzn-ami-hvm-2017.09.1.20180103-x86_64-gp2      Amazon Linux AMI 2017.09.1.20180103 x86_64 HVM GP2
us-west-2       ami-32cf7b4a    amzn-ami-hvm-2017.09.1.20180108-x86_64-gp2      Amazon Linux AMI 2017.09.1.20180108 x86_64 HVM GP2
us-west-2       ami-f2d3638a    amzn-ami-hvm-2017.09.1.20180115-x86_64-gp2      Amazon Linux AMI 2017.09.1.20180115 x86_64 HVM GP2

Elite8300@Elite8300-PC MINGW64 ~/Documents/ReInvent 2017 Docs/code
$
```

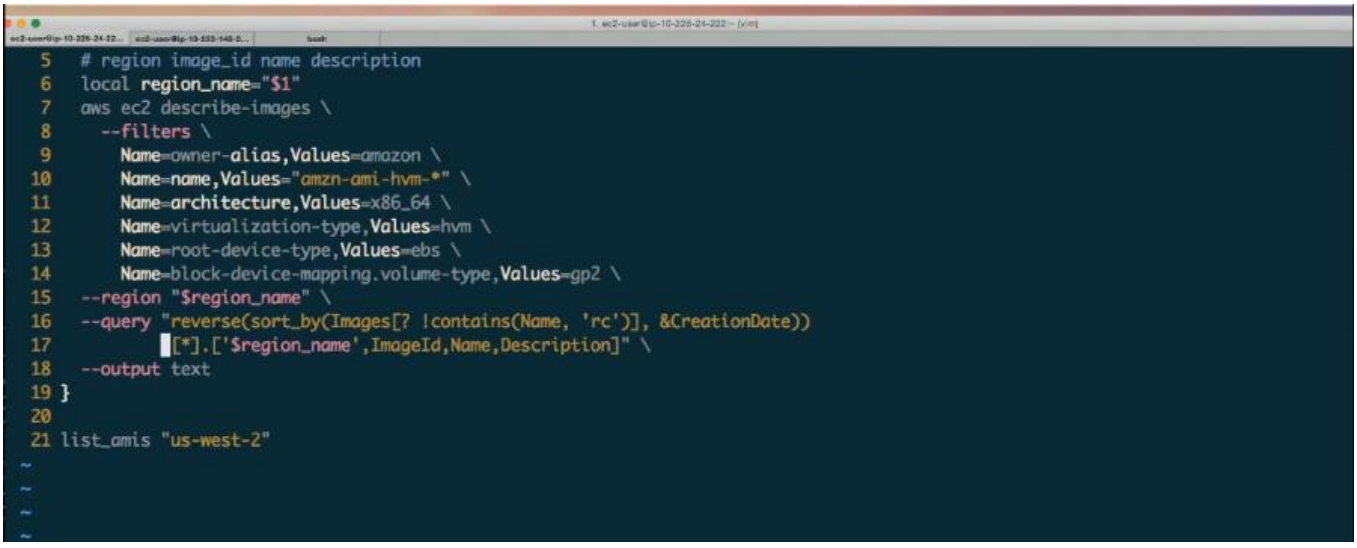
We get the images sorted by the Images and then by ***CreationDate*** as above

```
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images, &CreationDate))[*].['$region_name',ImageId,Name,Description]" \
  --output text
}

list_amis "us-west-2"
```

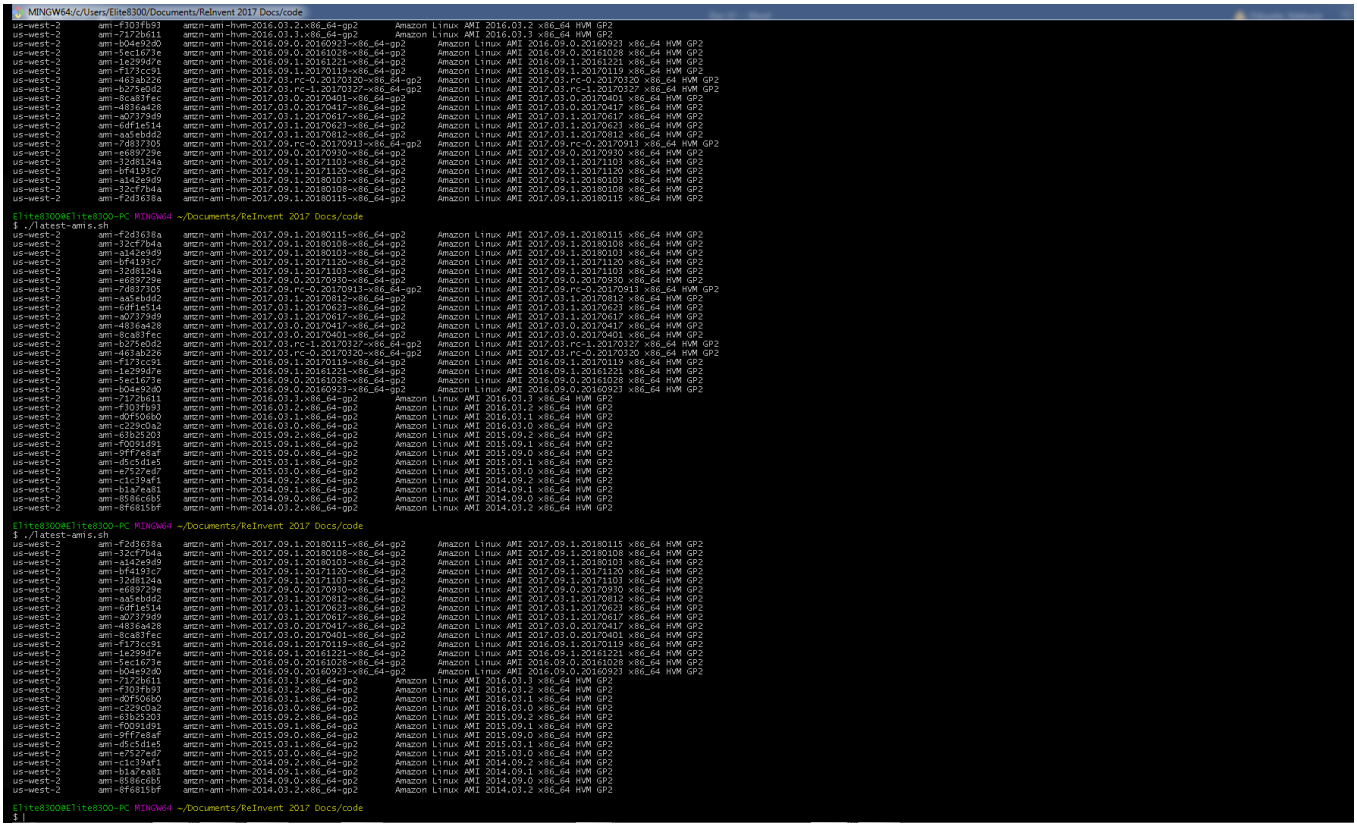
Then we add the text to reverse the result using the query flag ***- -query "reverse(sort_by(Images, &CreationDate))[*].['$region_name',ImageId,Name,Description]"***

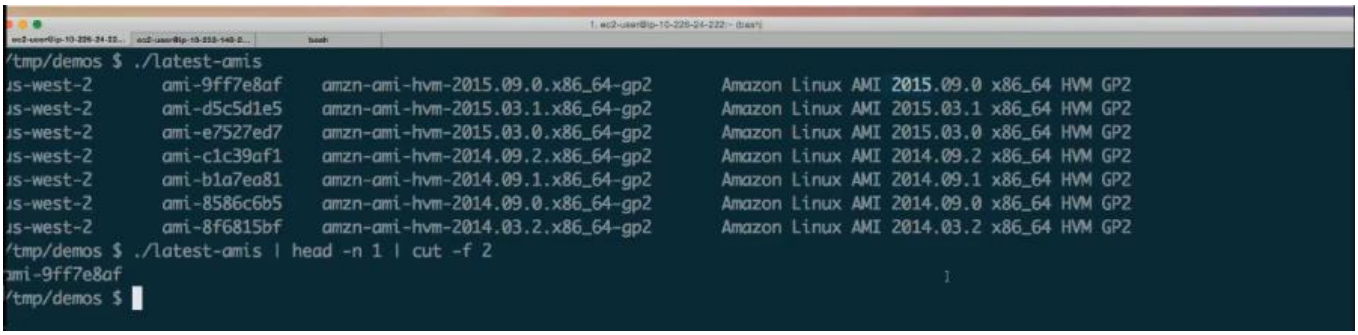


We have now ***reversed*** the order of the results and now showing the ***most recent AMIs*** first in the result as above.

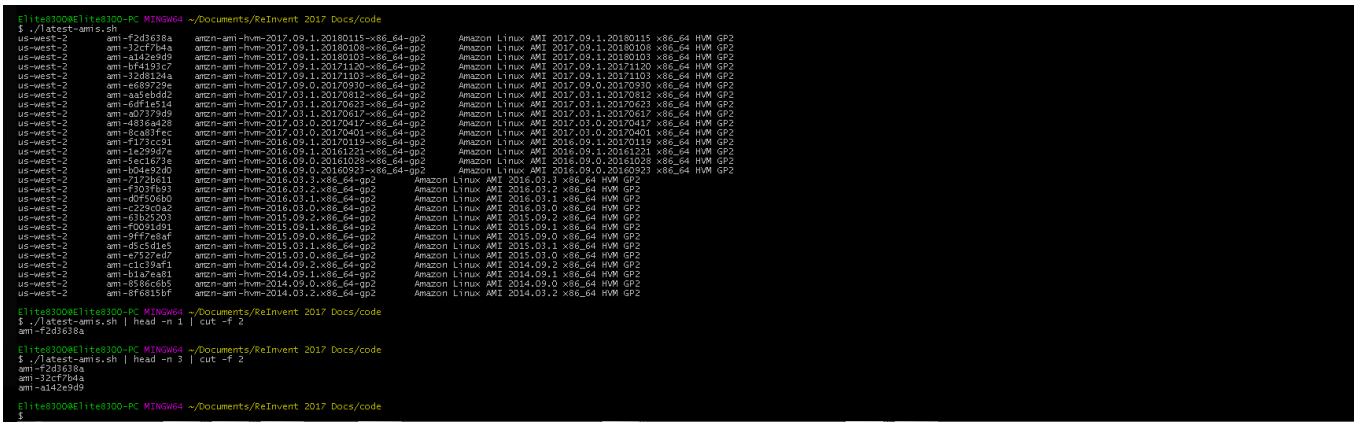


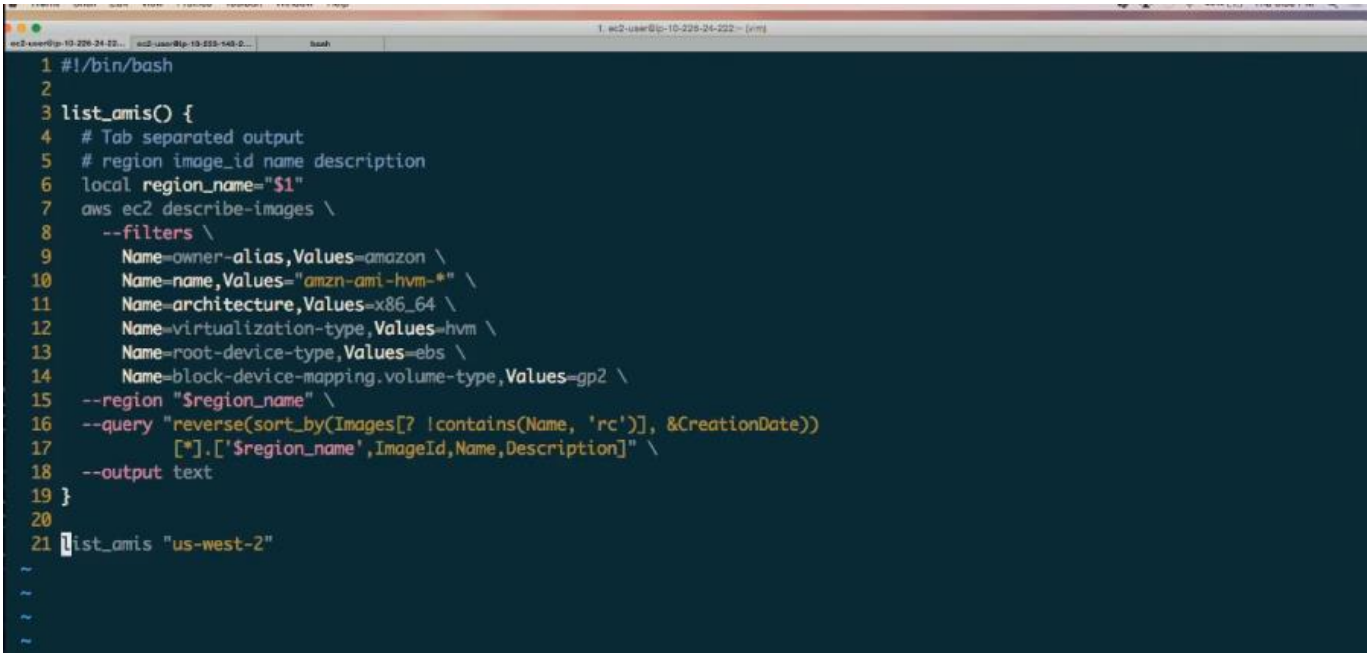
We can then filter out any image with the ***rc-**** in their name, we use a filter expression ***Images[? !contains(Name, 'rc')]*** for this as above





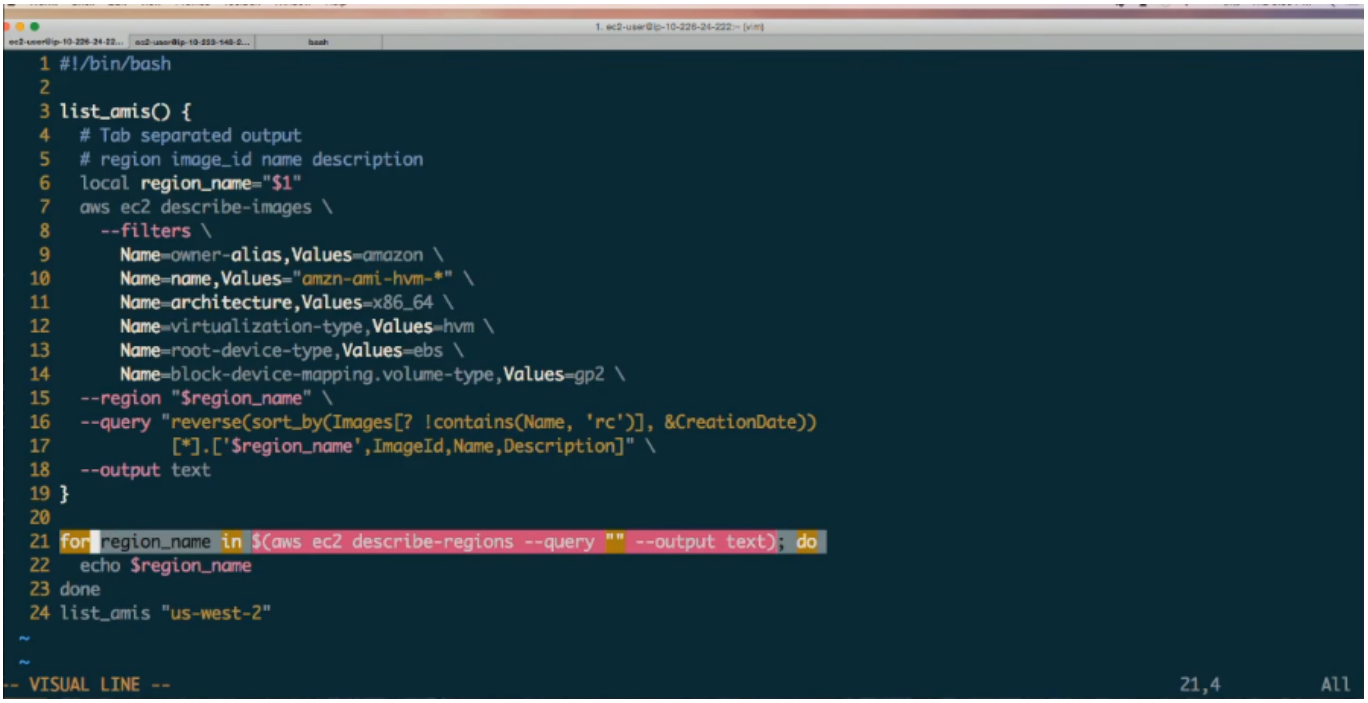
We can then pipe the result of this script into an expression to take the first AMI in the list and retun the AMI ID for it as above





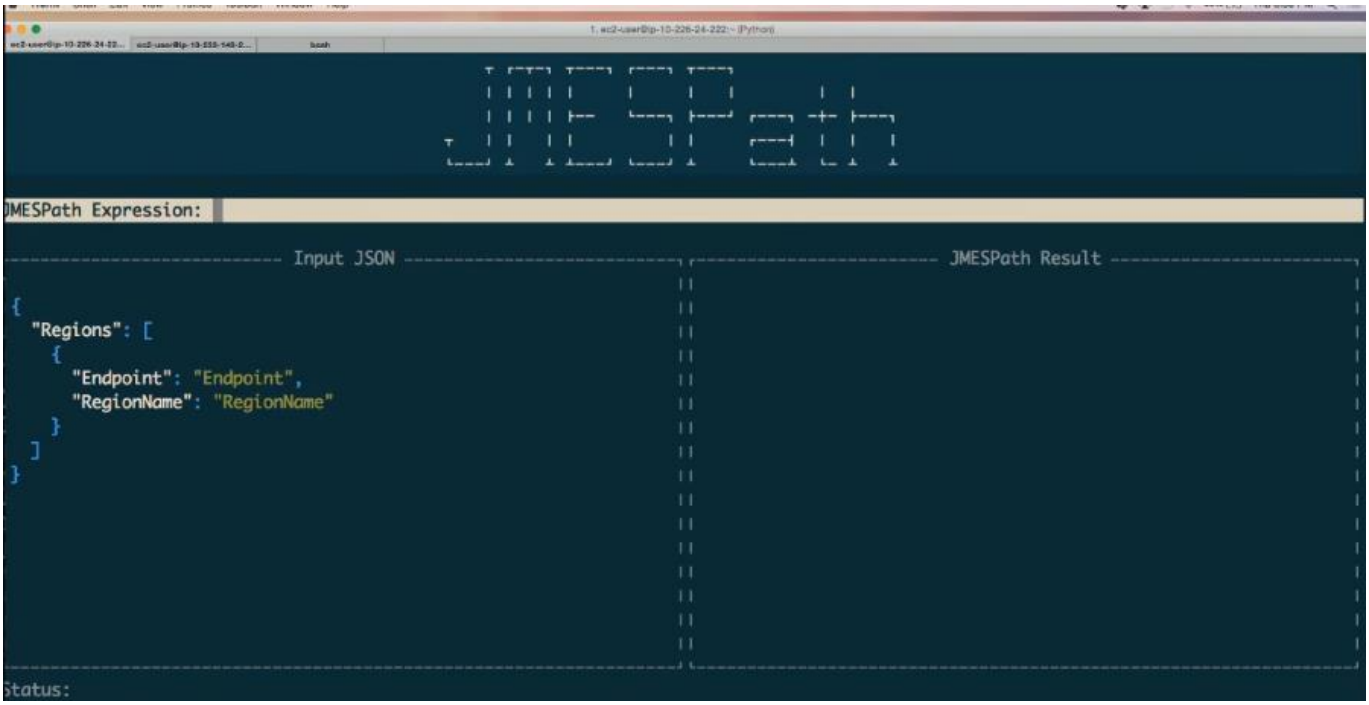
```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images[? !contains(Name, 'rc')], &CreationDate))
           [*].['$region_name',ImageId,Name,Description]" \
  --output text
}

list_amis "us-west-2"
```

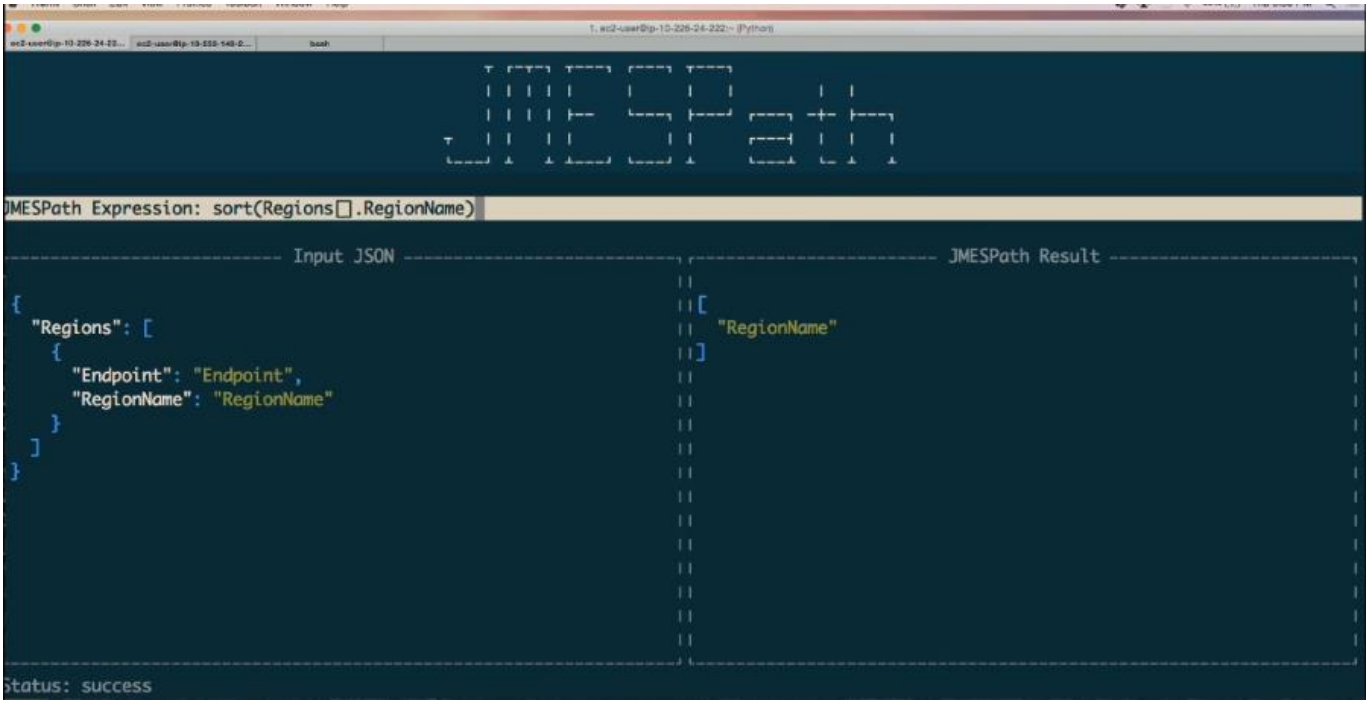
The script above gives us the AMI's filtered for a single region ***us-west-2***, we can use this script to see the latest AMIs for all regions using a for loop as below



```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images[? !contains(Name, 'rc')], &CreationDate))
           [*].['$region_name',ImageId,Name,Description]" \
  --output text
}

for region_name in $(aws ec2 describe-regions --query "" --output text); do
  echo $region_name
done
list_amis "us-west-2"
-- VISUAL LINE --                                                    21,4          All
```

In the for loop, we leave the - -query empty in the query expression so that we can use JMESPath again to help us with what the query should be. We then simply plan to echo out the region names for now





This query has the data that we want, so we exit and have JMESPath insert the query expression into our script as below

```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images[? !contains(Name, 'rc')], &CreationDate))
           [*].['$region_name',ImageId,Name,Description]" \
  --output text
}

for region_name in $(aws ec2 describe-regions --query "sort(Regions[].RegionName)" --output text); do
  echo $region_name
done
list_amis "us-west-2"
~
~
:'<,'>py send_to_jp()                                                    21,1          All
```

Now we can run this command to see the list of region names

```
/tmp/demos $ ./latest-amis
ap-northeast-1
ap-southeast-1
ap-southeast-2
eu-central-1
eu-west-1
sa-east-1
us-east-1
us-west-1
us-west-2
/tmp/demos $
```

We get the list of regions as expected

```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images[? !contains(Name, 'rc')], &CreationDate))
           [*].['$region_name',ImageId,Name,Description]" \
  --output text
}

for region_name in $(aws ec2 describe-regions --query "sort(Regions[].RegionName)" --output text); do
  list_amis "$region_name"
done
~
```

We replace the query with the echo with our script function ***list_amis "$region_name"*** to be run within the loop so that we can now have a list of all the latest AMIs in each available region.

```
ap-northeast-1
ap-southeast-1
ap-southeast-2
eu-central-1
eu-west-1
sa-east-1
us-east-1
us-west-1
us-west-2
/tmp/demos $ fg
/im latest-amis
/tmp/demos $ ./latest-amis
ap-northeast-1  ami-9a2fb89a  amzn-ami-hvm-2015.09.0.x86_64-gp2  Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
ap-northeast-1  ami-1c1b9f1c  amzn-ami-hvm-2015.03.1.x86_64-gp2  Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
ap-northeast-1  ami-cbf90ecb  amzn-ami-hvm-2015.03.0.x86_64-gp2  Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
ap-northeast-1  ami-1e86981f  amzn-ami-hvm-2014.09.2.x86_64-gp2  Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
ap-northeast-1  ami-4585b044  amzn-ami-hvm-2014.09.1.x86_64-gp2  Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
ap-northeast-1  ami-45072844  amzn-ami-hvm-2014.09.0.x86_64-gp2  Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
ap-northeast-1  ami-df470ede  amzn-ami-hvm-2014.03.2.x86_64-gp2  Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
ap-southeast-1  ami-52978200  amzn-ami-hvm-2015.09.0.x86_64-gp2  Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
ap-southeast-1  ami-d44b4286  amzn-ami-hvm-2015.03.1.x86_64-gp2  Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
ap-southeast-1  ami-68d8e93a  amzn-ami-hvm-2015.03.0.x86_64-gp2  Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
ap-southeast-1  ami-94bb90c6  amzn-ami-hvm-2014.09.2.x86_64-gp2  Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
ap-southeast-1  ami-525d7b00  amzn-ami-hvm-2014.09.1.x86_64-gp2  Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
ap-southeast-1  ami-d2e1c580  amzn-ami-hvm-2014.09.0.x86_64-gp2  Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
ap-southeast-1  ami-82d78bd0  amzn-ami-hvm-2014.03.2.x86_64-gp2  Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
```

```
ap-southeast-1  ami-68d8e93a  amzn-ami-hvm-2015.03.0.x86_64-gp2  Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
ap-southeast-1  ami-94bb90c6  amzn-ami-hvm-2014.09.2.x86_64-gp2  Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
ap-southeast-1  ami-525d7b00  amzn-ami-hvm-2014.09.1.x86_64-gp2  Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
ap-southeast-1  ami-d2e1c580  amzn-ami-hvm-2014.09.0.x86_64-gp2  Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
ap-southeast-1  ami-82d78bd0  amzn-ami-hvm-2014.03.2.x86_64-gp2  Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
ap-southeast-2  ami-c11856fb  amzn-ami-hvm-2015.09.0.x86_64-gp2  Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
ap-southeast-2  ami-db7b39e1  amzn-ami-hvm-2015.03.1.x86_64-gp2  Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
ap-southeast-2  ami-fd9cecc7  amzn-ami-hvm-2015.03.0.x86_64-gp2  Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
ap-southeast-2  ami-d70773ed  amzn-ami-hvm-2014.09.2.x86_64-gp2  Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
ap-southeast-2  ami-6bf79551  amzn-ami-hvm-2014.09.1.x86_64-gp2  Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
ap-southeast-2  ami-fb4724c1  amzn-ami-hvm-2014.09.0.x86_64-gp2  Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
ap-southeast-2  ami-91d9bcab  amzn-ami-hvm-2014.03.2.x86_64-gp2  Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
eu-central-1    ami-daaeaec7  amzn-ami-hvm-2015.09.0.x86_64-gp2  Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
eu-central-1    ami-a6b0b7bb  amzn-ami-hvm-2015.03.1.x86_64-gp2  Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
eu-central-1    ami-a8221fb5  amzn-ami-hvm-2015.03.0.x86_64-gp2  Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
eu-central-1    ami-0a003317  amzn-ami-hvm-2014.09.2.x86_64-gp2  Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
eu-central-1    ami-b03503ad  amzn-ami-hvm-2014.09.1.x86_64-gp2  Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
eu-central-1    ami-744a7c69  amzn-ami-hvm-2014.09.0.x86_64-gp2  Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
eu-central-1    ami-224c7a3f  amzn-ami-hvm-2014.03.2.x86_64-gp2  Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
eu-west-1       ami-69b9941e  amzn-ami-hvm-2015.09.0.x86_64-gp2  Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
eu-west-1       ami-e4d18e93  amzn-ami-hvm-2015.03.1.x86_64-gp2  Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
eu-west-1       ami-a10897d6  amzn-ami-hvm-2015.03.0.x86_64-gp2  Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
eu-west-1       ami-8723aef0  amzn-ami-hvm-2014.09.2.x86_64-gp2  Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
eu-west-1       ami-6a7bd91d  amzn-ami-hvm-2014.09.1.x86_64-gp2  Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
eu-west-1       ami-768e2901  amzn-ami-hvm-2014.09.0.x86_64-gp2  Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
eu-west-1       ami-dd925baa  amzn-ami-hvm-2014.03.2.x86_64-gp2  Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
```

Now that we have all these set up, we want to use this ***latest_amis.sh*** shell script in another script. We are going to have an option where the user can give a region name and we can show you the AMIs for only that region, therwise we will show you the latest AMIs for all the regions.

```
/tmp/demos $ ls
dev-ec2-instance   latest-amis       my-security-groups remove-sgs
/tmp/demos $ vim latest-amis
```

```
#!/bin/bash

list_amis() {
  # Tab separated output
  # region image_id name description
  local region_name="$1"
  aws ec2 describe-images \
    --filters \
      Name=owner-alias,Values=amazon \
      Name=name,Values="amzn-ami-hvm-*" \
      Name=architecture,Values=x86_64 \
      Name=virtualization-type,Values=hvm \
      Name=root-device-type,Values=ebs \
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images[? !contains(Name, 'rc')], &CreationDate))
          [*].['$region_name',ImageId,Name,Description]" \
  --output text
}

for region_name in $(aws ec2 describe-regions --query "sort(Regions[].RegionName)" --output text); do
  list_amis "$region_name"
done
```

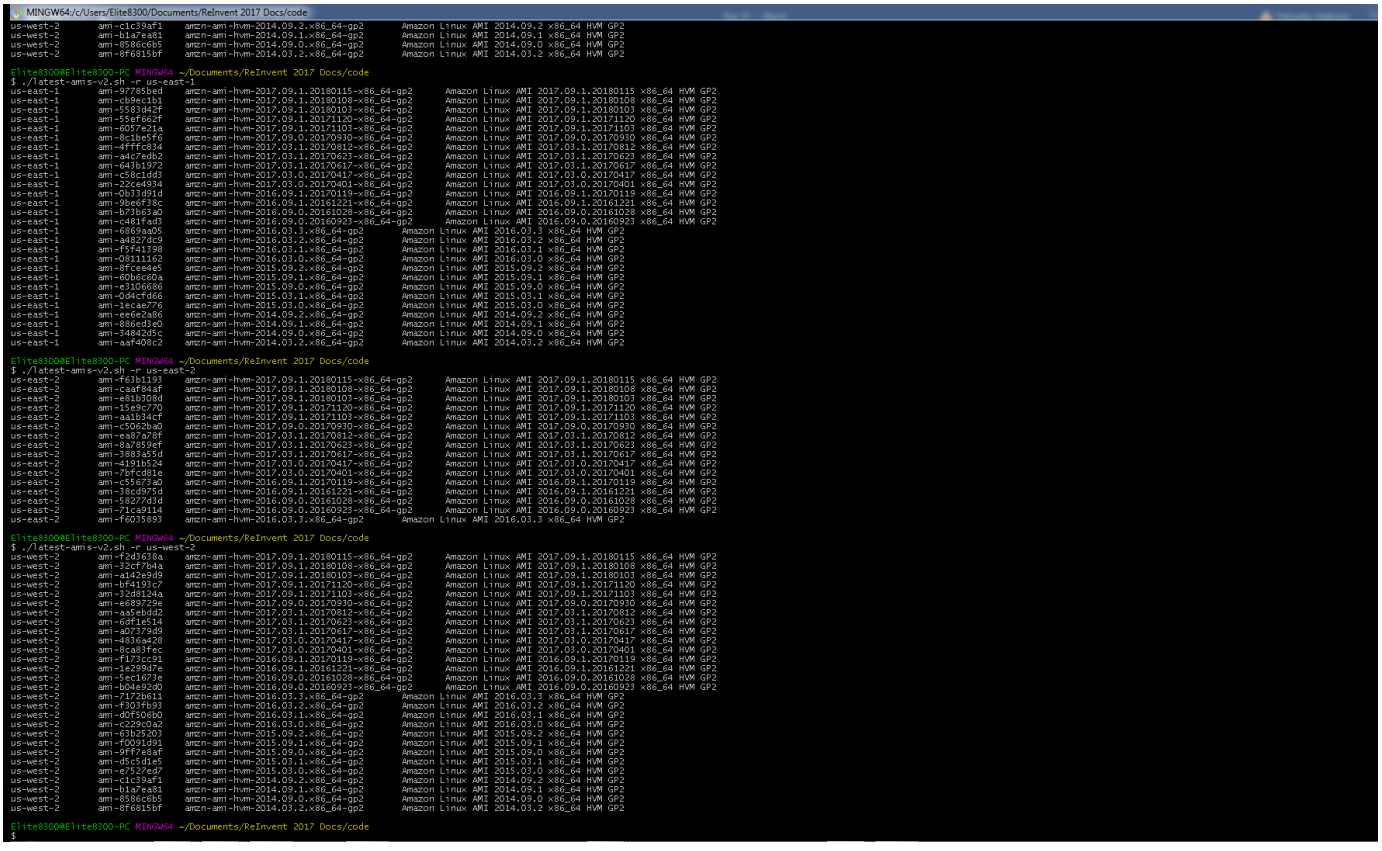
We are now going to write the logic for taking a region name or not below

```
      Name=block-device-mapping.volume-type,Values=gp2 \
  --region "$region_name" \
  --query "reverse(sort_by(Images[? !contains(Name, 'rc')], &CreationDate))
          [*].['$region_name',ImageId,Name,Description]" \
  --output text
}

if [ -z "$1" ]; then
  for region_name in $(aws ec2 describe-regions --query "sort(Regions[].RegionName)" --output text); do
    list_amis "$region_name"
  done
else
  case "$1" in
    -r|--region)
      shift
      region_name="$1"
      ;;
    *)
      echo "usage: latest-amis [-r | --region]" 1>&2
      exit 1
  esac
  list_amis "$region_name"
fi
```

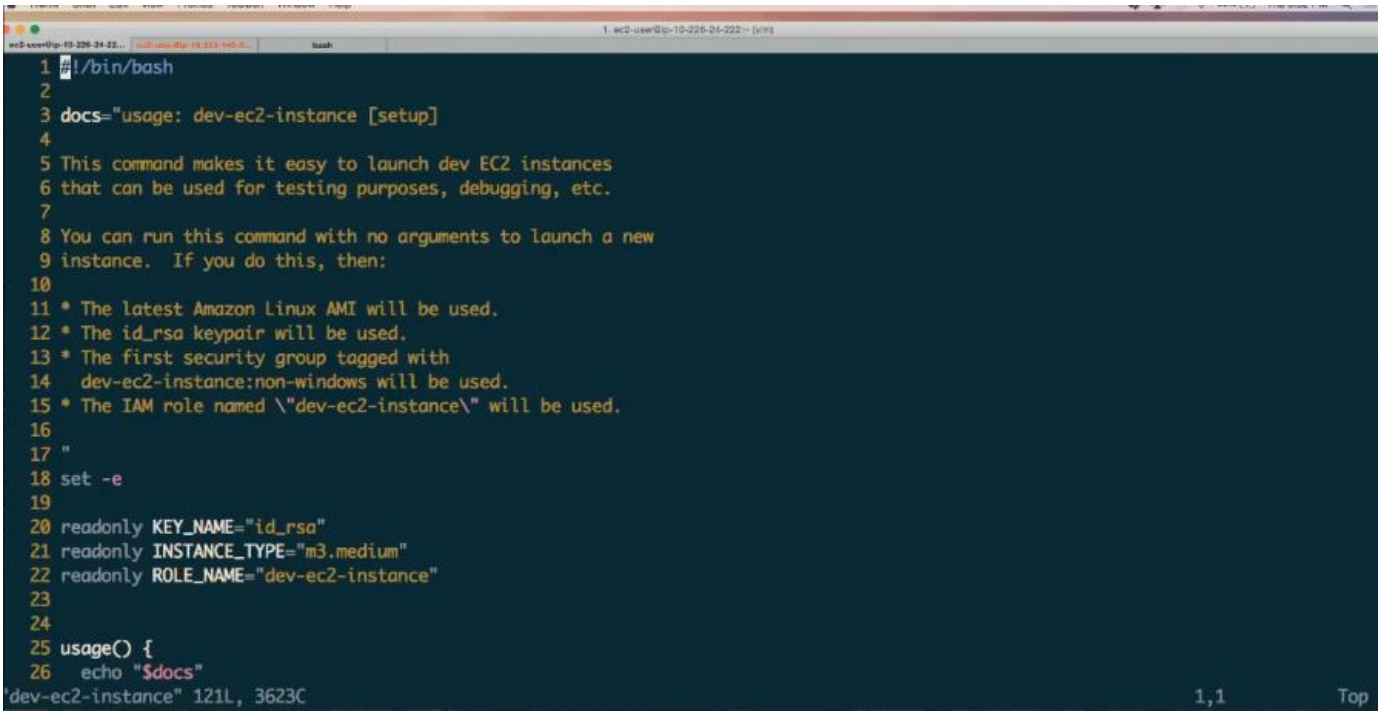
Now we need to run the command

```
/tmp/demos $ ls
dev-ec2-instance   latest-amis        my-security-groups remove-sgs
/tmp/demos $ vim latest-amis
/tmp/demos $ ./latest-amis -r us-west-2
us-west-2       ami-9ff7e8af    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
us-west-2       ami-d5c5d1e5    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
us-west-2       ami-e7527ed7    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
us-west-2       ami-c1c39af1    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
us-west-2       ami-b1a7ea81    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
us-west-2       ami-8586c6b5    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
us-west-2       ami-8f6815bf    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
/tmp/demos $ ./latest-amis -r us-west-1
us-west-1       ami-cd3aff89    amzn-ami-hvm-2015.09.0.x86_64-gp2       Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
us-west-1       ami-87ea13c3    amzn-ami-hvm-2015.03.1.x86_64-gp2       Amazon Linux AMI 2015.03.1 x86_64 HVM GP2
us-west-1       ami-d114f295    amzn-ami-hvm-2015.03.0.x86_64-gp2       Amazon Linux AMI 2015.03.0 x86_64 HVM GP2
us-west-1       ami-5a90891f    amzn-ami-hvm-2014.09.2.x86_64-gp2       Amazon Linux AMI 2014.09.2 x86_64 HVM GP2
us-west-1       ami-556f6510    amzn-ami-hvm-2014.09.1.x86_64-gp2       Amazon Linux AMI 2014.09.1 x86_64 HVM GP2
us-west-1       ami-dba8a19e    amzn-ami-hvm-2014.09.0.x86_64-gp2       Amazon Linux AMI 2014.09.0 x86_64 HVM GP2
us-west-1       ami-e48b8ca1    amzn-ami-hvm-2014.03.2.x86_64-gp2       Amazon Linux AMI 2014.03.2 x86_64 HVM GP2
/tmp/demos $
```

Now we can run the script for single regions using the ***$ ./latest-amis -r us-west-2*** command

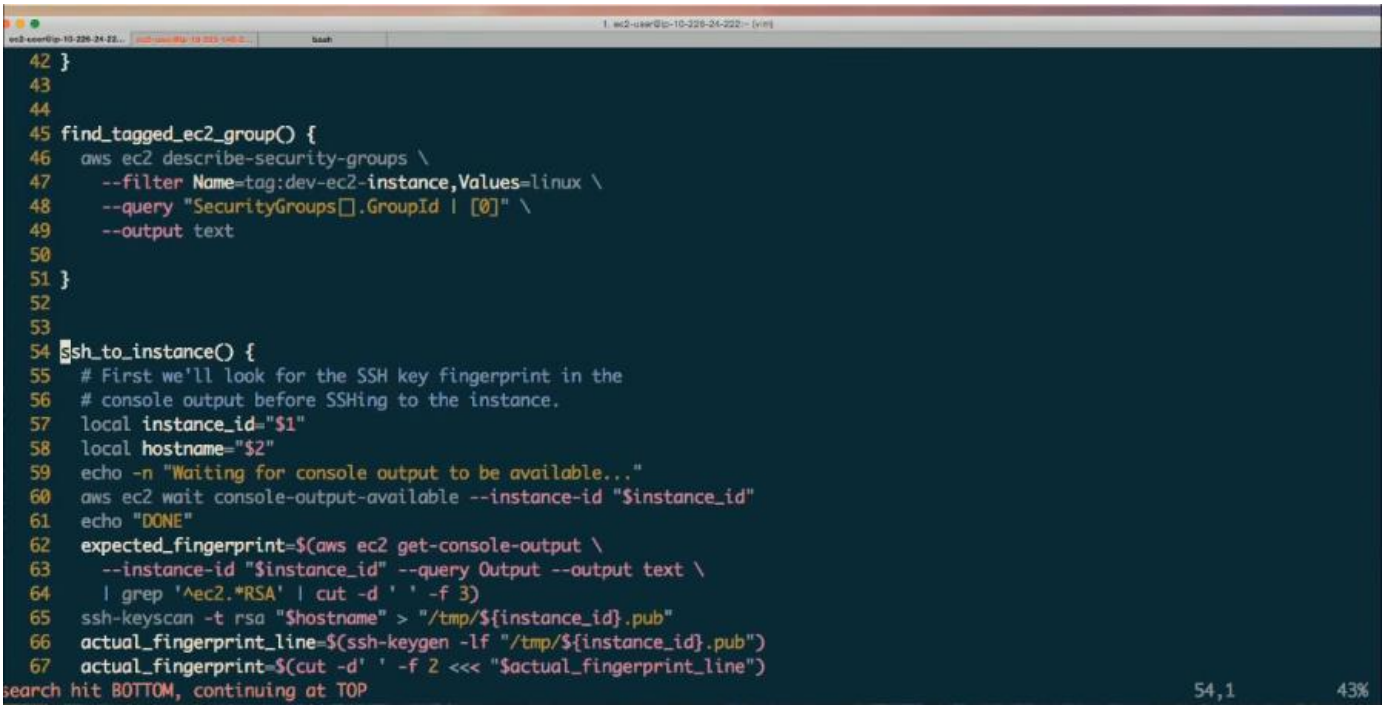


Note: I changed the code version to v2 on my machine to keep the old code copy intact



```
#!/bin/bash

docs="usage: dev-ec2-instance [setup]

This command makes it easy to launch dev EC2 instances
that can be used for testing purposes, debugging, etc.

You can run this command with no arguments to launch a new
instance.  If you do this, then:

* The latest Amazon Linux AMI will be used.
* The id_rsa keypair will be used.
* The first security group tagged with
  dev-ec2-instance:non-windows will be used.
* The IAM role named \"dev-ec2-instance\" will be used.

"
set -e

readonly KEY_NAME="id_rsa"
readonly INSTANCE_TYPE="m3.medium"
readonly ROLE_NAME="dev-ec2-instance"


usage() {
  echo "$docs"
"dev-ec2-instance" 121L, 3623C                                  1,1           Top
```

Let us now see how we can take that tool and start combining tools together



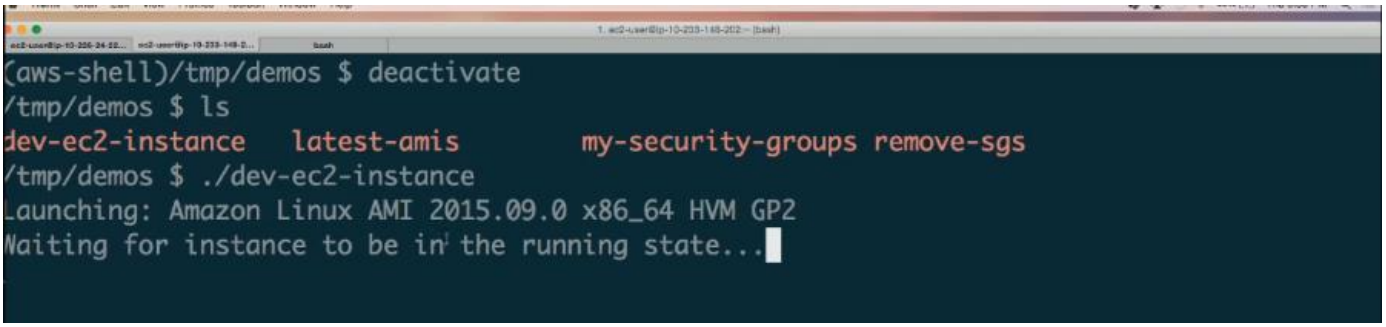
```
}


find_tagged_ec2_group() {
  aws ec2 describe-security-groups \
    --filter Name=tag:dev-ec2-instance,Values=linux \
    --query "SecurityGroups[].GroupId | [0]" \
    --output text

}


ssh_to_instance() {
  # First we'll look for the SSH key fingerprint in the
  # console output before SSHing to the instance.
  local instance_id="$1"
  local hostname="$2"
  echo -n "Waiting for console output to be available..."
  aws ec2 wait console-output-available --instance-id "$instance_id"
  echo "DONE"
  expected_fingerprint=$(aws ec2 get-console-output \
    --instance-id "$instance_id" --query Output --output text \
    | grep '^ec2.*RSA' | cut -d ' ' -f 3)
  ssh-keyscan -t rsa "$hostname" > "/tmp/${instance_id}.pub"
  actual_fingerprint_line=$(ssh-keygen -lf "/tmp/${instance_id}.pub")
  actual_fingerprint=$(cut -d' ' -f 2 <<< "$actual_fingerprint_line")
search hit BOTTOM, continuing at TOP                            54,1          43%
```

```
64      | grep '^ec2.*RSA' | cut -d ' ' -f 3)
65    ssh-keyscan -t rsa "$hostname" > "/tmp/${instance_id}.pub"
66    actual_fingerprint_line=$(ssh-keygen -lf "/tmp/${instance_id}.pub")
67    actual_fingerprint=$(cut -d' ' -f 2 <<< "$actual_fingerprint_line")
68    if [ "$expected_fingerprint" != "$actual_fingerprint" ]; then
69      errexit "SSH fingerprint does not match for $hostname"
70    fi
71    cat < "/tmp/${instance_id}.pub" >> ~/.ssh/known_hosts
72    terminal-notifier -title dev-ec2-instance -message "EC2 instance is now available"
73    ssh "ec2-user@${hostname}"
74  }
75
76  launch_instance() {
77    local image_id="$1"
78    local security_groups=$(find_tagged_ec2_group) || \
79      errexit "Can't find tagged security group"
80
81    local instance_id=$(aws ec2 run-instances \
82      --image-id "$image_id" \
83      --key-name "$KEY_NAME" \
84      --security-group-ids "$security_groups" \
85      --instance-type "$INSTANCE_TYPE" \
86      --iam-instance-profile "Name=$ROLE_NAME" \
87      --output text \
88      --query "Instances[0].InstanceId")
89
search hit BOTTOM, continuing at TOP                                    76,1          66%
```

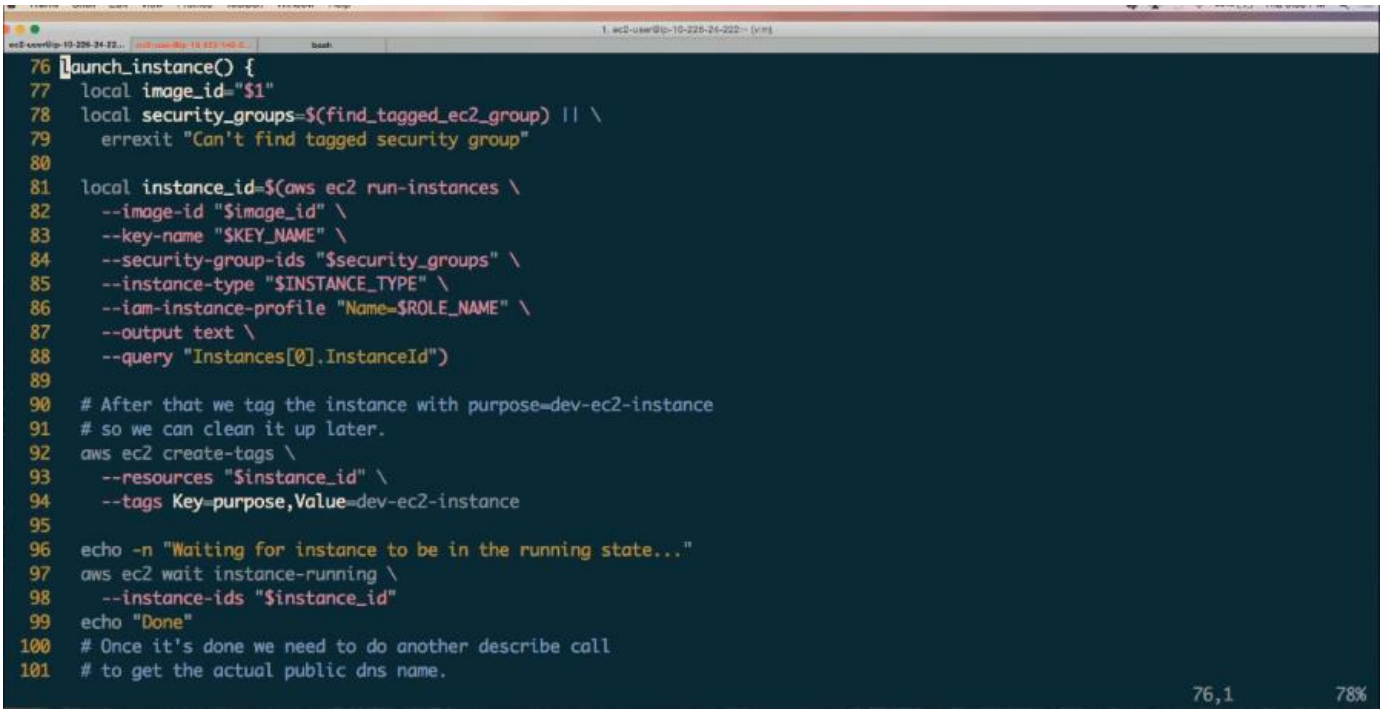
```
96    echo -n "Waiting for instance to be in the running state..."
97    aws ec2 wait instance-running \
98      --instance-ids "$instance_id"
99    echo "Done"
100   # Once it's done we need to do another describe call
101   # to get the actual public dns name.
102   local instance_hostname=$(aws ec2 describe-instances \
103     --instance-ids "$instance_id" \
104     --output text \
105     --query Reservations[0].Instances[0].PublicDnsName)
106   echo -n "Waiting for SSH port to be available on ${instance_hostname}..."
107   wait_for_port "$instance_hostname" 22
108   echo "Done"
109   echo ""
110   # Now actually SSH to instance
111   ssh_to_instance "$instance_id" "$instance_hostname"
112 }
113
114
115 export PATH="$PATH:$(pwd)"
116 region=${AWS_DEFAULT_REGION:-$(aws configure get region)}
117 latest_amis=$(latest-amis -r "$region")
118 latest_ami=$(grep -v rc <<< "$latest_amis" | head -n 1)
119 # Format is: <region>\t<image_id>\t<name>\t<description>
120 echo "Launching: $(cut -f 4 <<< "$latest_ami" )"
121 launch_instance "$(cut -f 2 <<< "$latest_ami")"
                                                                        121,1         Bot
```

```
115 export PATH="$PATH:$(pwd)"
116 region=${AWS_DEFAULT_REGION:-$(aws configure get region)}
117 latest_amis=$(latest-amis -r "$region")
118 latest_ami=$(grep -v rc <<< "$latest_amis" | head -n 1)
119 # Format is: <region>\t<image_id>\t<name>\t<description>
120 echo "Launching: $(cut -f 4 <<< "$latest_ami" )"
121 launch_instance "$(cut -f 2 <<< "$latest_ami")"
~
~
```

We have a script that can grab the latest AMI in a region, then launch an EC2 instance with it. Let us go further by waiting for the launched EC2 instance to be available, let also wait for the SSH port to be available on the instance, then let us SSH into the instance.



On line 116, we are grabbing a default value called ***AWS_DEFAULT_REGION*** or make the ***$(aws configure get region)*** call if the default is not available. Then on line 117 we are using our custom shell script to grab the value of the latest AMI in that region, print out the value, then call the launch_instance command to launch an EC2 instance with that AMI.



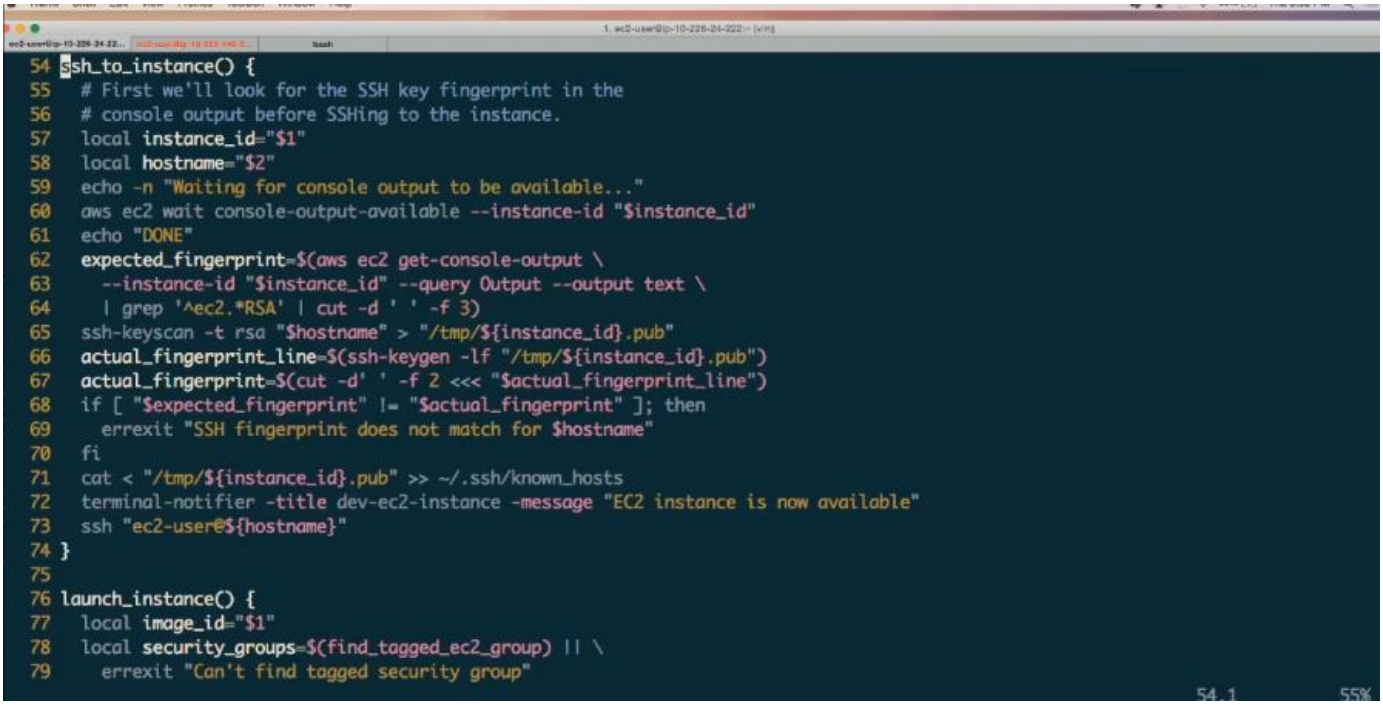
Above is the ***launch_instance*** function

```
 96  echo -n "Waiting for instance to be in the running state..."
 97  aws ec2 wait instance-running \
 98    --instance-ids "$instance_id"
 99  echo "Done"
100  # Once it's done we need to do another describe call
101  # to get the actual public dns name.
102  local instance_hostname=$(aws ec2 describe-instances \
103    --instance-ids "$instance_id" \
104    --output text \
105    --query Reservations[0].Instances[0].PublicDnsName)
106  echo -n "Waiting for SSH port to be available on ${instance_hostname}..."
107  wait_for_port "$instance_hostname" 22
108  echo "Done"
109  echo ""
110  # Now actually SSH to instance
111  ssh_to_instance "$instance_id" "$instance_hostname"
112 }
113
114
115 export PATH="$PATH:$(pwd)"
116 region=${AWS_DEFAULT_REGION:-$(aws configure get region)}
117 latest_amis=$(latest-amis -r "$region")
118 latest_ami=$(grep -v rc <<< "$latest_amis" | head -n 1)
119 # Format is: <region>\t<image_id>\t<name>\t<description>
120 echo "Launching: $(cut -f 4 <<< "$latest_ami" )"
121 launch_instance "$(cut -f 2 <<< "$latest_ami")"
```

On line 97, we are using the ***aws ec2 wait instance_running*** command that will block until the instance associated with the ***instance_id*** we provided hits the RUNNING state. In the background it is making a ***describe_instances*** call and it knows where in that response call the instance's state value is, and it is going to check if it matches ***RUNNING*** and if it doesn't, it will sleep for a little bit and then try again.

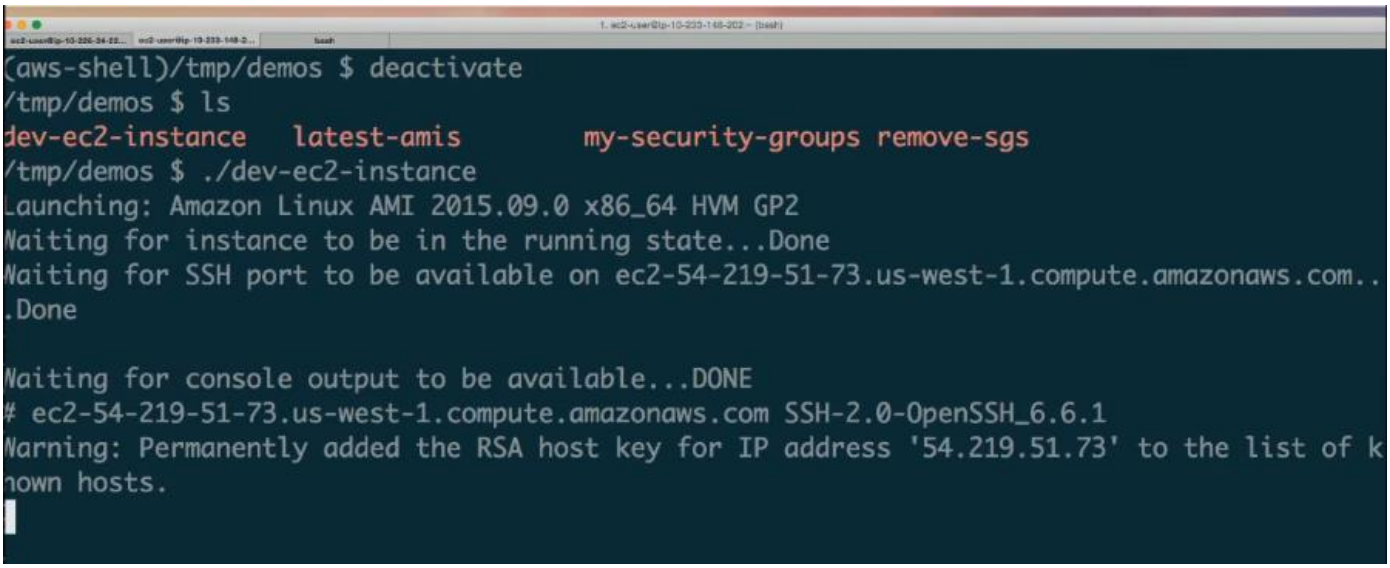
Once the instance is RUNNING, we will grab the ***instance_hostname*** which is the public DNS name for the instance, we then wait for the port to be available using the ***wait_for_port "$instance_hostname" 22*** command on line 107, then we SSH into the instance using the ***ssh_to_instance "$instance_id" "$instance_hostname"*** on line 111 above.

```
 42 }
 43
 44
 45 find_tagged_ec2_group() {
 46   aws ec2 describe-security-groups \
 47     --filter Name=tag:dev-ec2-instance,Values=linux \
 48     --query "SecurityGroups[].GroupId | [0]" \
 49     --output text
 50
 51 }
 52
 53
 54 ssh_to_instance() {
 55   # First we'll look for the SSH key fingerprint in the
 56   # console output before SSHing to the instance.
 57   local instance_id="$1"
 58   local hostname="$2"
 59   echo -n "Waiting for console output to be available..."
 60   aws ec2 wait console-output-available --instance-id "$instance_id"
 61   echo "DONE"
 62   expected_fingerprint=$(aws ec2 get-console-output \
 63     --instance-id "$instance_id" --query Output --output text \
 64     | grep '^ec2.*RSA' | cut -d ' ' -f 3)
 65   ssh-keyscan -t rsa "$hostname" > "/tmp/${instance_id}.pub"
 66   actual_fingerprint_line=$(ssh-keygen -lf "/tmp/${instance_id}.pub")
 67   actual_fingerprint=$(cut -d' ' -f 2 <<< "$actual_fingerprint_line")
search hit BOTTOM, continuing at TOP                                54,1          43%
```



```
54 ssh_to_instance() {
55   # First we'll look for the SSH key fingerprint in the
56   # console output before SSHing to the instance.
57   local instance_id="$1"
58   local hostname="$2"
59   echo -n "Waiting for console output to be available..."
60   aws ec2 wait console-output-available --instance-id "$instance_id"
61   echo "DONE"
62   expected_fingerprint=$(aws ec2 get-console-output \
63     --instance-id "$instance_id" --query Output --output text \
64     | grep 'Aec2.*RSA' | cut -d ' ' -f 3)
65   ssh-keyscan -t rsa "$hostname" > "/tmp/${instance_id}.pub"
66   actual_fingerprint_line=$(ssh-keygen -lf "/tmp/${instance_id}.pub")
67   actual_fingerprint=$(cut -d' ' -f 2 <<< "$actual_fingerprint_line")
68   if [ "$expected_fingerprint" != "$actual_fingerprint" ]; then
69     errexit "SSH fingerprint does not match for $hostname"
70   fi
71   cat < "/tmp/${instance_id}.pub" >> ~/.ssh/known_hosts
72   terminal-notifier -title dev-ec2-instance -message "EC2 instance is now available"
73   ssh "ec2-user@${hostname}"
74 }
75
76 launch_instance() {
77   local image_id="$1"
78   local security_groups=$(find_tagged_ec2_group) || \
79     errexit "Can't find tagged security group"
```

Above is the ***ssh_to_instance*** function used, we can see how to start integrating with some of the local commands as well as AWS CLI calls.

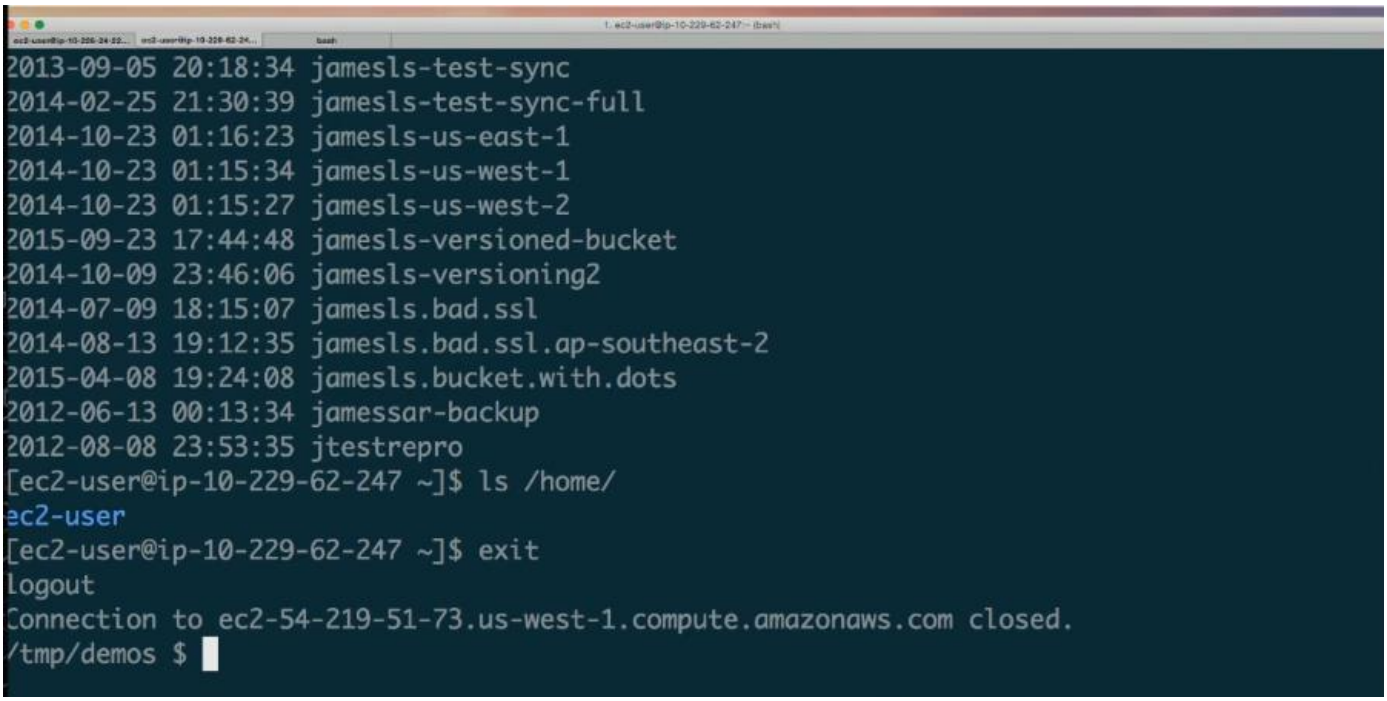


```
(aws-shell)/tmp/demos $ deactivate
/tmp/demos $ ls
dev-ec2-instance   latest-amis        my-security-groups remove-sgs
/tmp/demos $ ./dev-ec2-instance
Launching: Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
Waiting for instance to be in the running state...Done
Waiting for SSH port to be available on ec2-54-219-51-73.us-west-1.compute.amazonaws.com..
.Done

Waiting for console output to be available...DONE
# ec2-54-219-51-73.us-west-1.compute.amazonaws.com SSH-2.0-OpenSSH_6.6.1
Warning: Permanently added the RSA host key for IP address '54.219.51.73' to the list of k
nown hosts.
```

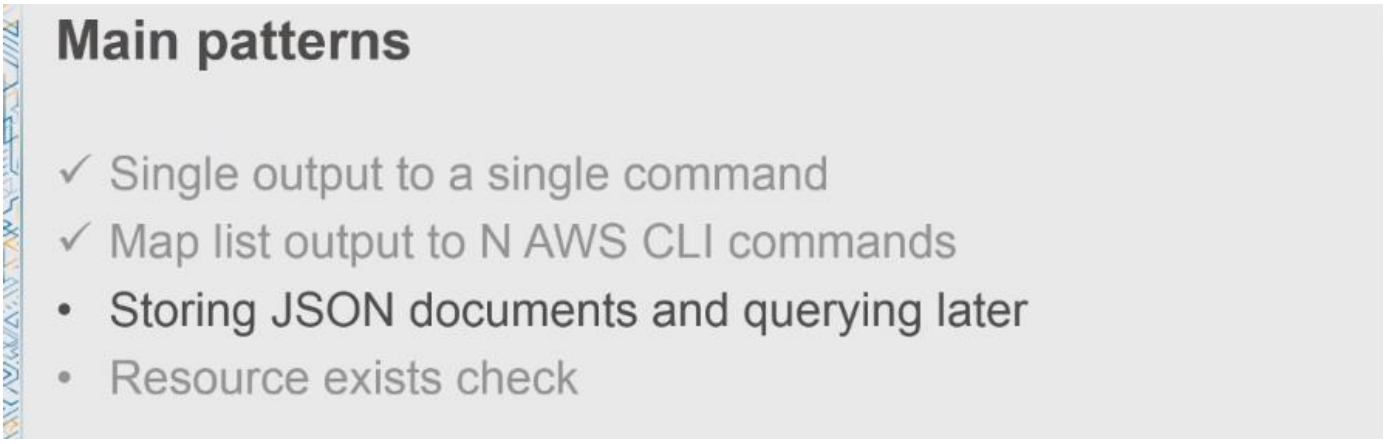
```
Launching: Amazon Linux AMI 2015.09.0 x86_64 HVM GP2
Waiting for instance to be in the running state...Done
Waiting for SSH port to be available on ec2-54-219-51-73.us-west-1.compute.amazonaws.com..
.Done

Waiting for console output to be available...DONE
# ec2-54-219-51-73.us-west-1.compute.amazonaws.com SSH-2.0-OpenSSH_6.6.1
Warning: Permanently added the RSA host key for IP address '54.219.51.73' to the list of k
nown hosts.

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

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
No packages needed for security; 7 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-229-62-247 ~]$
```

We are now on an EC2 instance and can start running CLI commands as below

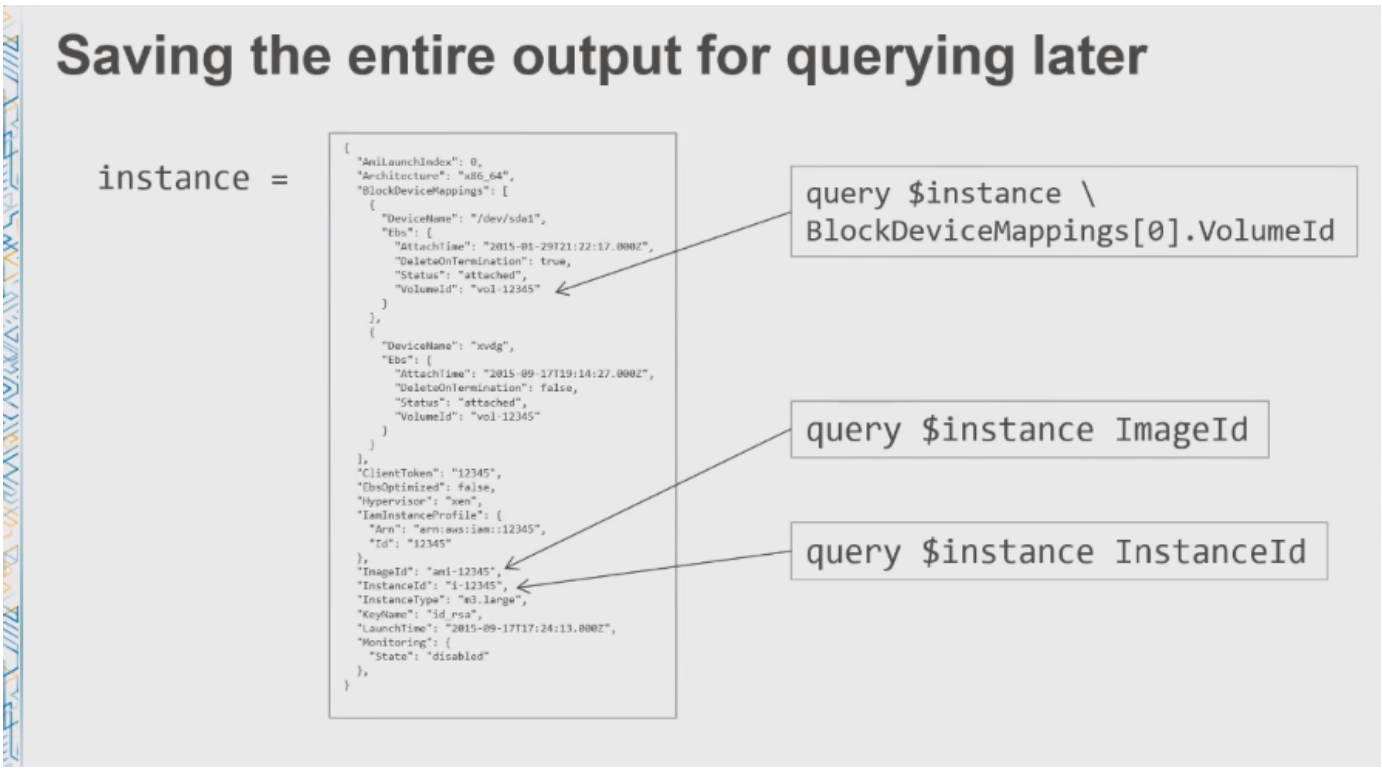
```
.Done

Waiting for console output to be available...DONE
# ec2-54-219-51-73.us-west-1.compute.amazonaws.com SSH-2.0-OpenSSH_6.6.1
Warning: Permanently added the RSA host key for IP address '54.219.51.73' to the list of k
nown hosts.

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

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
No packages needed for security; 7 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-229-62-247 ~]$
[ec2-user@ip-10-229-62-247 ~]$
[ec2-user@ip-10-229-62-247 ~]$ aws s3 ls
```



The script sets up the EC2 instance for us, and we can now start doing some development work



In complex scripts, we will have multiple context where we want to use values in. In this example, we are going to make some call where we want part of the result for the first part, middle part, and the last parts of our script separately.



Rather than try to figure out the query expression upfront, we are going to store our query as a big JSON variable like above. Then we can later pick out the values we need to use in our queries within the JSON object.

The **- *-query*** flag is part of the CLI and is the expression language that lets you take in JSON data ad filter it down to whatever you want, the ***jp*** flag is a separate executable and does the same thing as the - -query expression but is not part of the CLI.

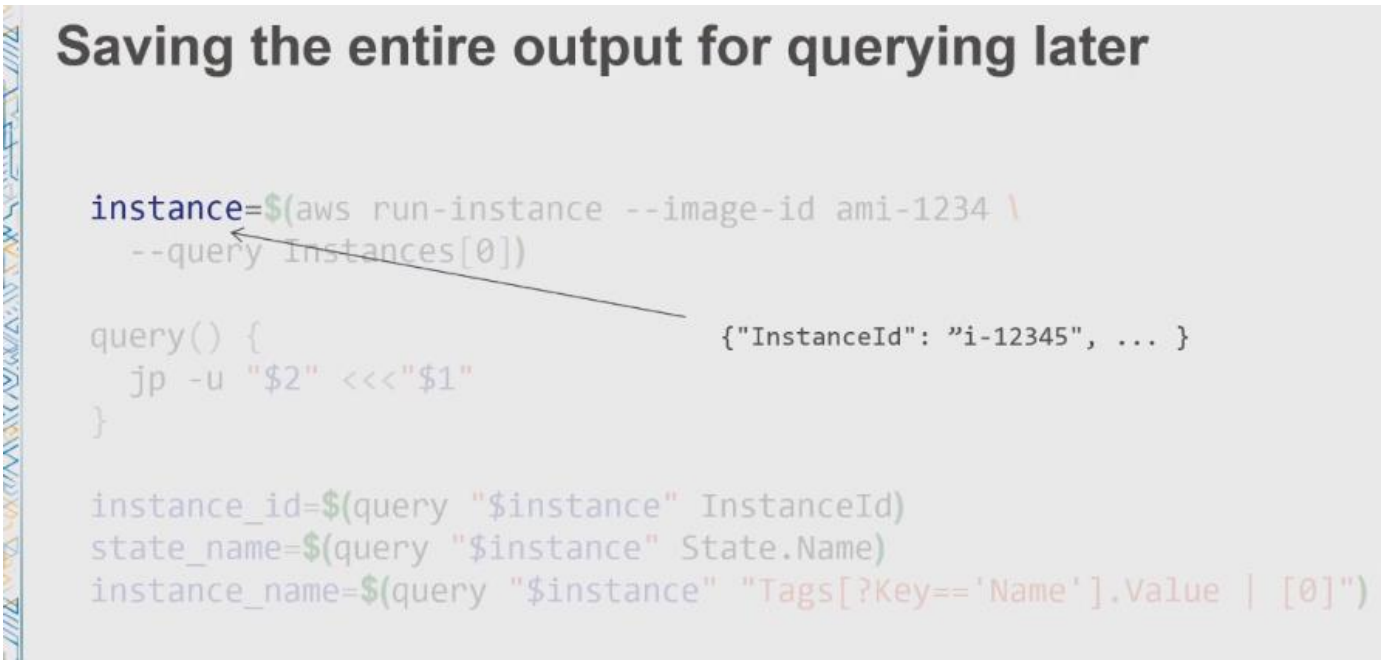
Saving the entire output for querying later

```
instance=$(aws run-instance --image-id ami-1234 \
  --query Instances[0])

query() {
  jp -u "$2" <<<"$1"
}

instance_id=$(query "$instance" InstanceId)
state_name=$(query "$instance" State.Name)
instance_name=$(query "$instance" "Tags[?Key=='Name'].Value | [0]")
```

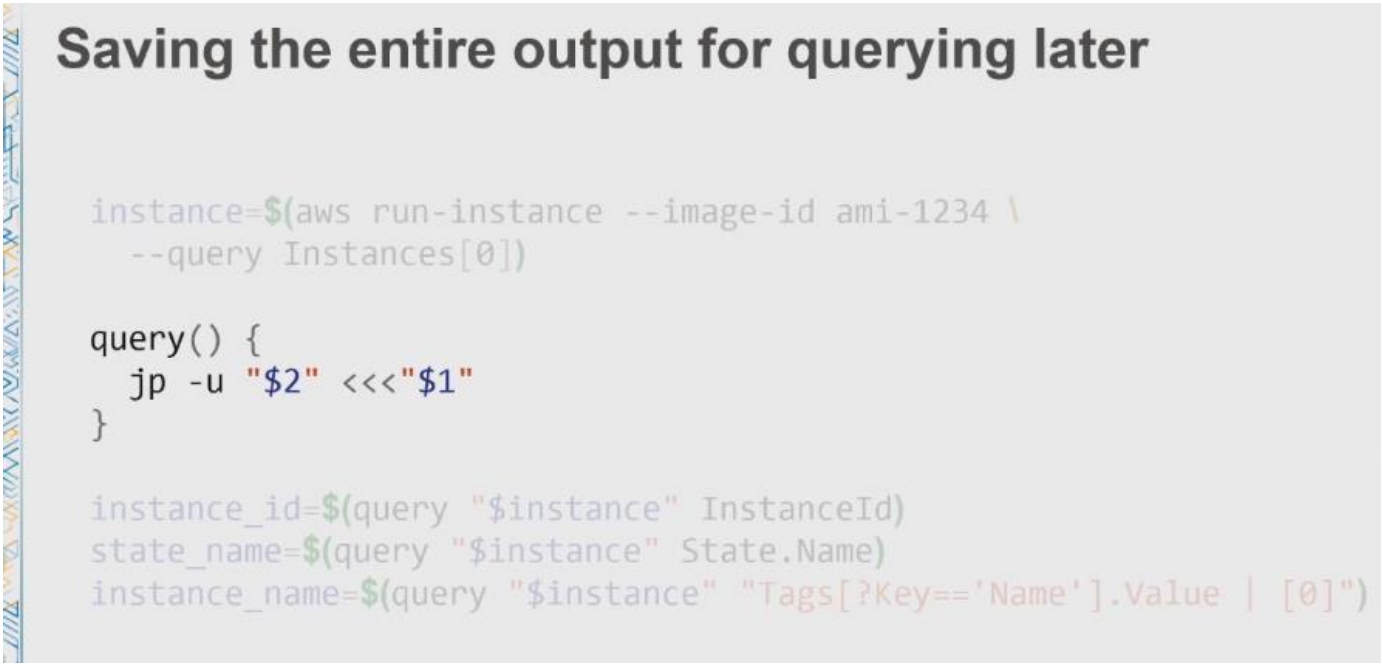

Saving the entire output for querying later

```
instance=$(aws run-instance --image-id ami-1234 \
  --query Instances[0])

query() {
  jp -u "$2" <<<"$1"
}

instance_id=$(query "$instance" InstanceId)
state_name=$(query "$instance" State.Name)
instance_name=$(query "$instance" "Tags[?Key=='Name'].Value | [0]")
```


To use this, we use the above command, note that we are note using the ***- -output text*** flag since we are outputting JSON. We then save this query as a variable called ***instance***.



We now have the query function called instance as above



The query() query function above takes 2 arguments, the instance data and an expression. This allows us to do the following in our shell scripts below

Saving the entire output for querying later

```
instance=$(aws run-instance --image-id ami-1234 \
  --query Instances[0])

query() {
  jp -u "$2" <<<"$1"
}

instance_id=$(query "$instance" InstanceId)
state_name=$(query "$instance" State.Name)
instance_name=$(query "$instance" "Tags[?Key=='Name'].Value | [0]")
```

The first line says we need an instance_id as a variable, the second line gives the state.name as a variable, the third line is finding an instance_name of the instance that has a Tag name of Name or the first instance

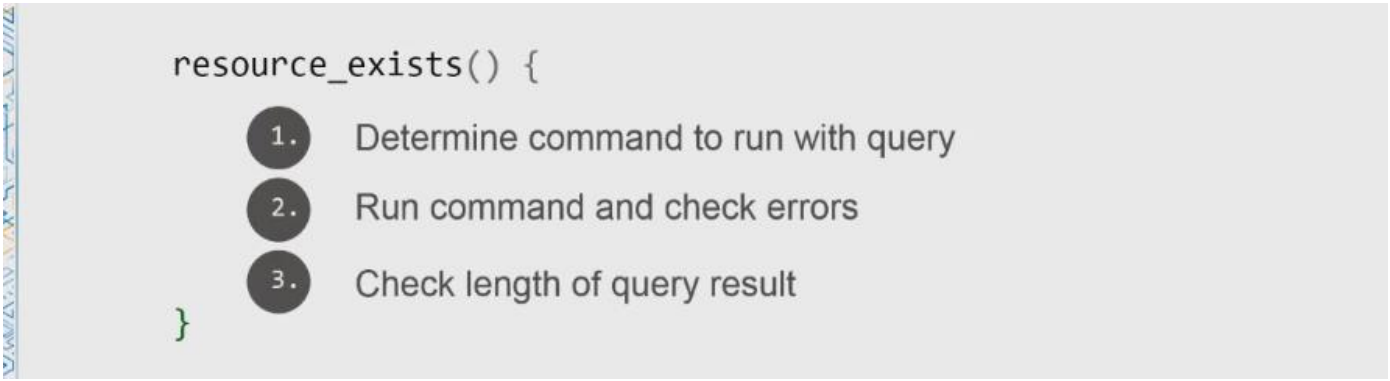
Saving the entire output for querying later

1. Use JSON output, not text, --output json
2. Store the entire thing as a variable, or write to a temp file if output size is a concern.
3. Use the “jp” command to filter down results as needed.

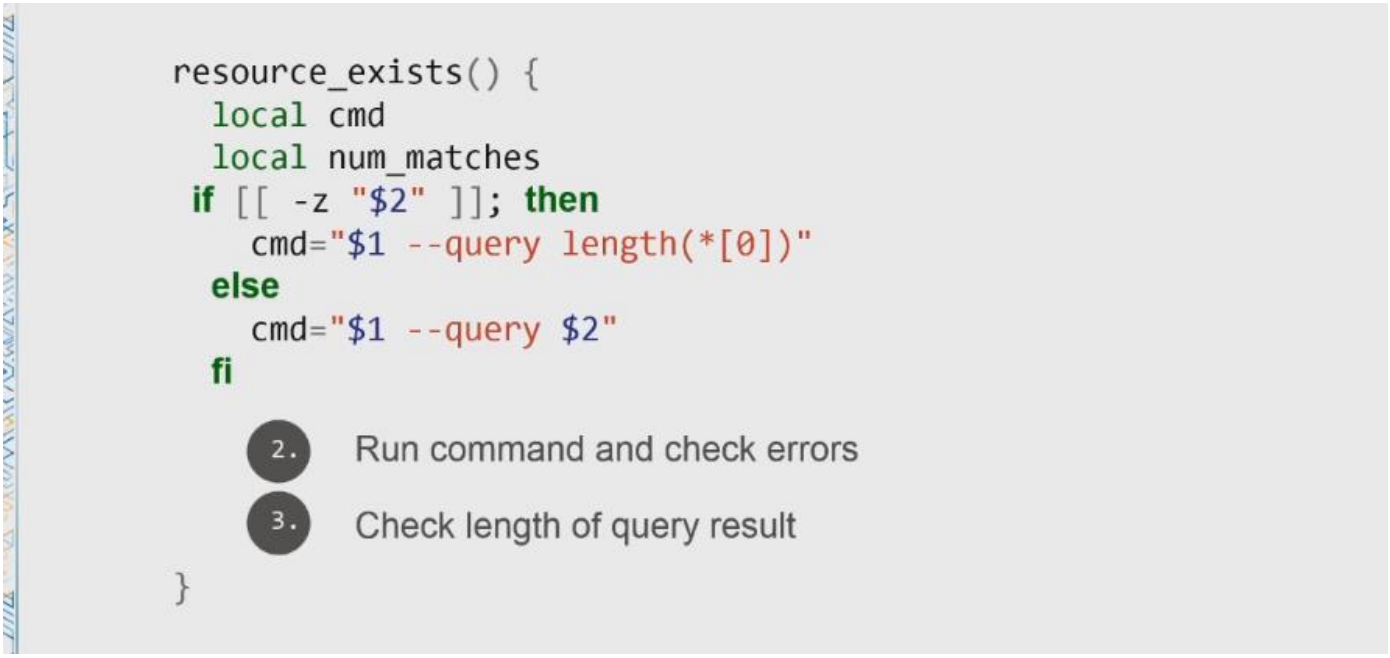
Main Patterns

- ✓ Single output to a single command
- ✓ Map list output to N AWS CLI commands
- ✓ Storing JSON documents and querying later
- • Resource exists check

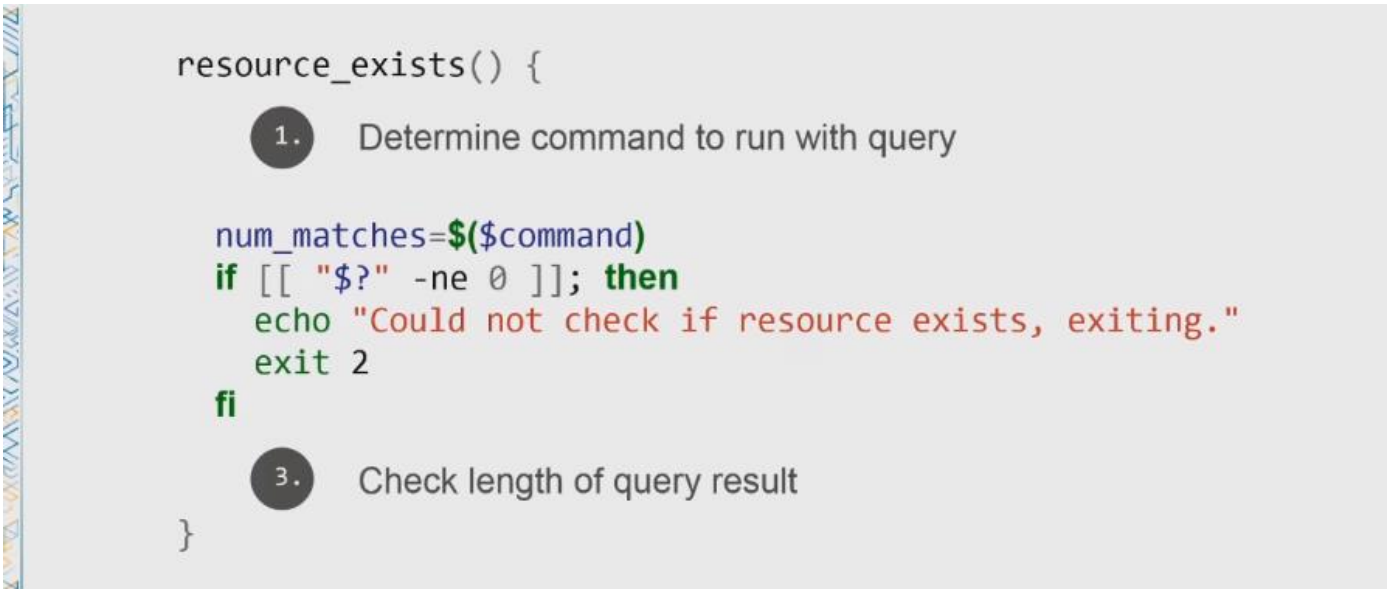
This is when we want to check if a resource exists, if it does we want to take an Action A or otherwise take an Action B.



We will try to create a sort of list, if that list is empty then the resource does not exist, but if the list contains something then the resource exists.



We are going to need 2 arguments, the command and the query to run.



After that we will go ahead and run some command and if the command output has any errors we will error out.

```
resource_exists() {
    1. Determine command to run with query
    2. Run command and check errors
  if [[ "$num_matches" -gt 0 ]]; then
     # RC of 0 mean the resource exists, success.
     return 0
   else
     return 1
   fi
}
```

Finally, we check the length of the list here.

Usage: Server-side filtering

```
describe_groups="aws ec2 describe-security-groups"
if resource_exists "$describe_groups \
  --filter Name=tag:dev-ec2-instance,Values=linux"; then
  echo "Security groups exists."
else
  echo "Security group not found."
fi
```

We can use the above pattern for server-side filtering as above. We are checking if a particular EC2 security group exists,

Usage: Client-side filtering

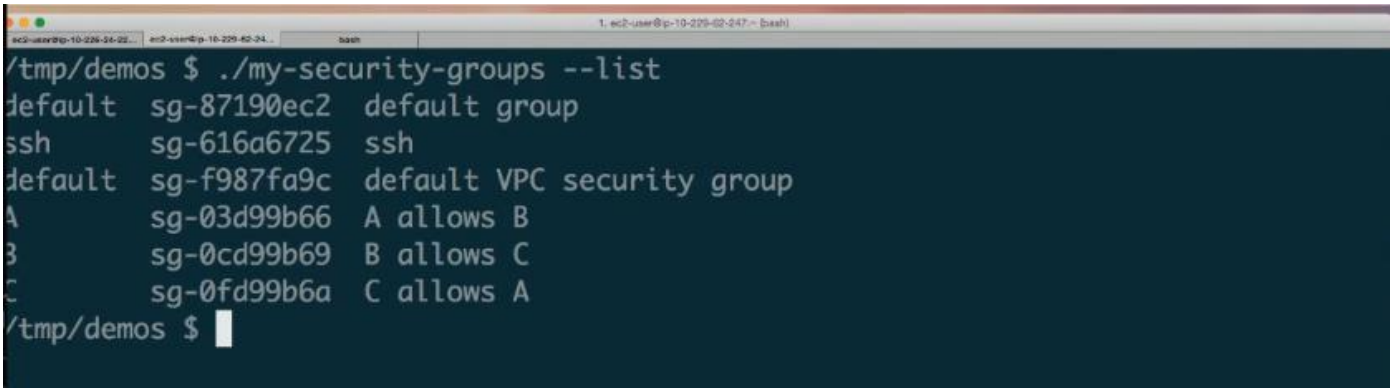
```
list_profiles="aws iam list-instance-profiles"
if resource_exists "$list_profiles" \
  "InstanceProfiles[?InstanceProfileName=='dev-ec2-instance']"
then
  echo "Instance profile exists."
else
  echo "Missing IAM instance profile 'dev-ec2-instance'"
  create_instance_profile
fi
```

We can also use the pattern for client-side filtering as above. Here we have a query expression that is looking for an EC2 instance having an InstanceProfile name of ‘dev-ec2-instance’.

Resource exists

- Check if a resource exists by filtering down a list and checking if the length is greater than 0.
- Use server side filtering when possible
- Use JMESPath query for client side filtering

Demo



We can display the list of available SGs using the command ***$ ./my-security-groups - -list***. The columns displayed are the ***security group name***, ***groupId***, and the ***description***.

```
/tmp/demos $ clear
/tmp/demos $ ./my-security-groups --list
default  sg-87190ec2  default group
ssh      sg-616a6725  ssh
default  sg-f987fa9c  default VPC security group
A        sg-03d99b66  A allows B
B        sg-0cd99b69  B allows C
C        sg-0fd99b6a  C allows A
/tmp/demos $ ./my-security-groups --list | grep -v default | grep -v ssh
A        sg-03d99b66  A allows B
B        sg-0cd99b69  B allows C
C        sg-0fd99b6a  C allows A
/tmp/demos $
```

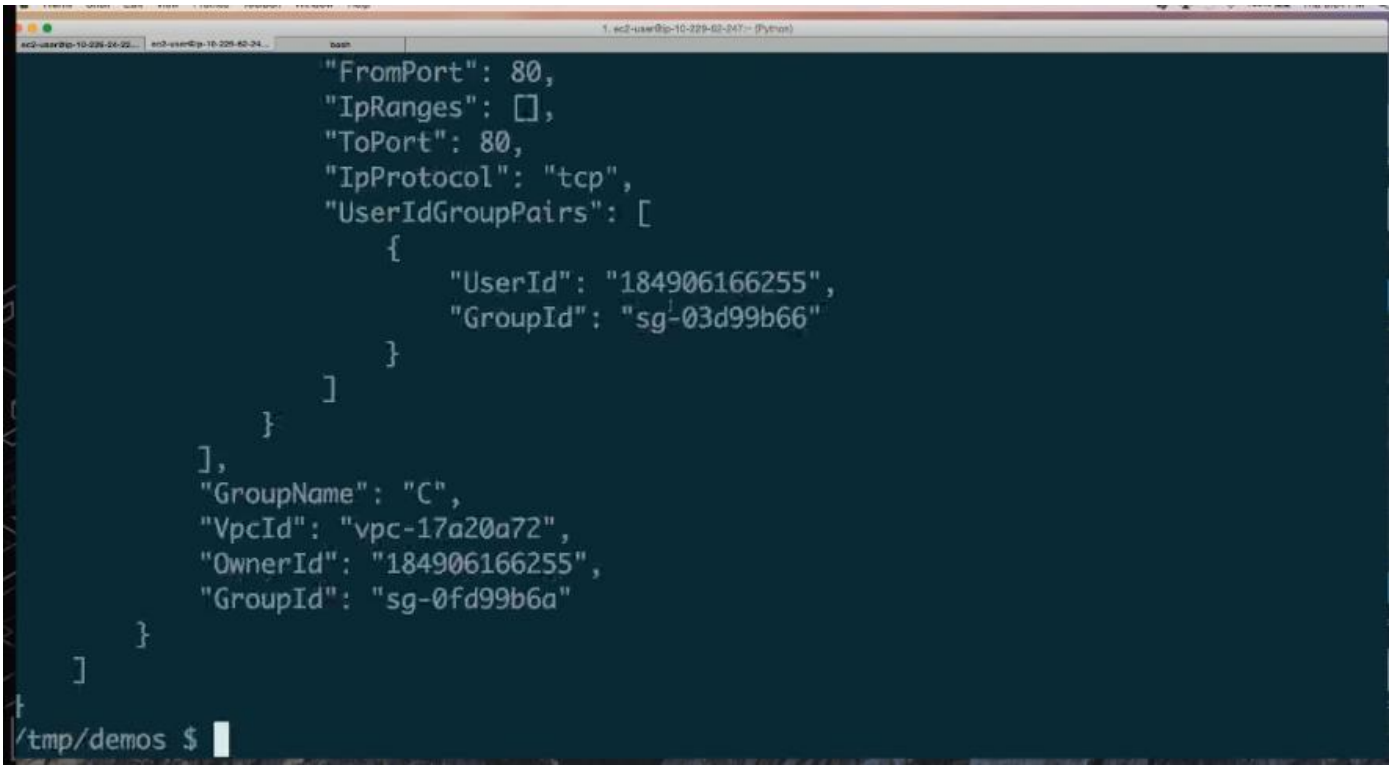
We can go ahead and remove the default and ssh SGs from the list result using the command ***$ ./my-security-groups –list | grep -v default | grep -v ssh***

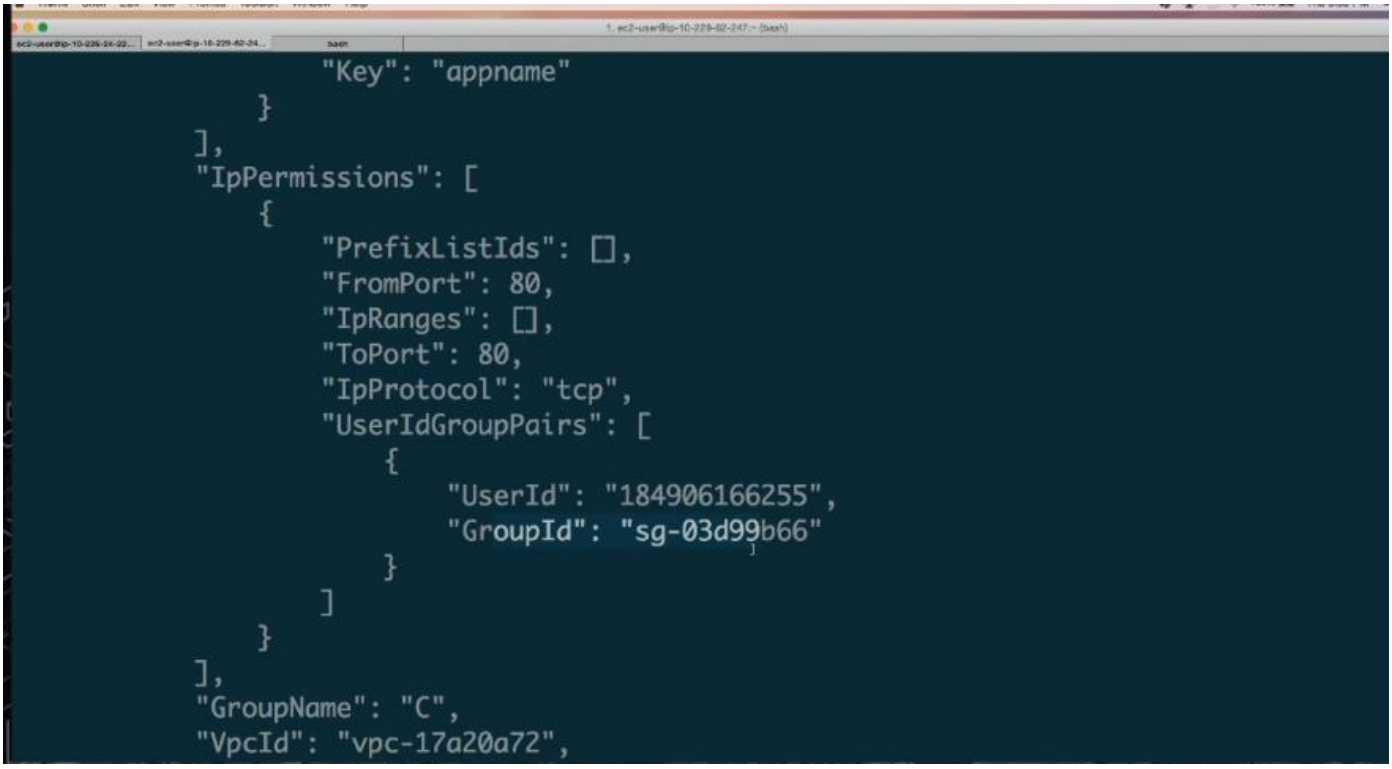
```
A        sg-03d99b66  A allows B
B        sg-0cd99b69  B allows C
C        sg-0fd99b6a  C allows A
/tmp/demos $ ./my-security-groups --list | grep -v default | grep -v ssh
A        sg-03d99b66  A allows B
B        sg-0cd99b69  B allows C
C        sg-0fd99b6a  C allows A
/tmp/demos $ aws ec2 delete-security-group --group-id sg-03d99b66

A client error (DependencyViolation) occurred when calling the DeleteSecurityGroup opera
tion: resource sg-03d99b66 has a dependent object
/tmp/demos $ aws ec2 delete-security-group --group-id sg-0cd99b69

A client error (DependencyViolation) occurred when calling the DeleteSecurityGroup opera
tion: resource sg-0cd99b69 has a dependent object
/tmp/demos $ aws ec2 delete-security-group --group-id sg-0fd99b6a

A client error (DependencyViolation) occurred when calling the DeleteSecurityGroup opera
tion: resource sg-0fd99b6a has a dependent object
/tmp/demos $
```

We cannot delete the any of the SGs because something depends on them or they depend on each other.

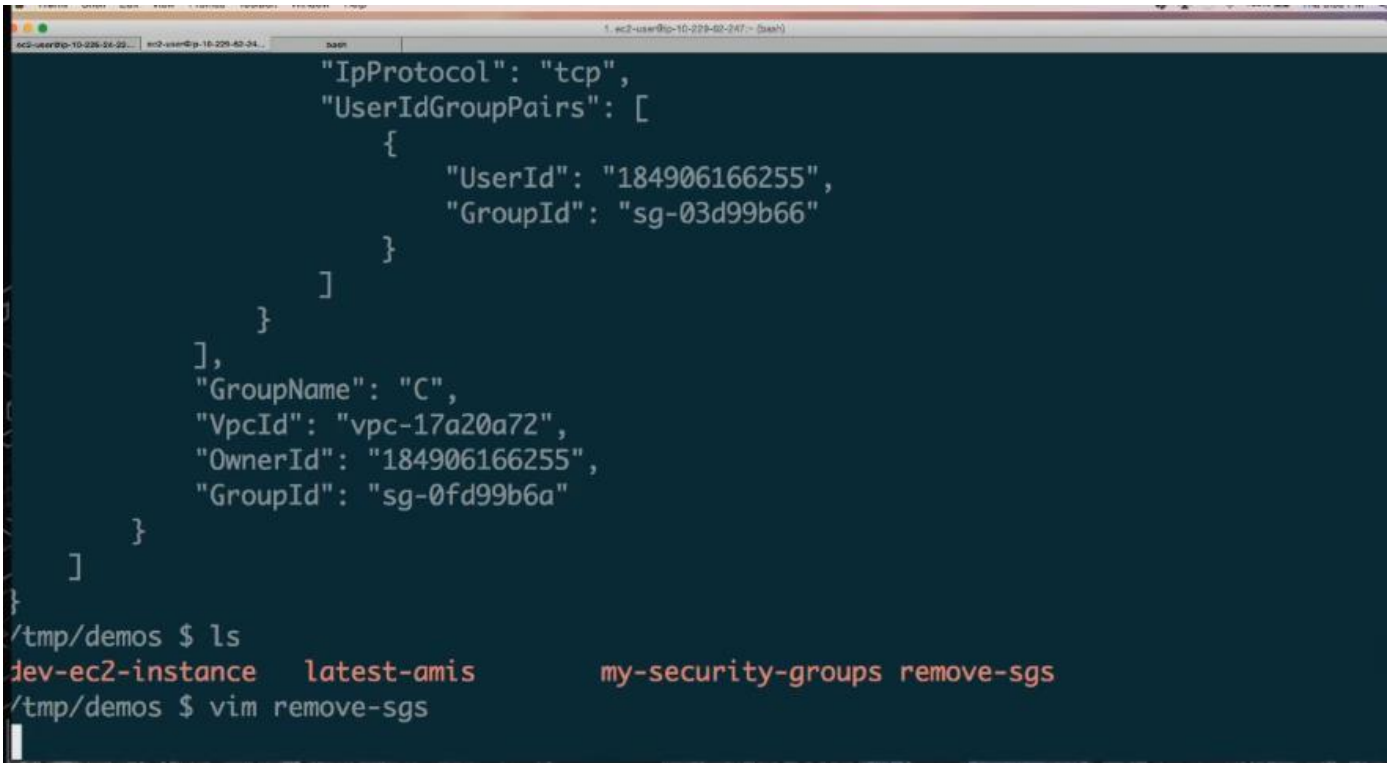
```
/tmp/demos $ aws ec2 describe-security-groups --group-id sg-0fd99b6a
```

To delete the SG, we need to delete all the rules associated with that SG using the command ***$ aws ec2 describe-security-groups - -group-id sg-ofd99b6a***





There is output here that has ***IpPermissions*** that says what this SG is allowed and we can see that it is referencing another ***GroupId*** sg-03d99b66. What we need to do is wipe out all these ***IpPremissions*** and then we can delete this SG.


"IpProtocol": "tcp",
"UserIdGroupPairs": [
{
"UserId": "184906166255",
"GroupId": "sg-03d99b66"
}
]
}
],
"GroupName": "C",
"VpcId": "vpc-17a20a72",
"OwnerId": "184906166255",
"GroupId": "sg-0fd99b6a"
}
]
}
/tmp/demos $ ls
dev-ec2-instance latest-amis my-security-groups remove-sgs
/tmp/demos $ vim remove-sgs


1 #!/bin/bash
2
3 # We require the --region argument because this is such
4 # a destructive thing you have to be absolutely sure you
5 # want to do this.
6 docs="usage: remove-sgs appname
7
8 This command will remove every single security group in a given
9 region that's tagged with appname=<your-provided-value>.
10 Be VERY careful using this command.
11 "
12 if [[-z "$1"]]; then
13 echo "$docs"
14 exit 1
15 fi
16
17 tag_value_name="$1"
18
19 errexit() {
"remove-sgs" 59L, 1608C 19,1 Top

```
18
19 errexit() {
20   echo "ERROR: $(basename "$0") (line ${LINENO}): ${1:-"Unknown Error"}" 1>&2
21   exit 1
22 }
23
24 query() {
25   jp -u "$2" <<<"$1"
26 }
27
28 security_groups=$(aws ec2 describe-security-groups \
29   --query SecurityGroups \
30   --filters "Name=tag:appname,Values=$tag_value_name") \
31   || errexit "Could not find security groups"
32
33 echo "$security_groups"
34
35 echo ""
@
```

```
26 }
27
28 security_groups=$(aws ec2 describe-security-groups \
29   --query SecurityGroups \
30   --filters "Name=tag:appname,Values=$tag_value_name") \
31   || errexit "Could not find security groups"
32
33 echo "$security_groups"
34
35 echo ""
36 echo "You are about to remove ALL these security groups in the region $region_name
   "
37 echo -n "To confirm, please type 'yes' and press enter: "
38 read confirmation
39 if [[ "$confirmation" != "yes" ]]
40 then
41   echo "Didn't receive a confirmation, exiting."
42   exit 2
43 fi
```

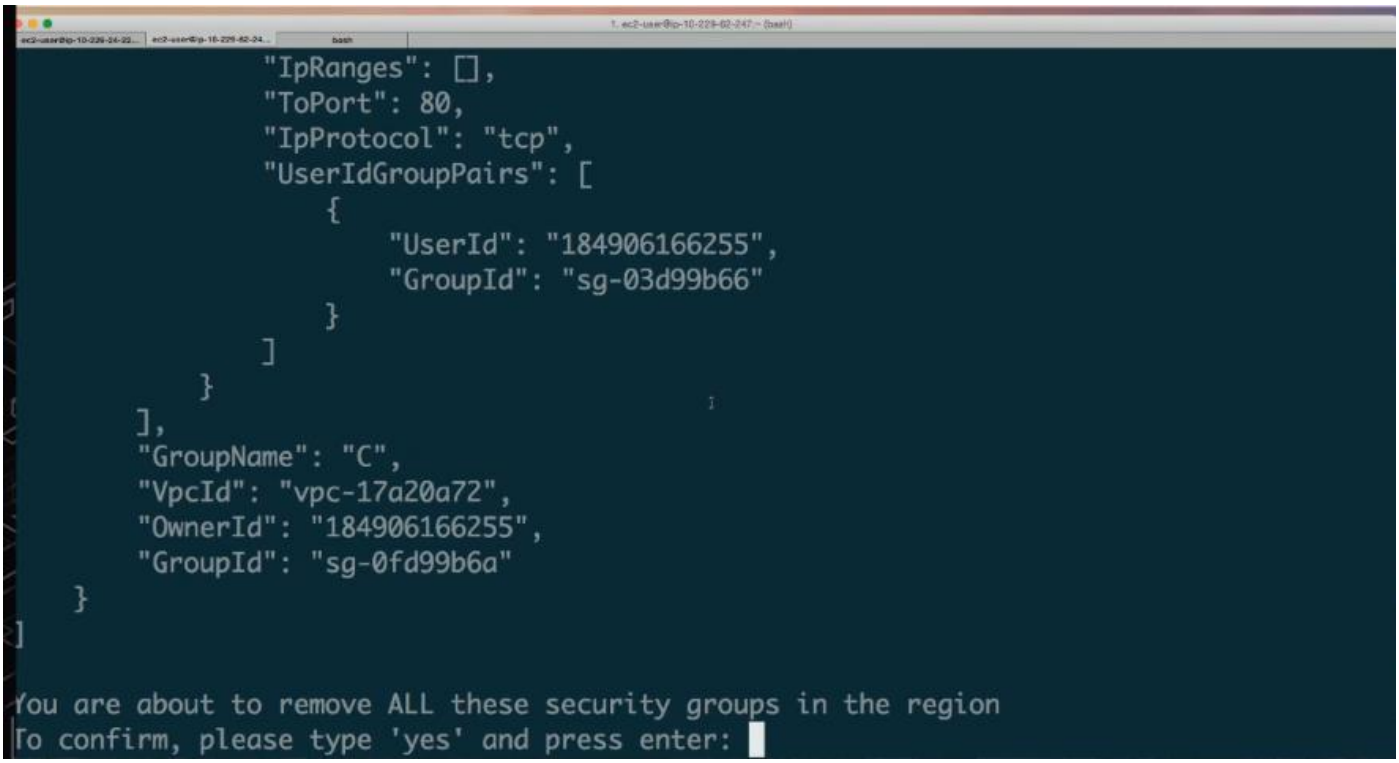
The way we are going to do this is to call ***describe security-groups*** command to see if the SG is still there, then confirm that they want to delete the SG,

```
46 num_groups=$(query "$security_groups" "length(@)")
47
48 for ((i = 0 ; i < "$num_groups" ; i++)); do
49   group_id=$(query "$security_groups" "[$i].GroupId")
50   ip_permissions=$(query "$security_groups" "[$i].IpPermissions")
51   echo "Revoking ingress rules for security group: $group_id"
52   aws ec2 revoke-security-group-ingress --group-id "$group_id" --ip-permissions "$
   ip_permissions"
53 done
54
55 for ((i = 0 ; i < "$num_groups" ; i++)); do
56   group_id=$(query "$security_groups" "[$i].GroupId")
57   echo "Removing security group: $group_id"
58   aws ec2 delete-security-group --group-id "$group_id"
59 done
~
~
```

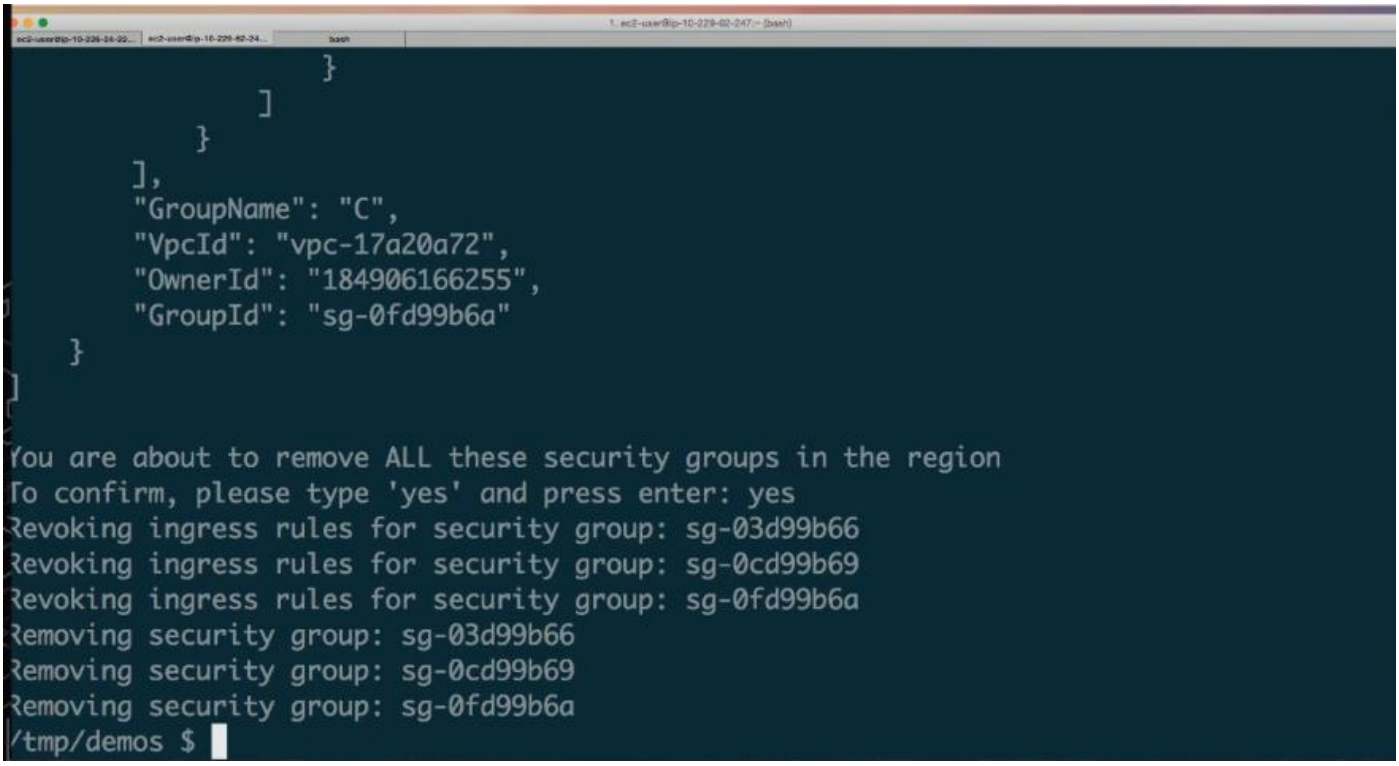
Then we can go ahead and say how many groups are there, and for each group we will have to extract out the ***GroupId*** on line 49 and the ***IpPermissions*** on line 50, then we will revoke the ingress rules using the ***$group_id*** and the ***$ip_permissions*** in a JSON document on line 52. Then on line 55, we can actually go through the available SGs and delete them one by one using the ***aws ec2 delete-security-group - -group-id "$group_id"*** command on line 58.

```
/tmp/demos $ ./remove-sgs myapp
```

We specify we want to delete all SGs having the tag myapp using the ***remove-sgs.sh*** script file and the ***./remove-sgs myapp*** command



It then gives us a list of affected SGs and asks if we are sure we really want to delete them



We answer ‘yes’, we can see that it does the operation in 2 steps by first revoking the ingress rules and then removing the SGs. This completes the example of using query expression to do multiple operations.

Common Patterns

- ✓ Single output to a single command
- ✓ Map list output to N AWS CLI commands
- ✓ Storing JSON documents and querying later
- ✓ Resource exists check

Deployment tips

Cron

```
# * * * * *  command to execute
# | | | | |
# | | | | |
# | | | | └──── day of week (0 – 6)
# | | | └─────── month (1 - 12)
# | | └──────── day of month (1 - 31)
# | └───────── hour (0 - 23)
# └──────── min (0 - 59)
```

$ whoami

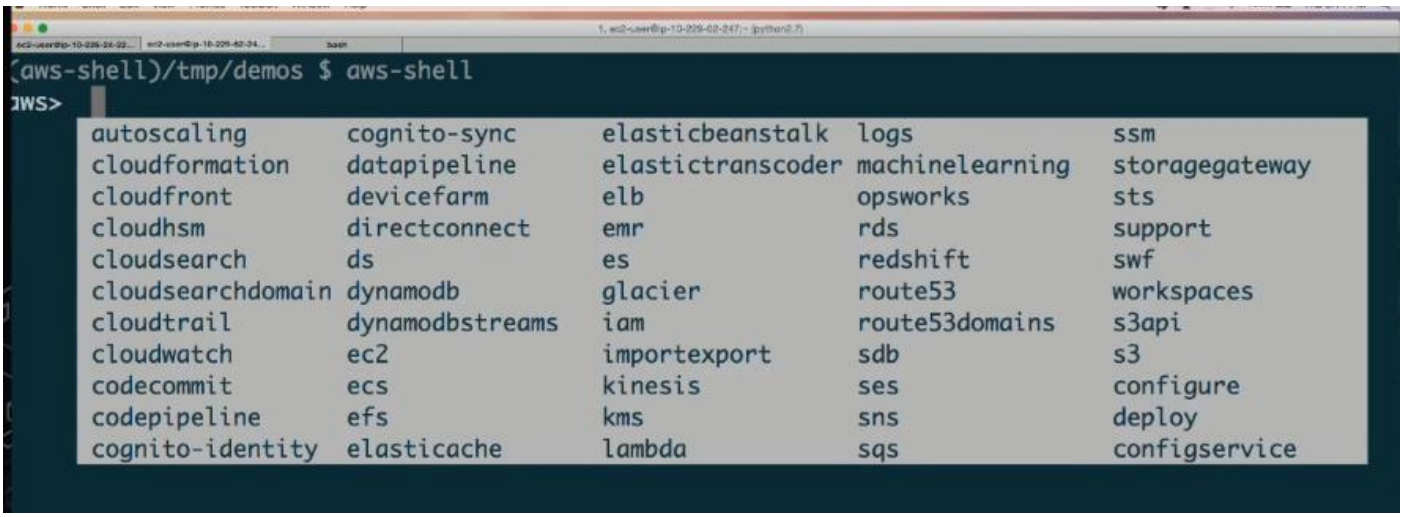
```
export AWS_CONFIG_FILE                  $HOME/.aws/config

export AWS_SHARED_CREDENTIALS_FILE      $HOME/.aws/credentials
```

- Explicitly set `AWS_CONFIG_FILE/AWS_SHARED_CREDENTIALS_FILE`
- Ensure user running cron job has `~/.aws/config ~/.aws/credentials`

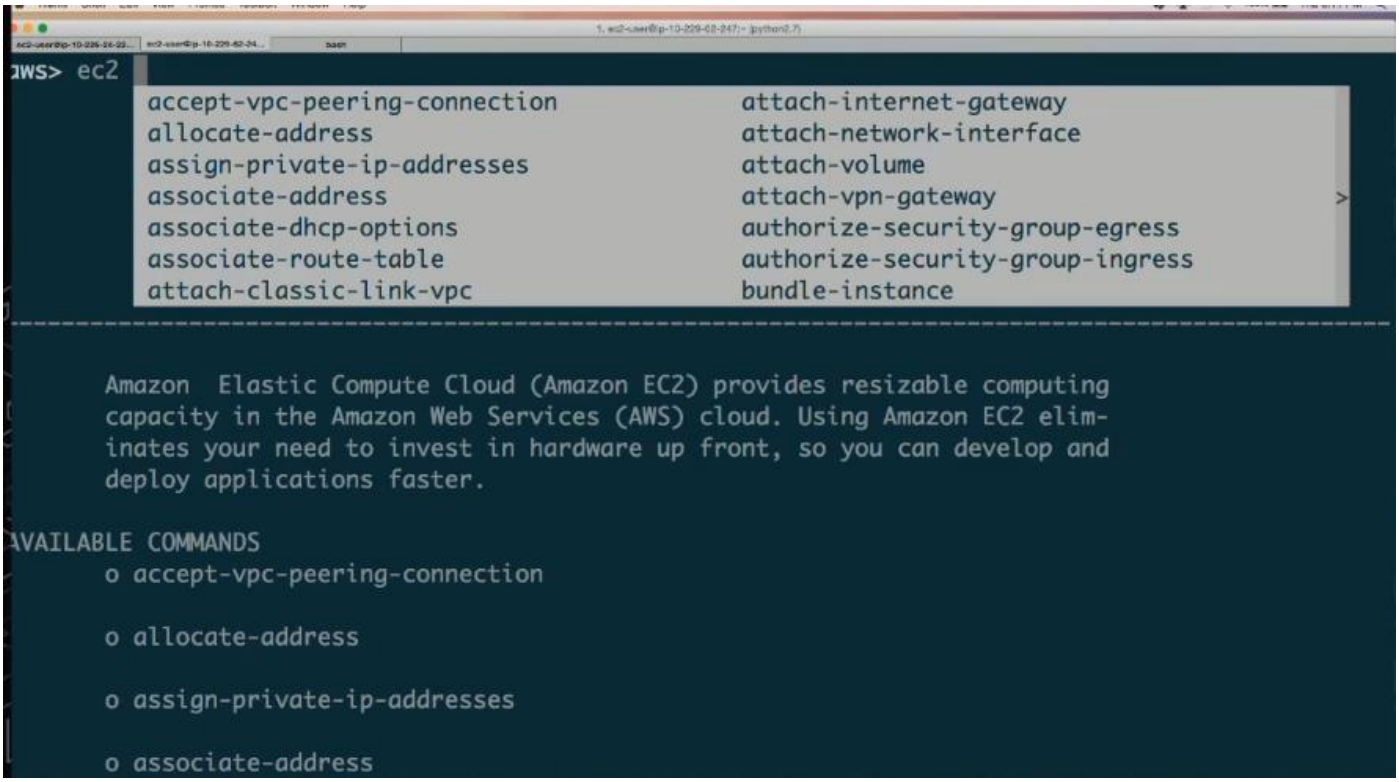
Configure a region

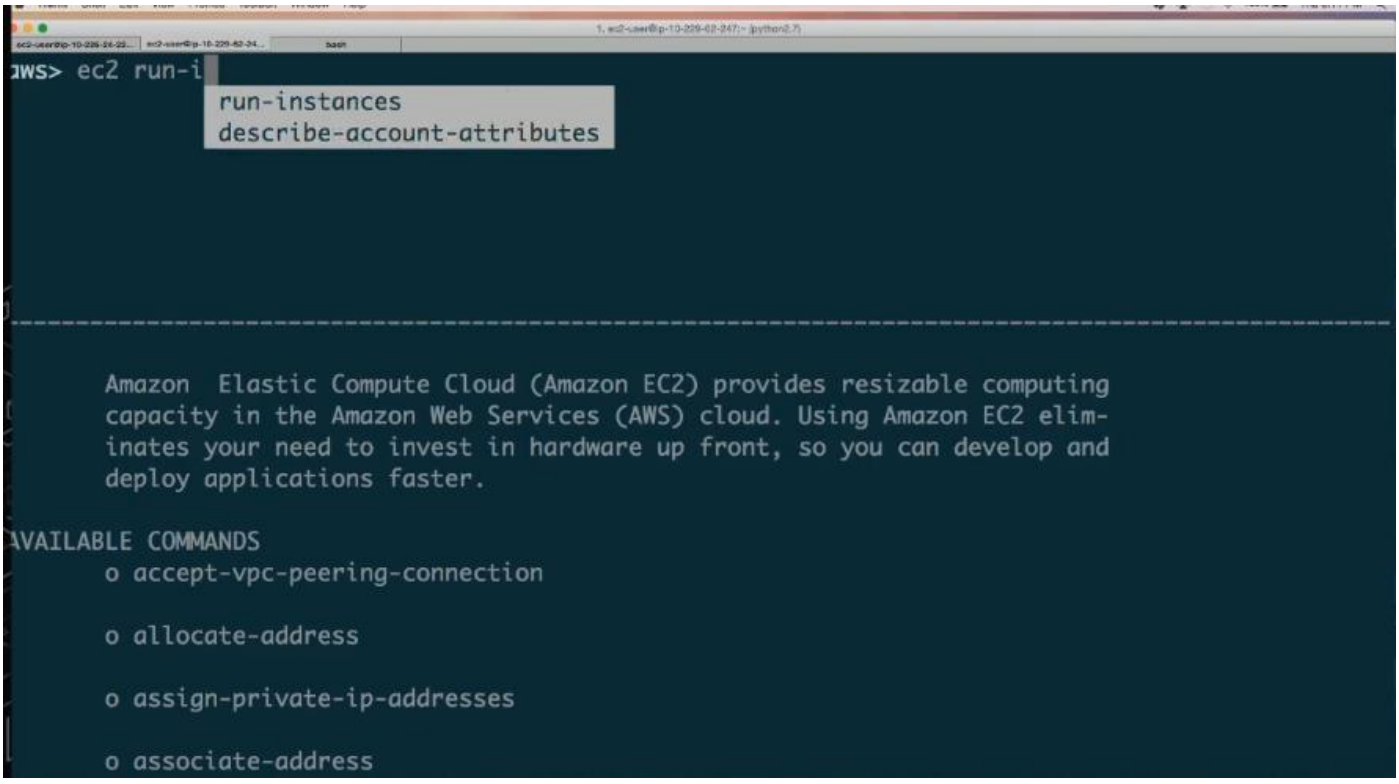
```
aws configure set region \
 $(curl -s http://169.254.169.254/latest/dynamic/instance-identity/document \
 | jp -u 'region')
```

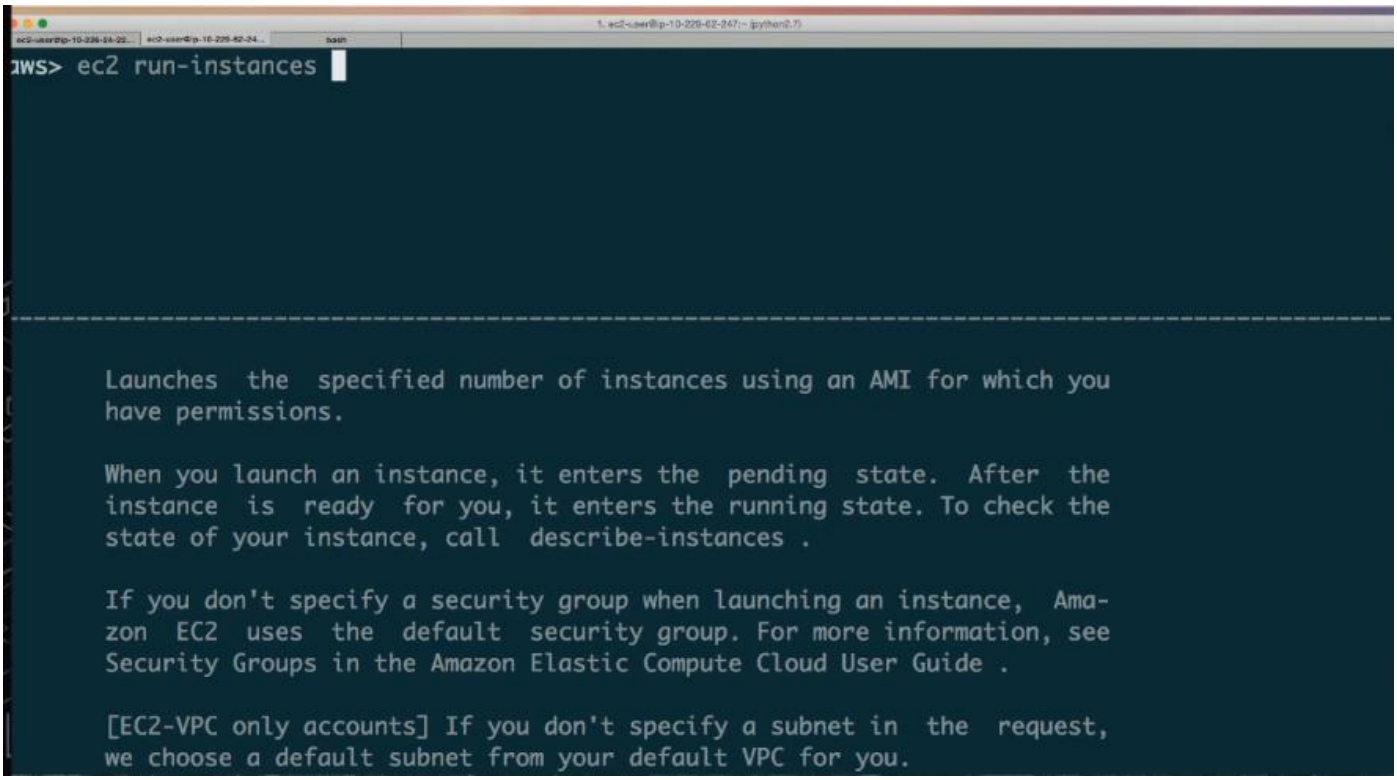
The CLI needs credentials and a region in order to be able to run commands successfully. This is a way to get all your Cron jobs set up.

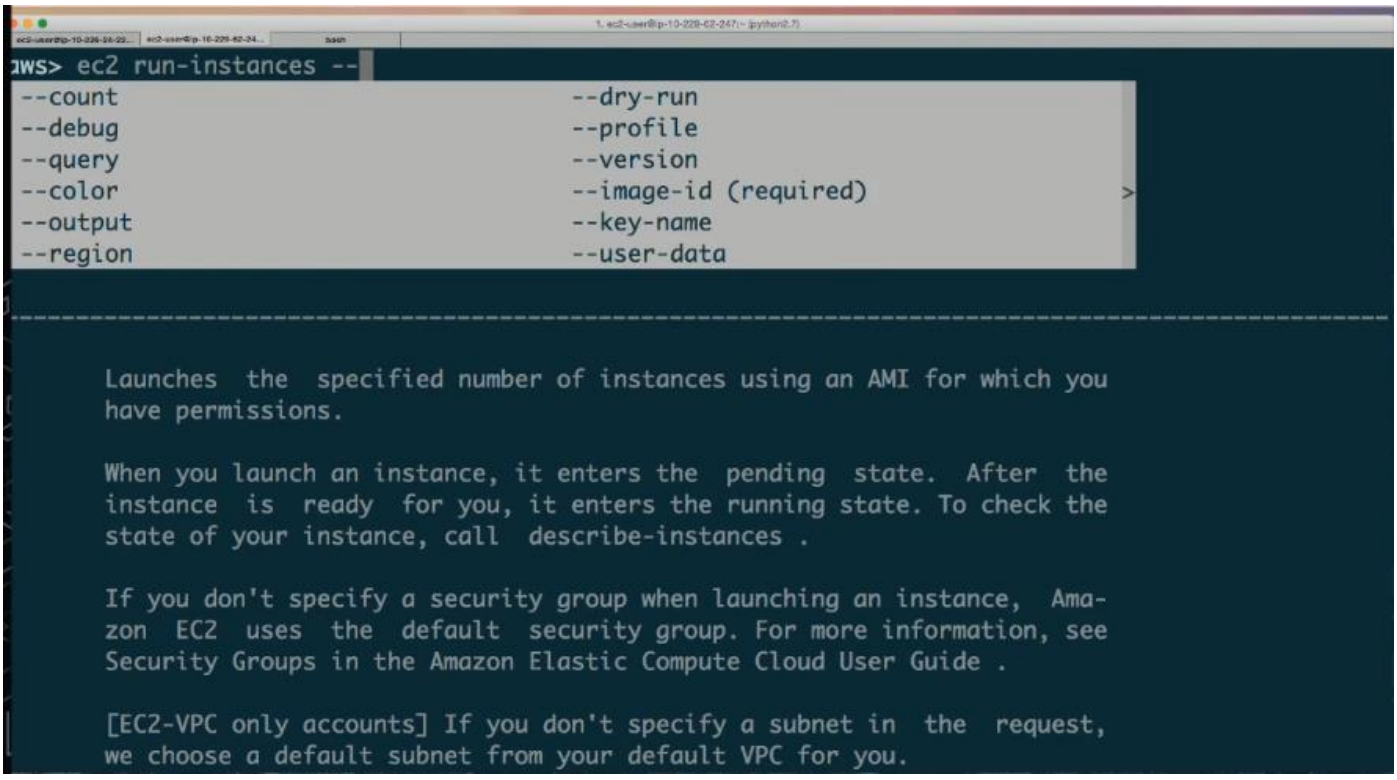


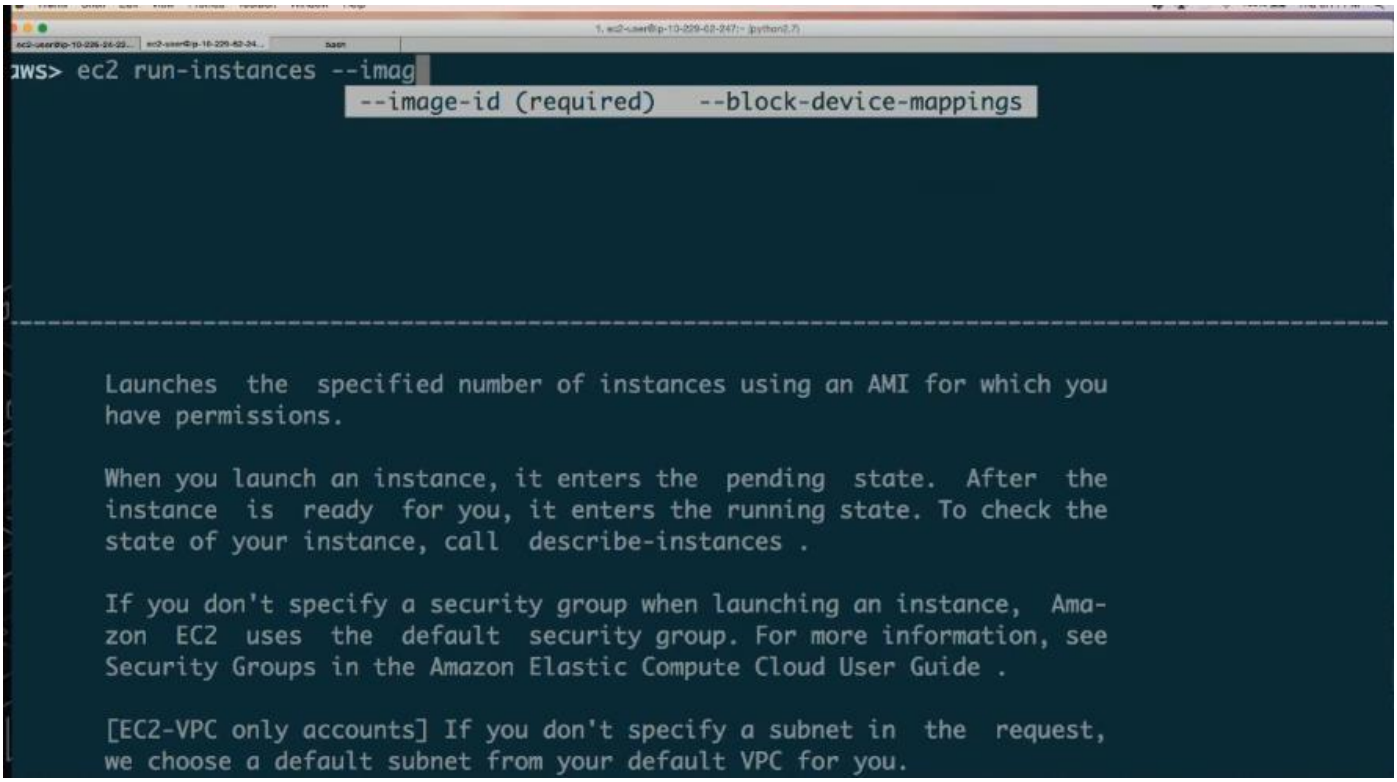
This ***aws-shell*** project addresses some of the problems that users face when trying to figure out what shell commands to use in their scripts.

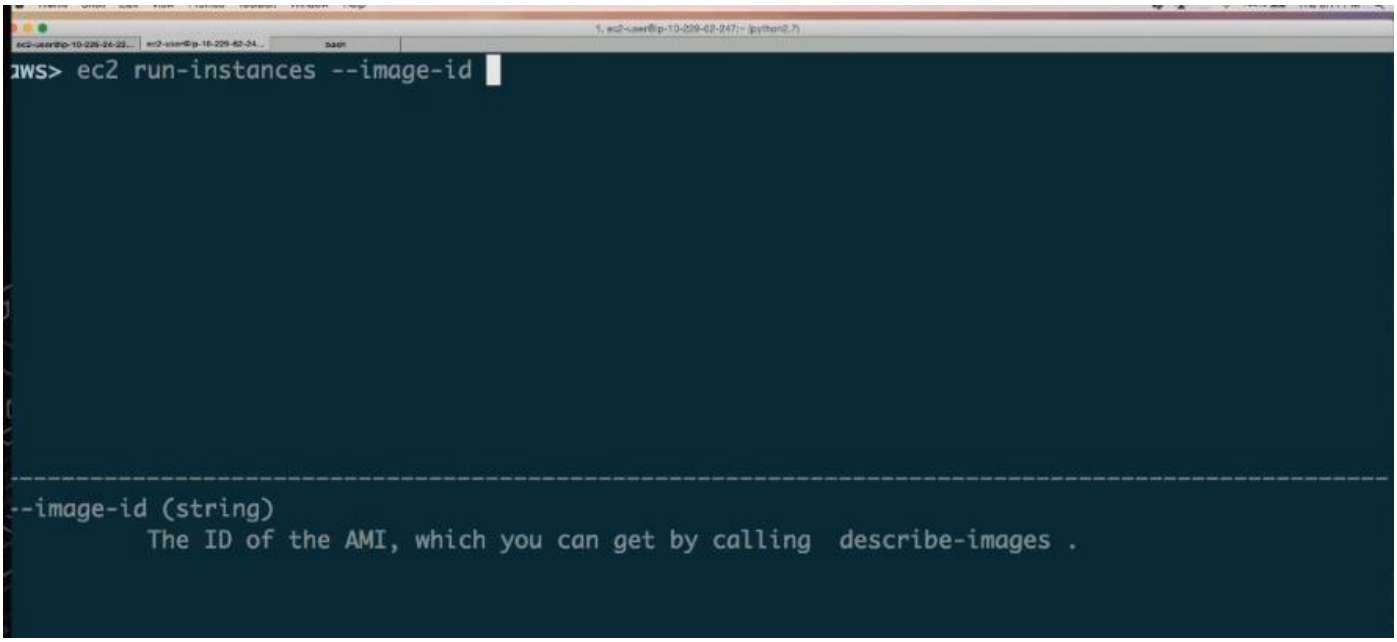

aws> ec2
accept-vpc-peering-connection
allocate-address
assign-private-ip-addresses
associate-address
associate-dhcp-options
associate-route-table
attach-classic-link-vpc
attach-internet-gateway
attach-network-interface
attach-volume
attach-vpn-gateway
authorize-security-group-egress
authorize-security-group-ingress
bundle-instance
Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster.
AVAILABLE COMMANDS
o accept-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

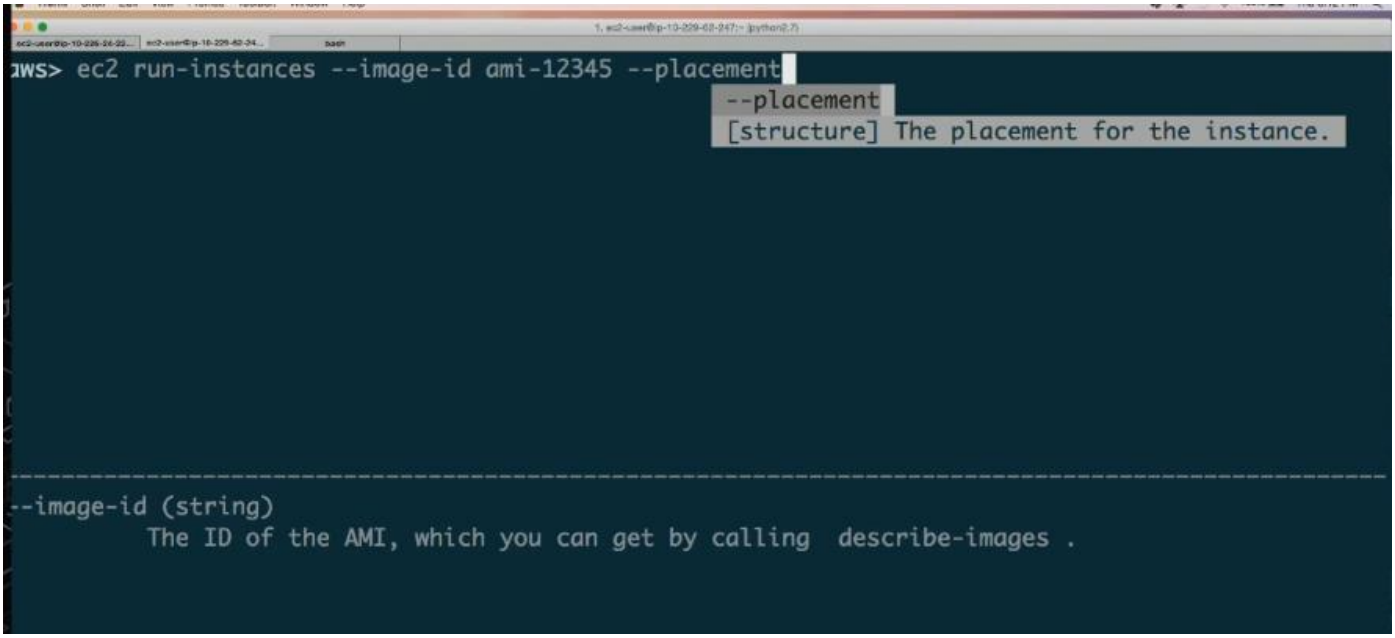

aws> ec2 run-i
run-instances
describe-account-attributes
Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster.
AVAILABLE COMMANDS
o accept-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

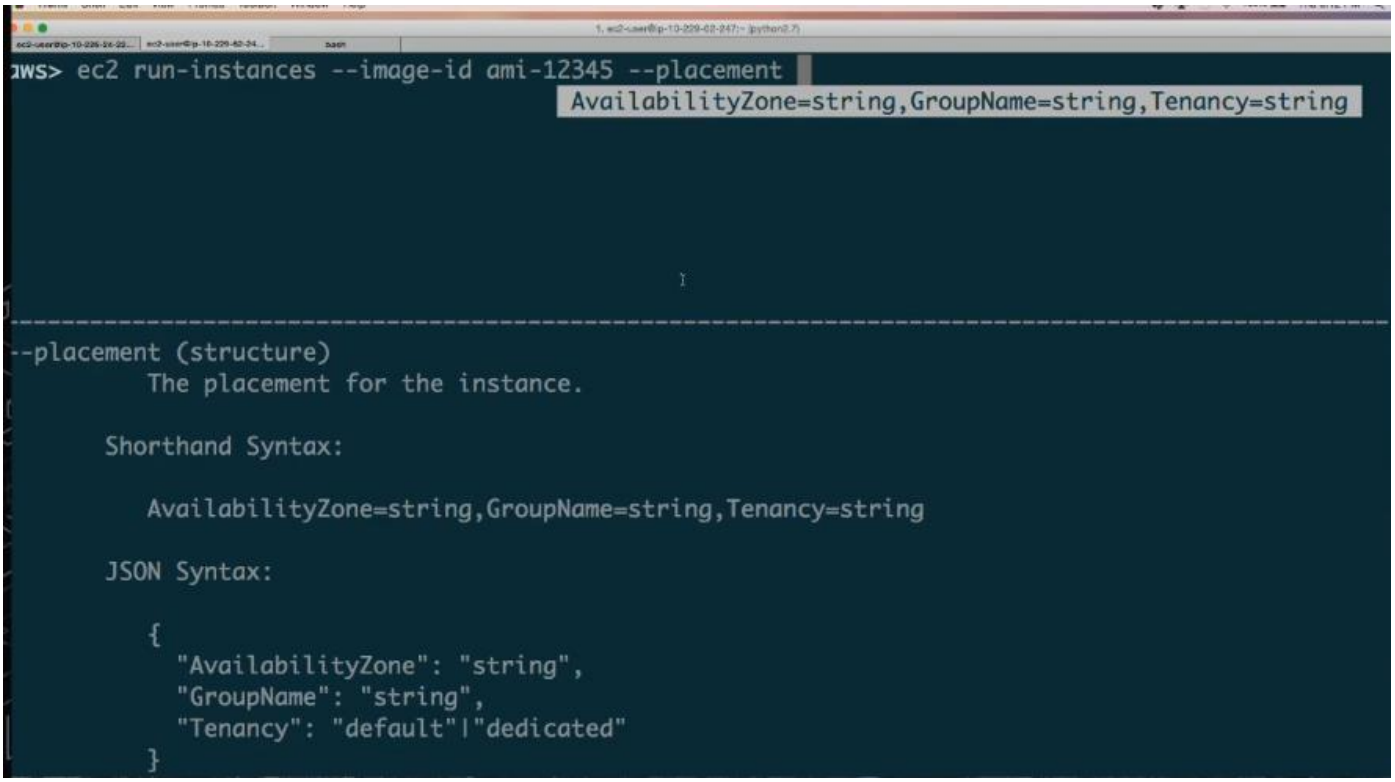

aws> ec2 run-instances
Launches the specified number of instances using an AMI for which you have permissions.
When you launch an instance, it enters the pending state. After the instance is ready for you, it enters the running state. To check the state of your instance, call describe-instances .
If you don't specify a security group when launching an instance, Amazon EC2 uses the default security group. For more information, see Security Groups in the Amazon Elastic Compute Cloud User Guide .
[EC2-VPC only accounts] If you don't specify a subnet in the request, we choose a default subnet from your default VPC for you.

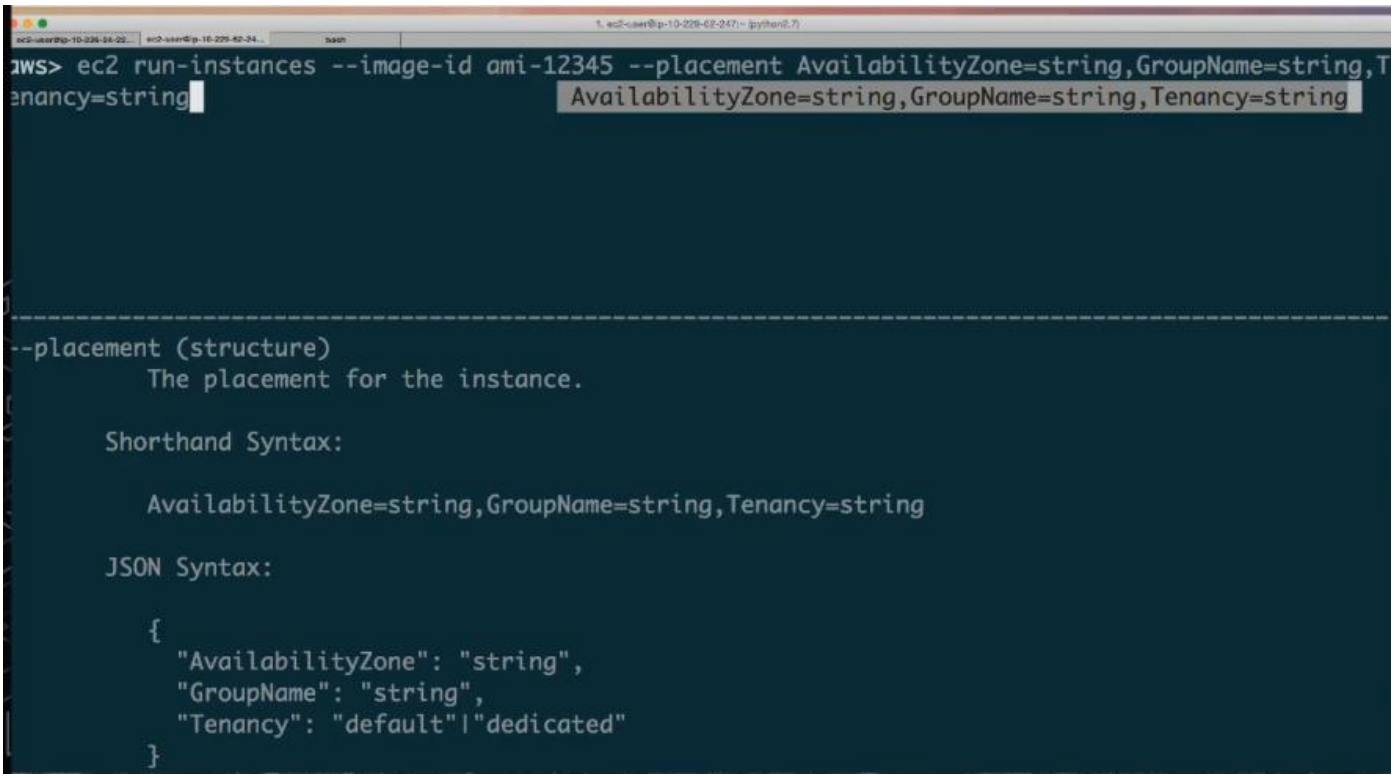

aws> ec2 run-instances --
--count
--debug
--query
--color
--output
--region
--dry-run
--profile
--version
--image-id (required)
--key-name
--user-data
Launches the specified number of instances using an AMI for which you have permissions.
When you launch an instance, it enters the pending state. After the instance is ready for you, it enters the running state. To check the state of your instance, call describe-instances .
If you don't specify a security group when launching an instance, Amazon EC2 uses the default security group. For more information, see Security Groups in the Amazon Elastic Compute Cloud User Guide .
[EC2-VPC only accounts] If you don't specify a subnet in the request, we choose a default subnet from your default VPC for you.

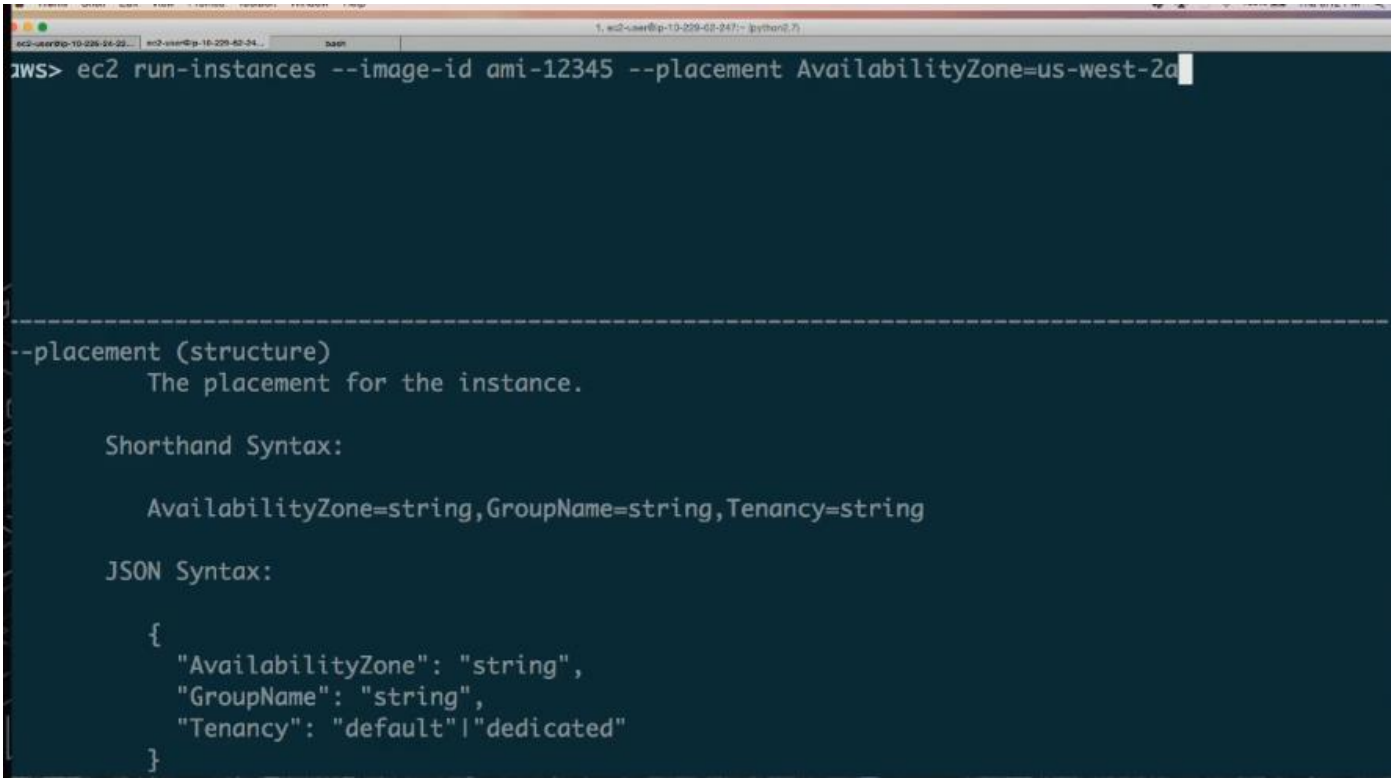

aws> ec2 run-instances --imag
--image-id (required) --block-device-mappings
Launches the specified number of instances using an AMI for which you have permissions.
When you launch an instance, it enters the pending state. After the instance is ready for you, it enters the running state. To check the state of your instance, call describe-instances .
If you don't specify a security group when launching an instance, Amazon EC2 uses the default security group. For more information, see Security Groups in the Amazon Elastic Compute Cloud User Guide .
[EC2-VPC only accounts] If you don't specify a subnet in the request, we choose a default subnet from your default VPC for you.

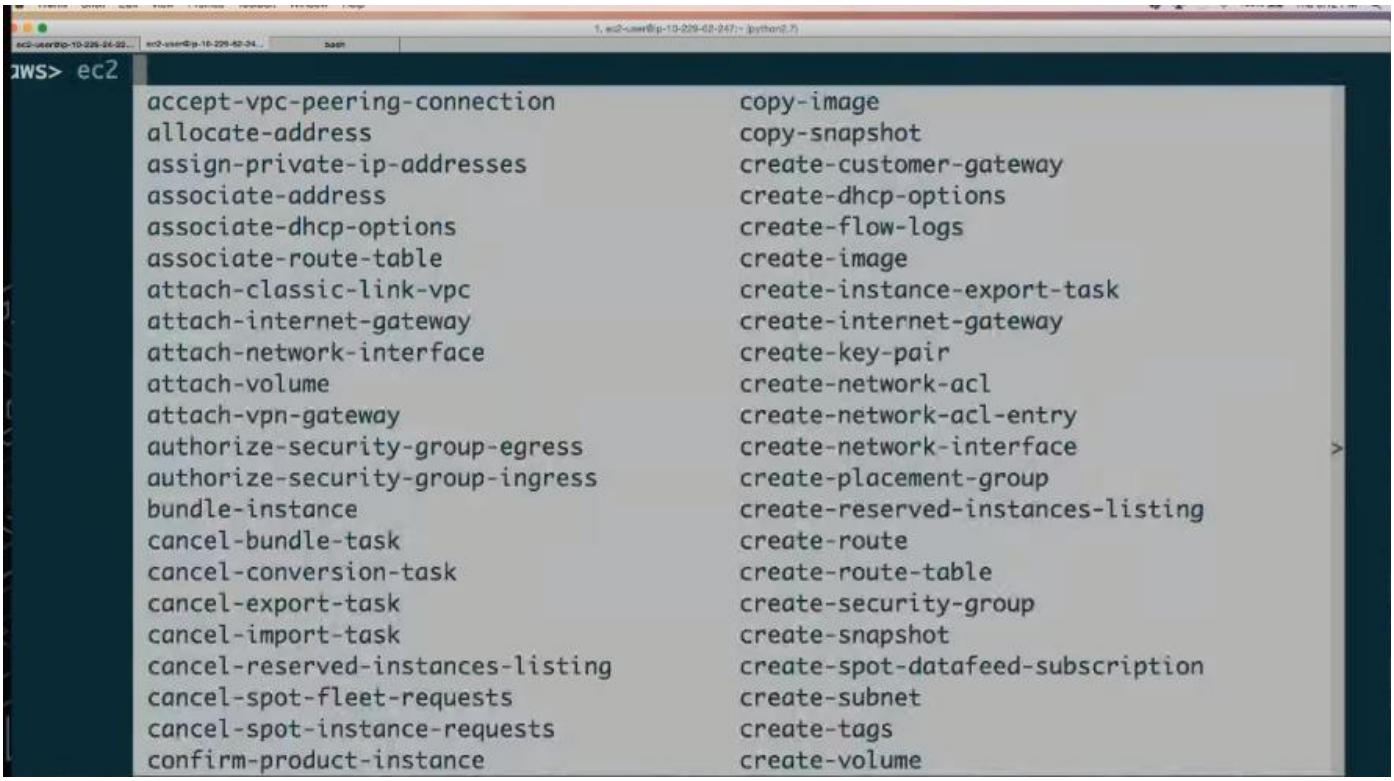

aws> ec2 run-instances --image-id
--image-id (string)
The ID of the AMI, which you can get by calling describe-images .

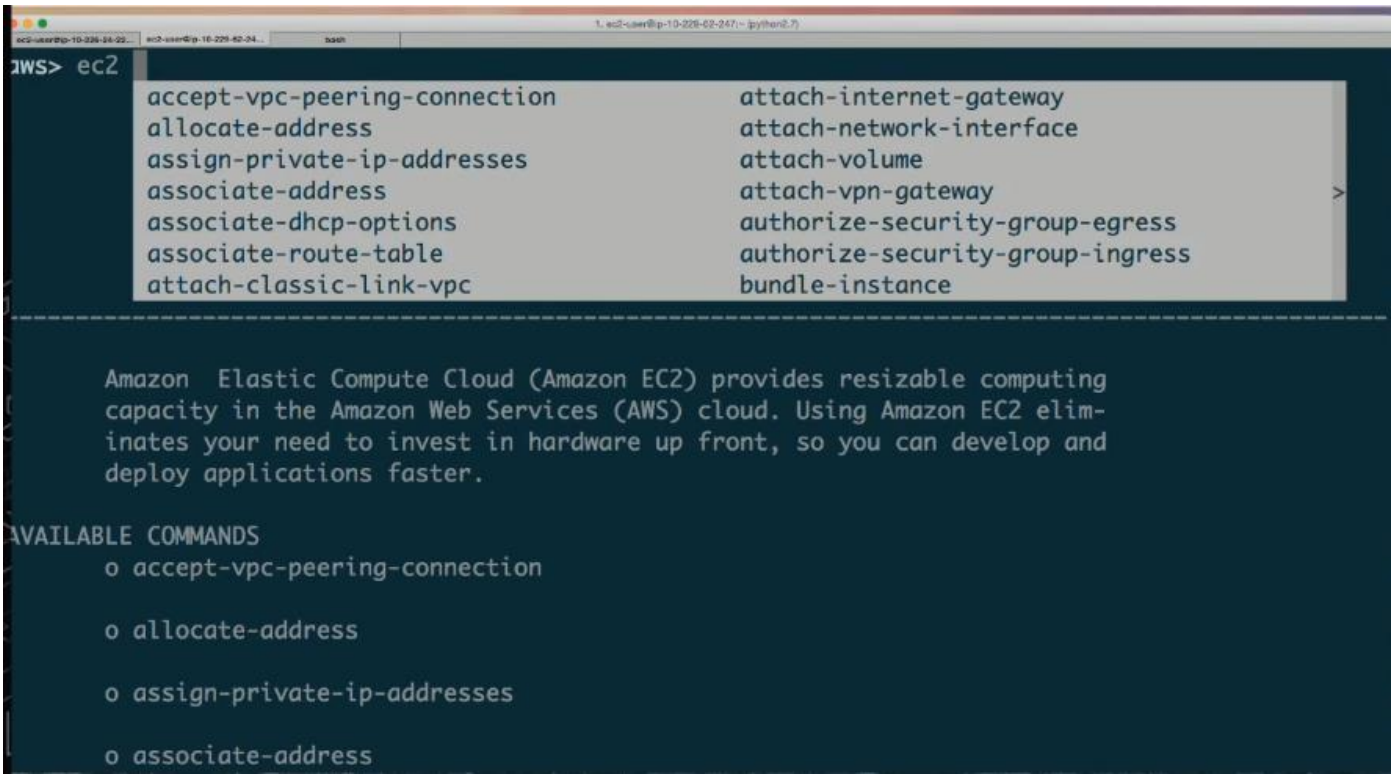

aws> ec2 run-instances --image-id ami-12345 --placement
--placement
[structure] The placement for the instance.
--image-id (string)
The ID of the AMI, which you can get by calling describe-images .

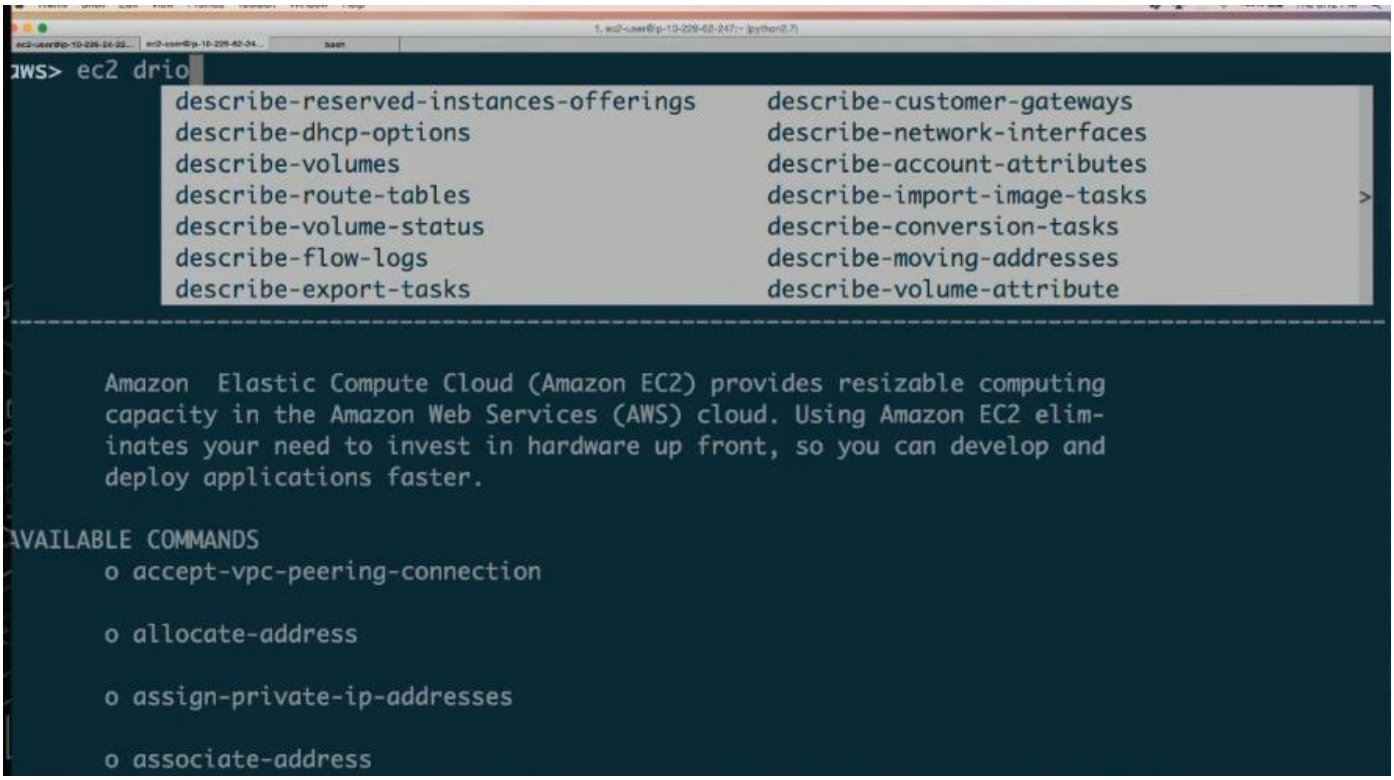

aws> ec2 run-instances --image-id ami-12345 --placement
AvailabilityZone=string,GroupName=string,Tenancy=string
--placement (structure)
The placement for the instance.
Shorthand Syntax:
AvailabilityZone=string,GroupName=string,Tenancy=string
JSON Syntax:
{
"AvailabilityZone": "string",
"GroupName": "string",
"Tenancy": "default"|"dedicated"
}

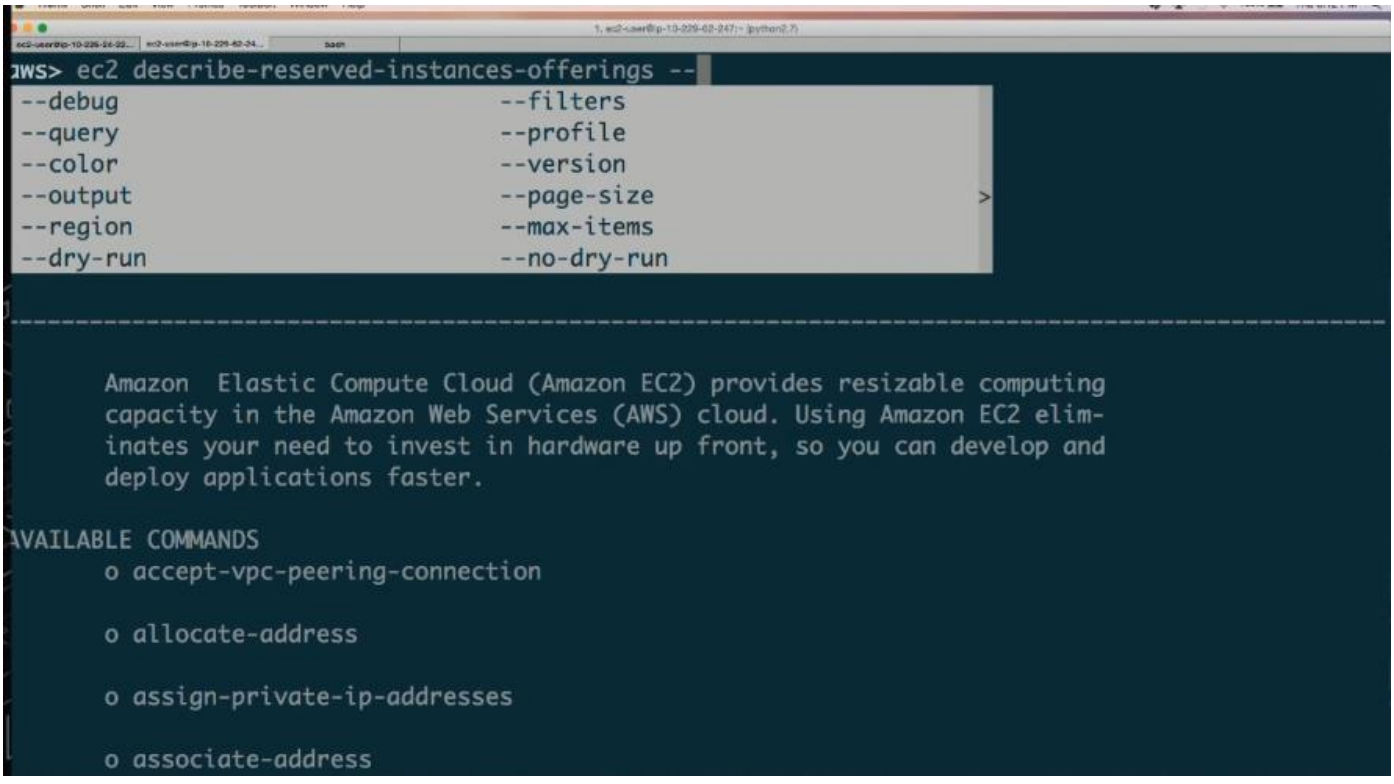

aws> ec2 run-instances --image-id ami-12345 --placement AvailabilityZone=string,GroupName=string,T
enancy=string
AvailabilityZone=string,GroupName=string,Tenancy=string
--placement (structure)
The placement for the instance.
Shorthand Syntax:
AvailabilityZone=string,GroupName=string,Tenancy=string
JSON Syntax:
{
"AvailabilityZone": "string",
"GroupName": "string",
"Tenancy": "default"|"dedicated"
}

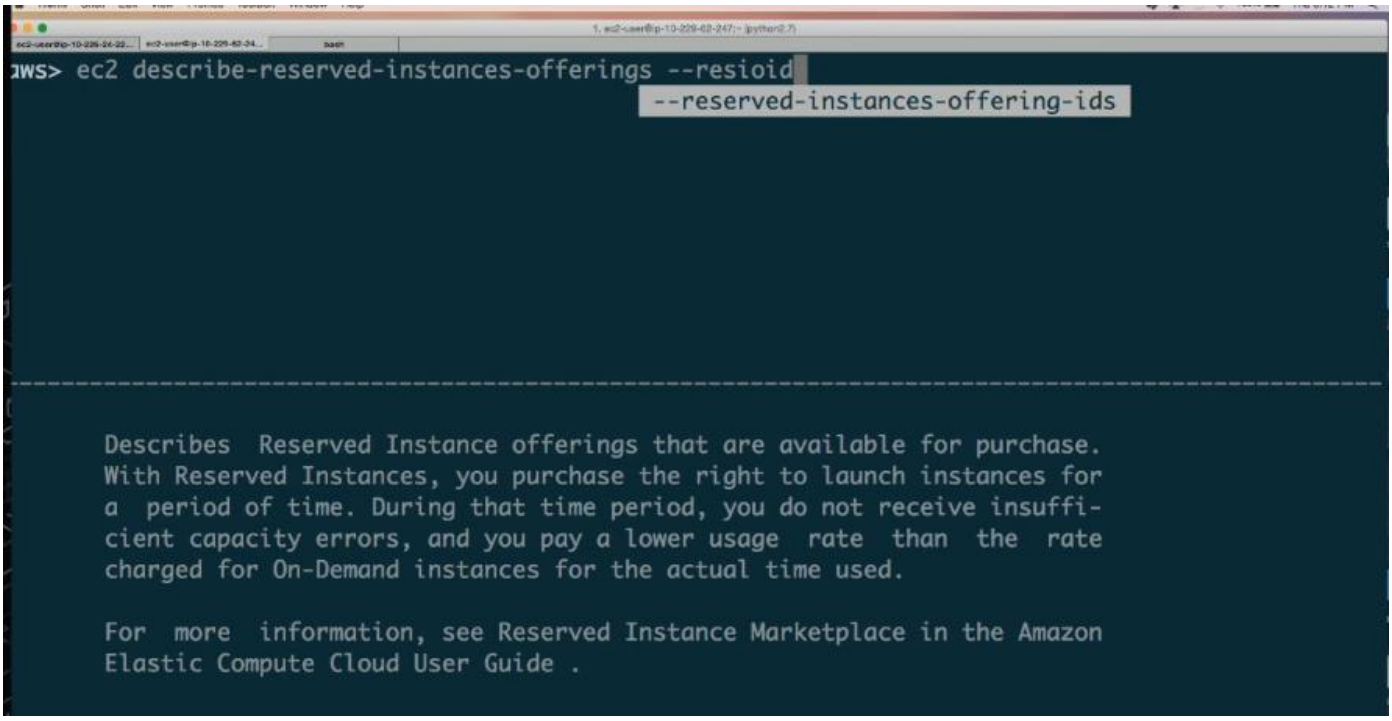

aws> ec2 run-instances --image-id ami-12345 --placement AvailabilityZone=us-west-2a
--placement (structure)
The placement for the instance.
Shorthand Syntax:
AvailabilityZone=string,GroupName=string,Tenancy=string
JSON Syntax:
{
"AvailabilityZone": "string",
"GroupName": "string",
"Tenancy": "default"|"dedicated"
}

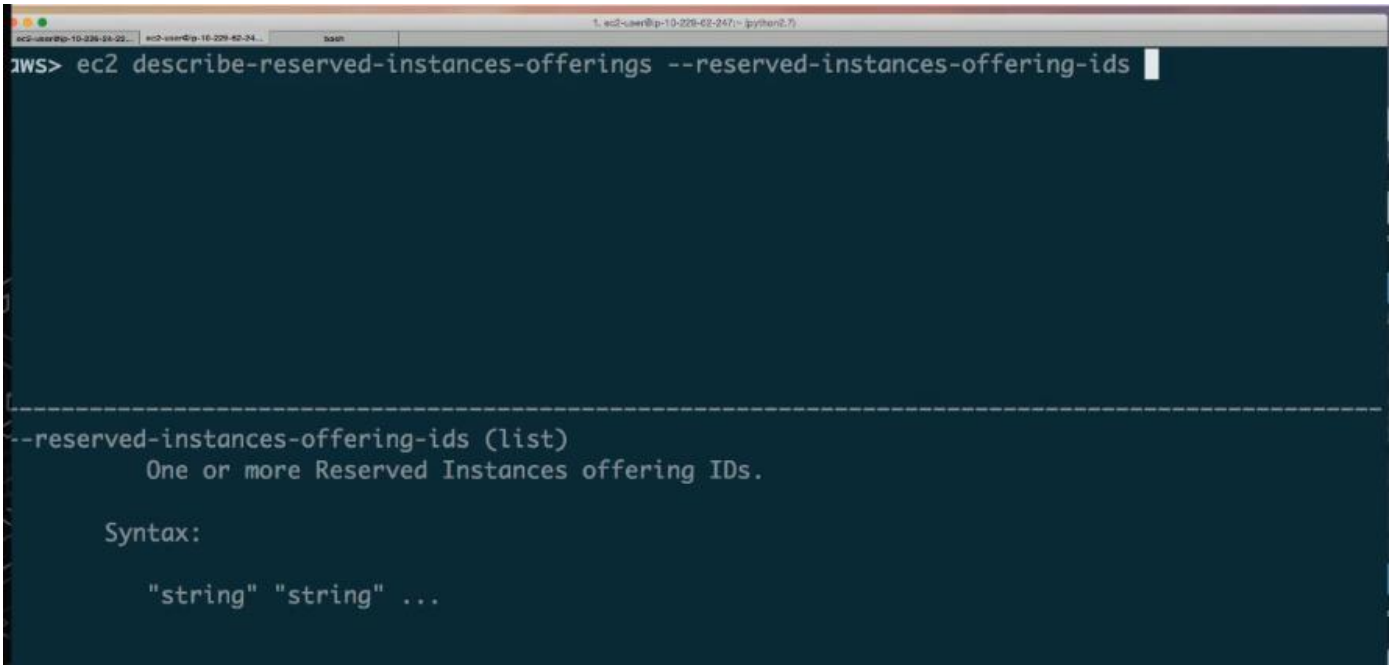

aws> ec2
accept-vpc-peering-connection
allocate-address
assign-private-ip-addresses
associate-address
associate-dhcp-options
associate-route-table
attach-classic-link-vpc
attach-internet-gateway
attach-network-interface
attach-volume
attach-vpn-gateway
authorize-security-group-egress
authorize-security-group-ingress
bundle-instance
cancel-bundle-task
cancel-conversion-task
cancel-export-task
cancel-import-task
cancel-reserved-instances-listing
cancel-spot-fleet-requests
cancel-spot-instance-requests
confirm-product-instance
copy-image
copy-snapshot
create-customer-gateway
create-dhcp-options
create-flow-logs
create-image
create-instance-export-task
create-internet-gateway
create-key-pair
create-network-acl
create-network-acl-entry
create-network-interface
create-placement-group
create-reserved-instances-listing
create-route
create-route-table
create-security-group
create-snapshot
create-spot-datafeed-subscription
create-subnet
create-tags
create-volume

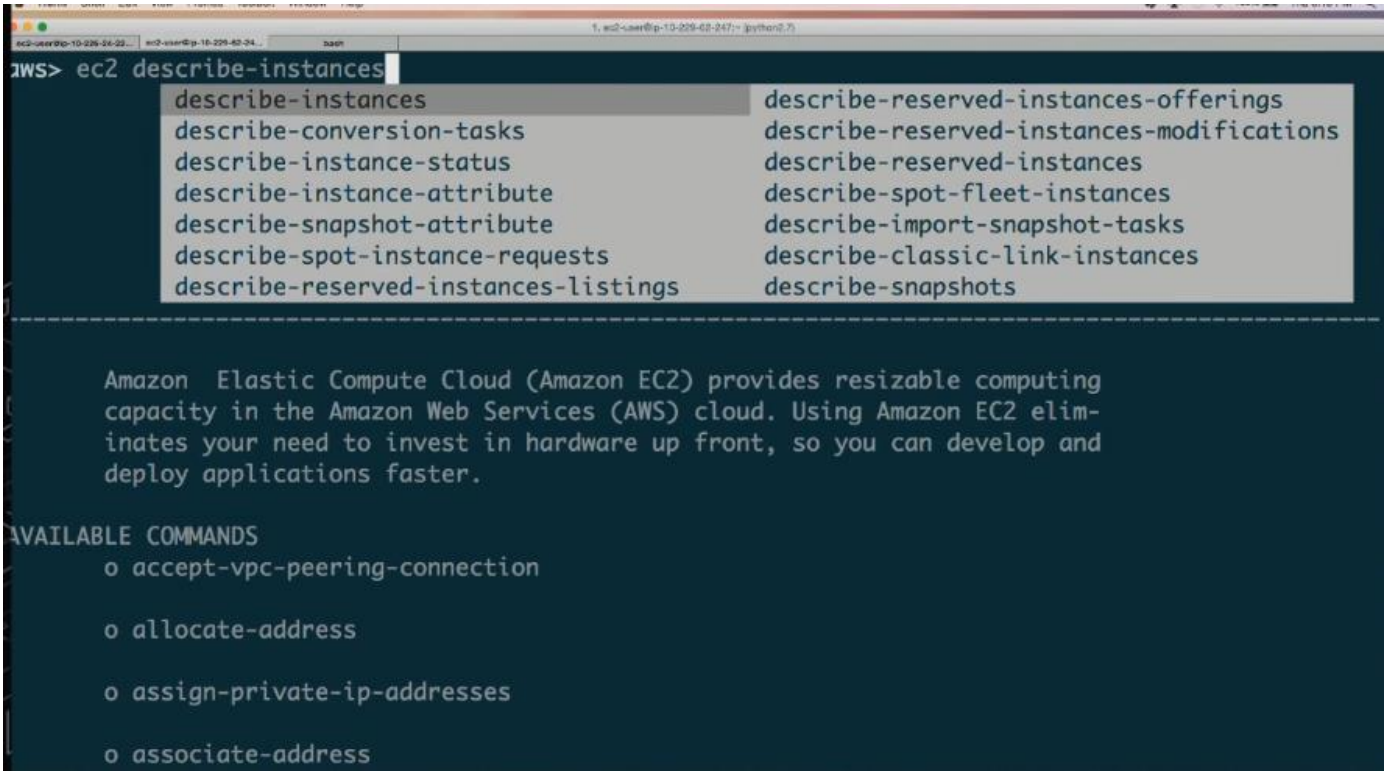

aws> ec2
accept-vpc-peering-connection
allocate-address
assign-private-ip-addresses
associate-address
associate-dhcp-options
associate-route-table
attach-classic-link-vpc
attach-internet-gateway
attach-network-interface
attach-volume
attach-vpn-gateway
authorize-security-group-egress
authorize-security-group-ingress
bundle-instance
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-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

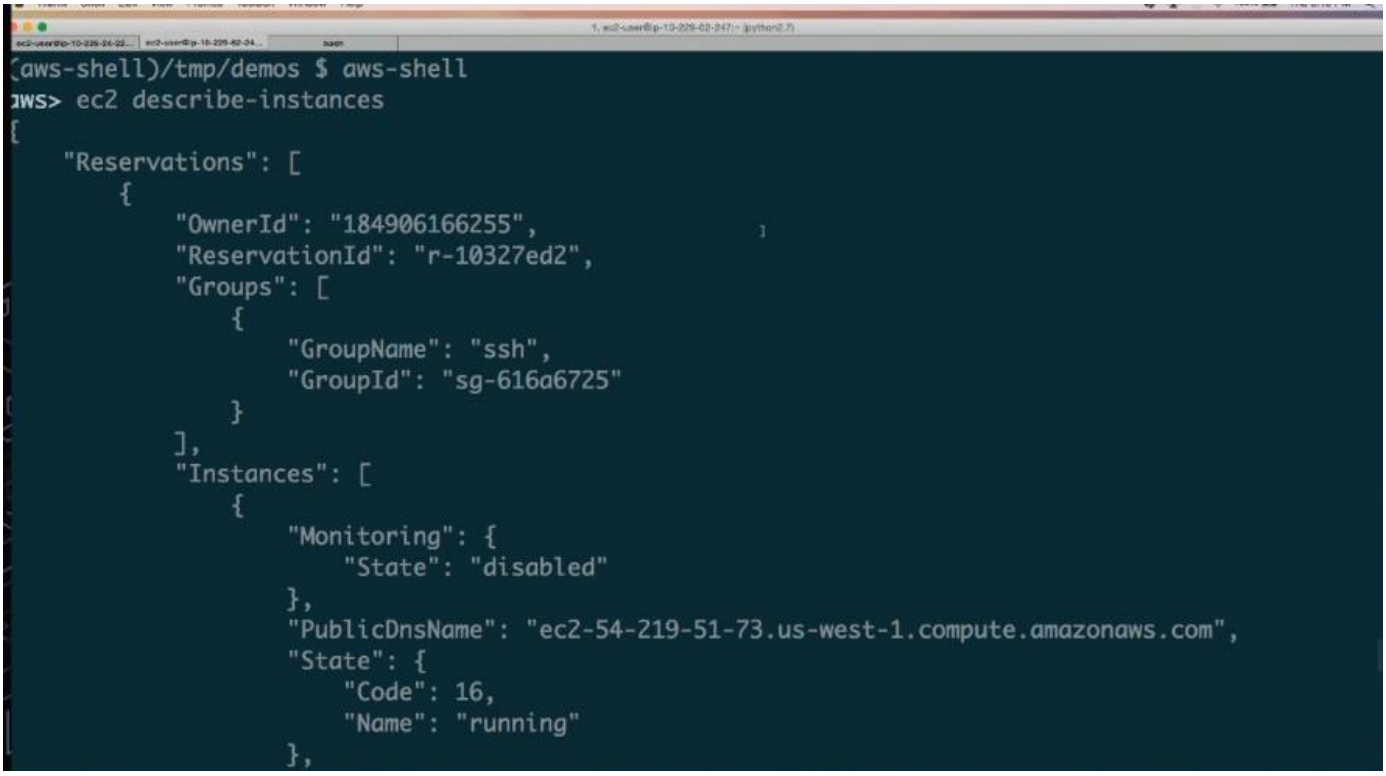

aws> ec2 drio
describe-reserved-instances-offerings
describe-dhcp-options
describe-volumes
describe-route-tables
describe-volume-status
describe-flow-logs
describe-export-tasks
describe-customer-gateways
describe-network-interfaces
describe-account-attributes
describe-import-image-tasks
describe-conversion-tasks
describe-moving-addresses
describe-volume-attribute
Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster.
AVAILABLE COMMANDS
o accept-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

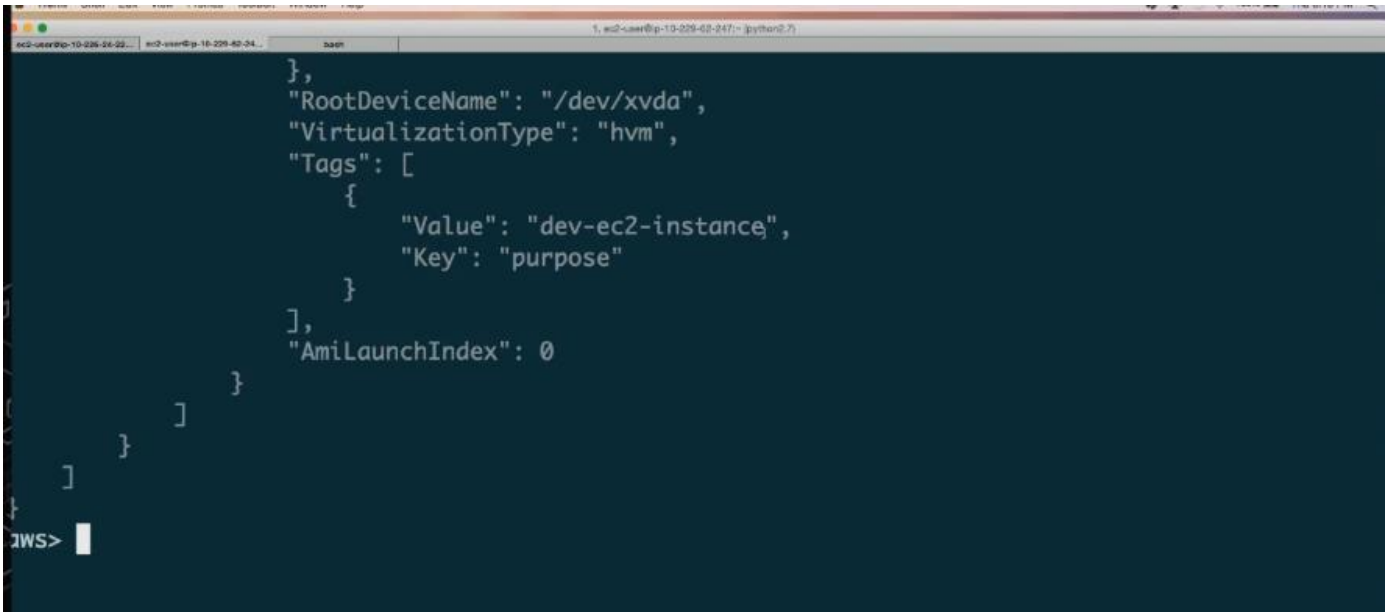

aws> ec2 describe-reserved-instances-offerings --
--debug
--query
--color
--output
--region
--dry-run
--filters
--profile
--version
--page-size
--max-items
--no-dry-run
Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster.
AVAILABLE COMMANDS
o accept-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

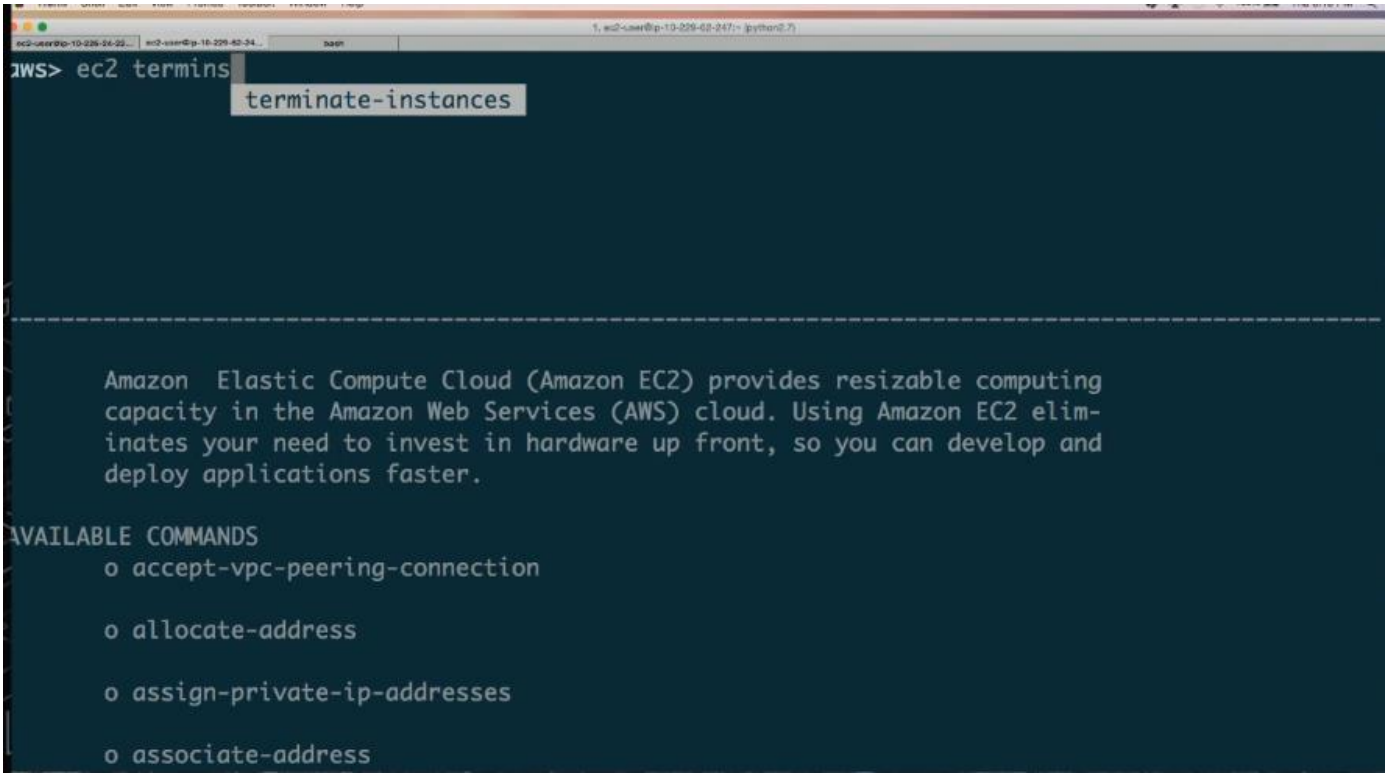

aws> ec2 describe-reserved-instances-offerings --resioid
--reserved-instances-offering-ids
Describes Reserved Instance offerings that are available for purchase. With Reserved Instances, you purchase the right to launch instances for a period of time. During that time period, you do not receive insufficient capacity errors, and you pay a lower usage rate than the rate charged for On-Demand instances for the actual time used.
For more information, see Reserved Instance Marketplace in the Amazon Elastic Compute Cloud User Guide .

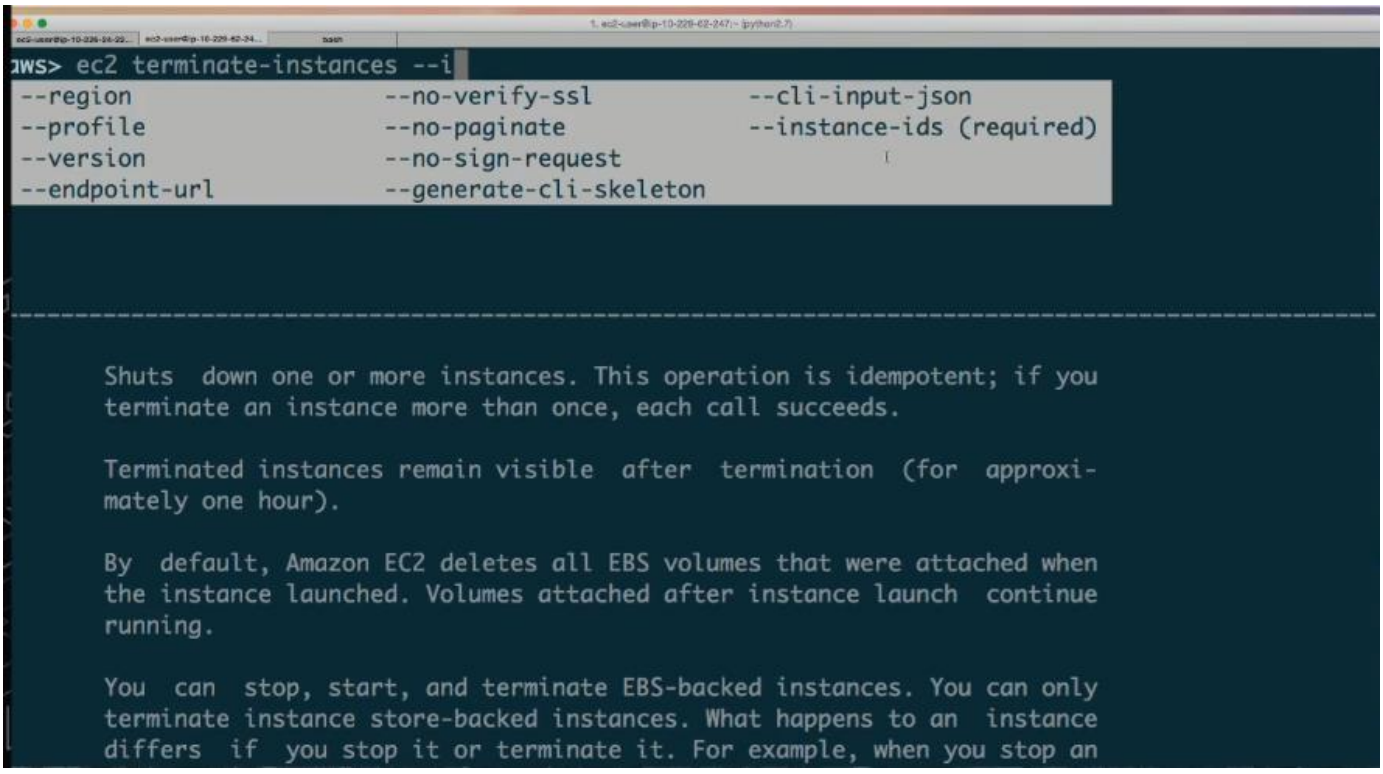

aws> ec2 describe-reserved-instances-offerings --reserved-instances-offering-ids
--reserved-instances-offering-ids (list)
One or more Reserved Instances offering IDs.
Syntax:
"string" "string" ...

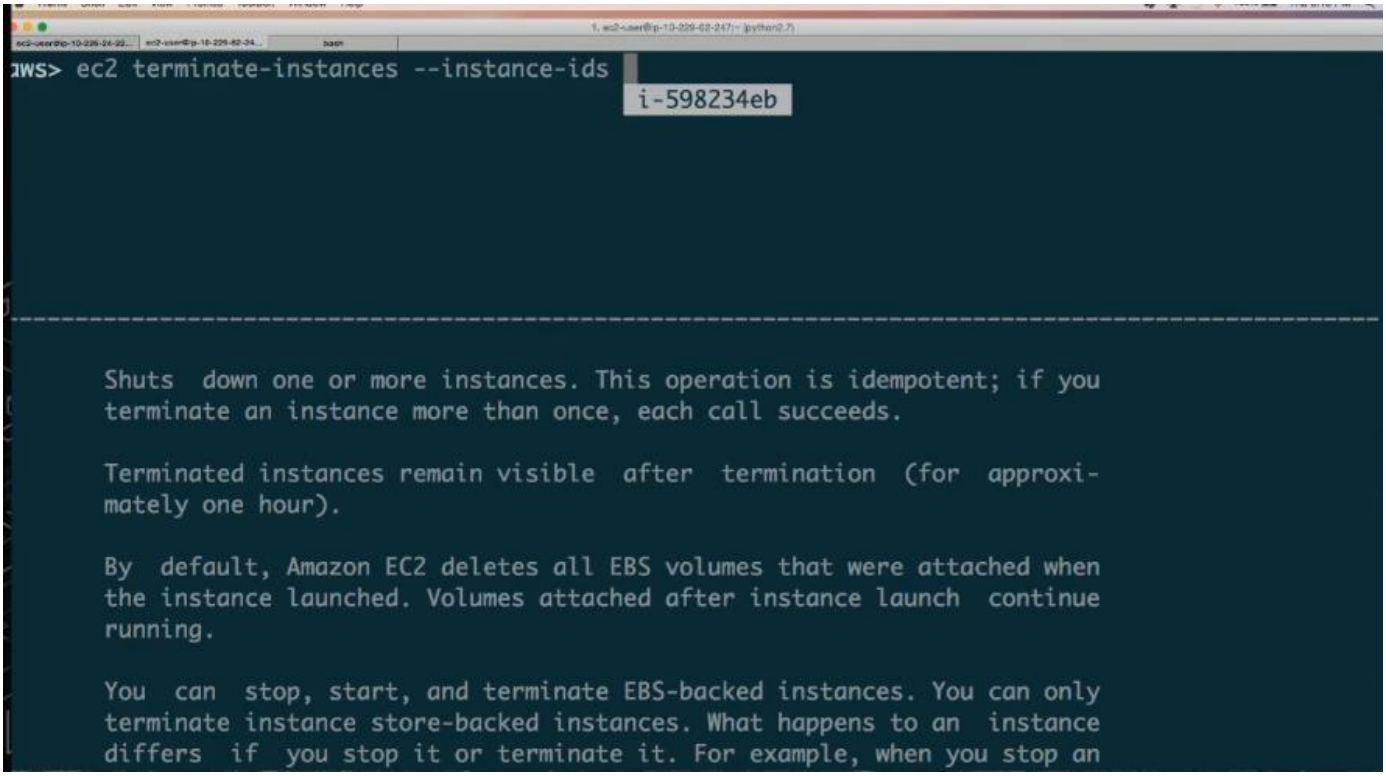

aws> ec2 describe-instances
describe-instances
describe-conversion-tasks
describe-instance-status
describe-instance-attribute
describe-snapshot-attribute
describe-spot-instance-requests
describe-reserved-instances-listings
describe-reserved-instances-offerings
describe-reserved-instances-modifications
describe-reserved-instances
describe-spot-fleet-instances
describe-import-snapshot-tasks
describe-classic-link-instances
describe-snapshots
Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster.
AVAILABLE COMMANDS
o accept-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

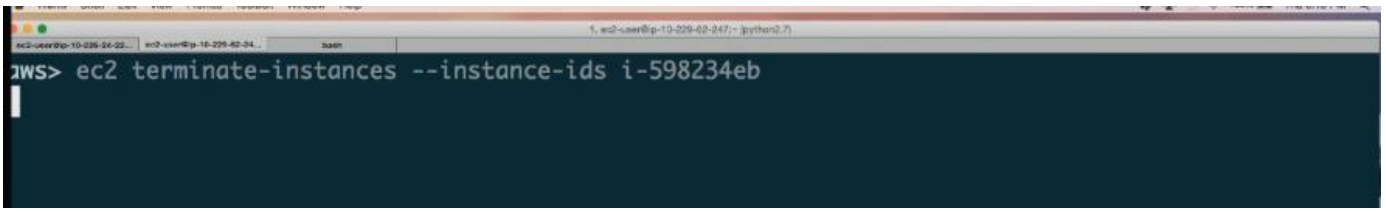

(aws-shell)/tmp/demos $ aws-shell
aws> ec2 describe-instances
{
"Reservations": [
{
"OwnerId": "184906166255",
"ReservationId": "r-10327ed2",
"Groups": [
{
"GroupName": "ssh",
"GroupId": "sg-616a6725"
}
],
"Instances": [
{
"Monitoring": {
"State": "disabled"
},
"PublicDnsName": "ec2-54-219-51-73.us-west-1.compute.amazonaws.com",
"State": {
"Code": 16,
"Name": "running"
},

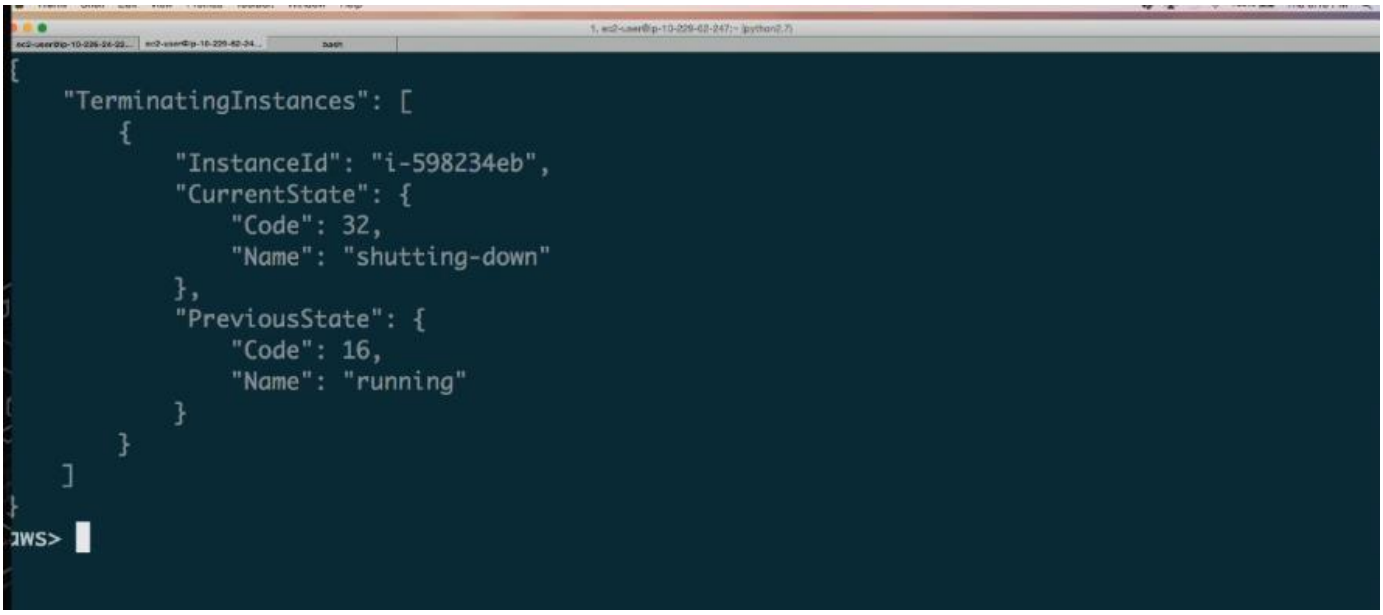

},
"RootDeviceName": "/dev/xvda",
"VirtualizationType": "hvm",
"Tags": [
{
"Value": "dev-ec2-instance",
"Key": "purpose"
}
],
"AmiLaunchIndex": 0
}
]
}
]
}
aws>

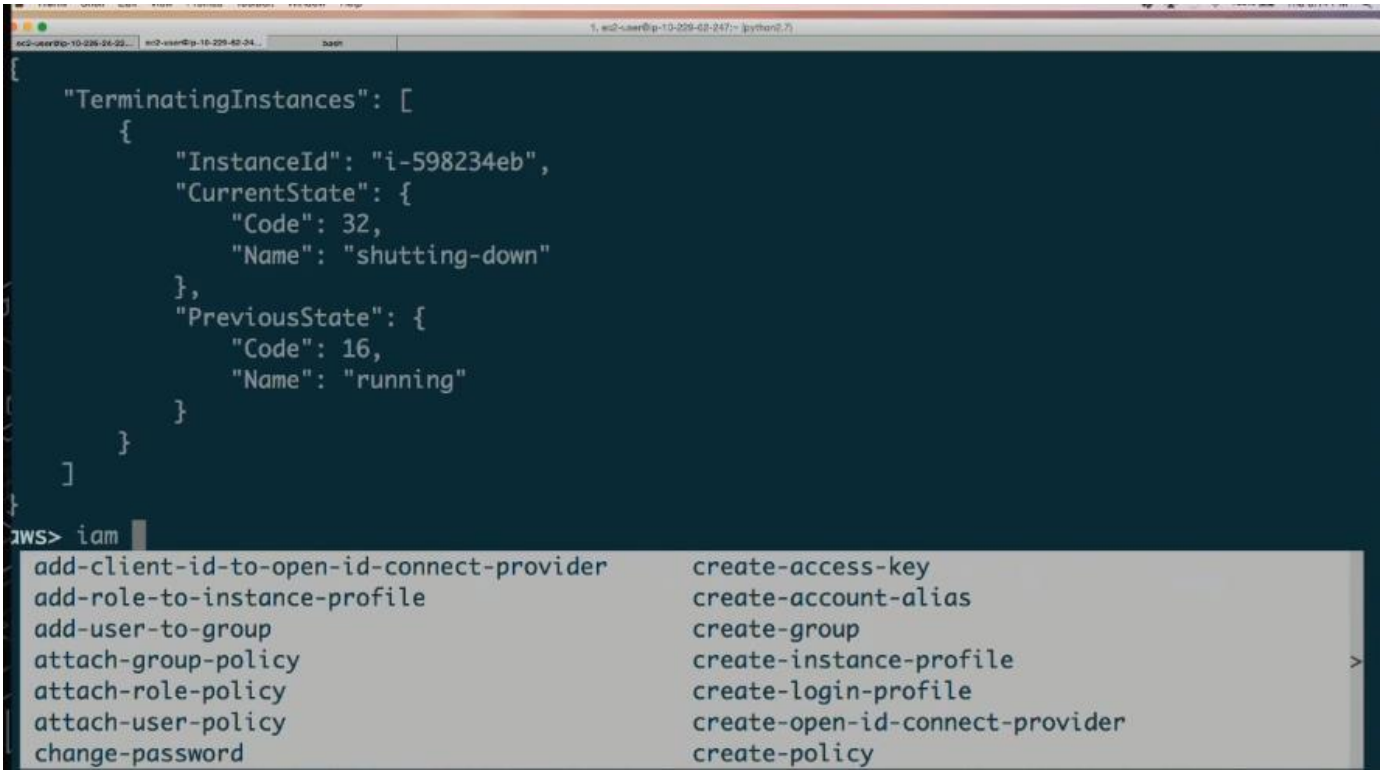

aws> ec2 termins
terminate-instances
Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing
capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 elim-
inates your need to invest in hardware up front, so you can develop and
deploy applications faster.
AVAILABLE COMMANDS
o accept-vpc-peering-connection
o allocate-address
o assign-private-ip-addresses
o associate-address

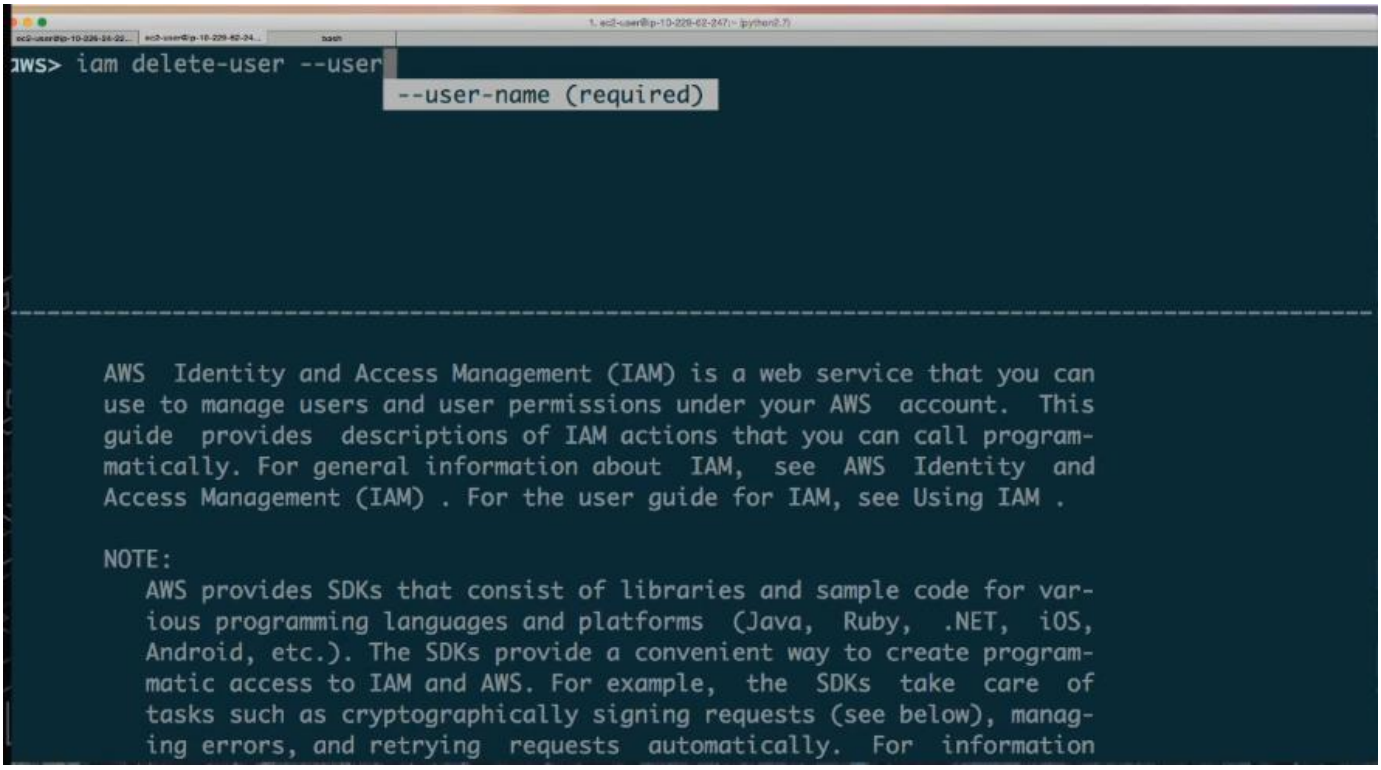

aws> ec2 terminate-instances --i
--region
--profile
--version
--endpoint-url
--no-verify-ssl
--no-paginate
--no-sign-request
--generate-cli-skeleton
--cli-input-json
--instance-ids (required)
Shuts down one or more instances. This operation is idempotent; if you terminate an instance more than once, each call succeeds.
Terminated instances remain visible after termination (for approximately one hour).
By default, Amazon EC2 deletes all EBS volumes that were attached when the instance launched. Volumes attached after instance launch continue running.
You can stop, start, and terminate EBS-backed instances. You can only terminate instance store-backed instances. What happens to an instance differs if you stop it or terminate it. For example, when you stop an

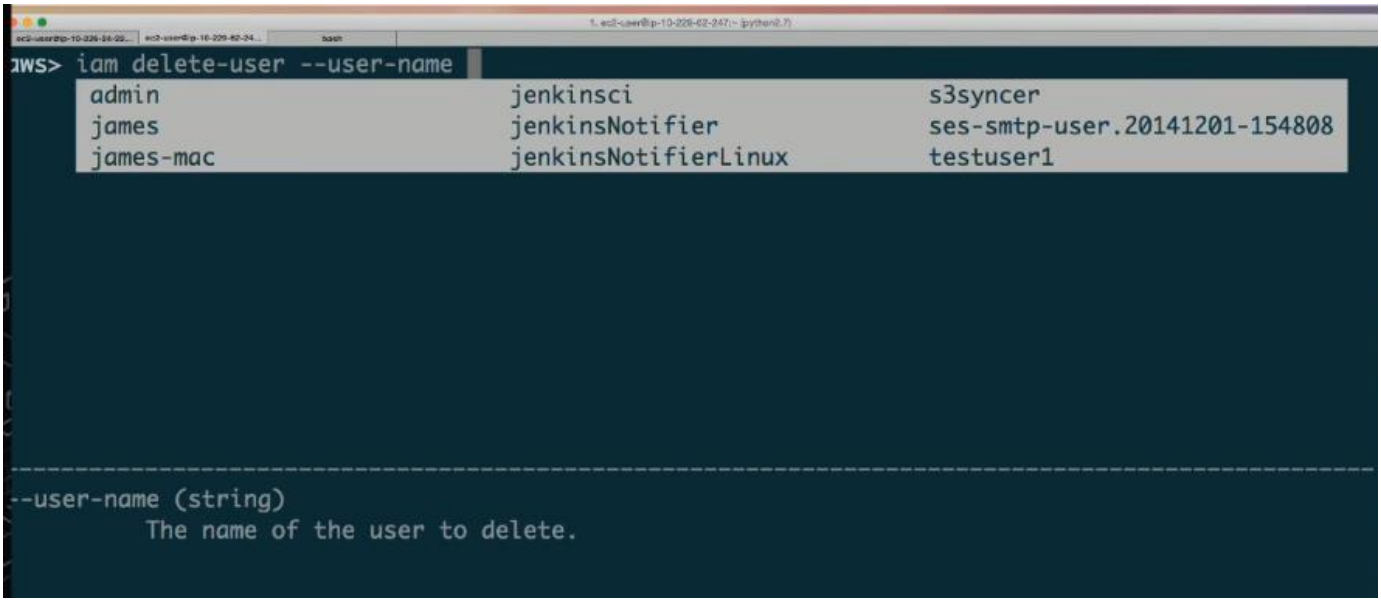

aws> ec2 terminate-instances --instance-ids
i-598234eb
Shuts down one or more instances. This operation is idempotent; if you terminate an instance more than once, each call succeeds.
Terminated instances remain visible after termination (for approximately one hour).
By default, Amazon EC2 deletes all EBS volumes that were attached when the instance launched. Volumes attached after instance launch continue running.
You can stop, start, and terminate EBS-backed instances. You can only terminate instance store-backed instances. What happens to an instance differs if you stop it or terminate it. For example, when you stop an

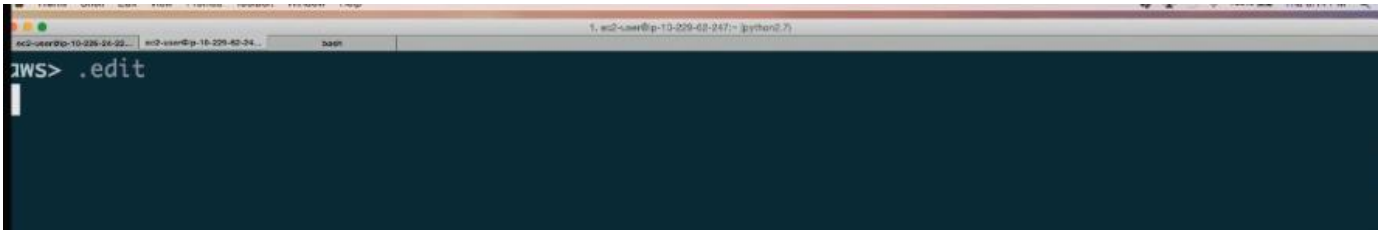

aws> ec2 terminate-instances --instance-ids i-598234eb

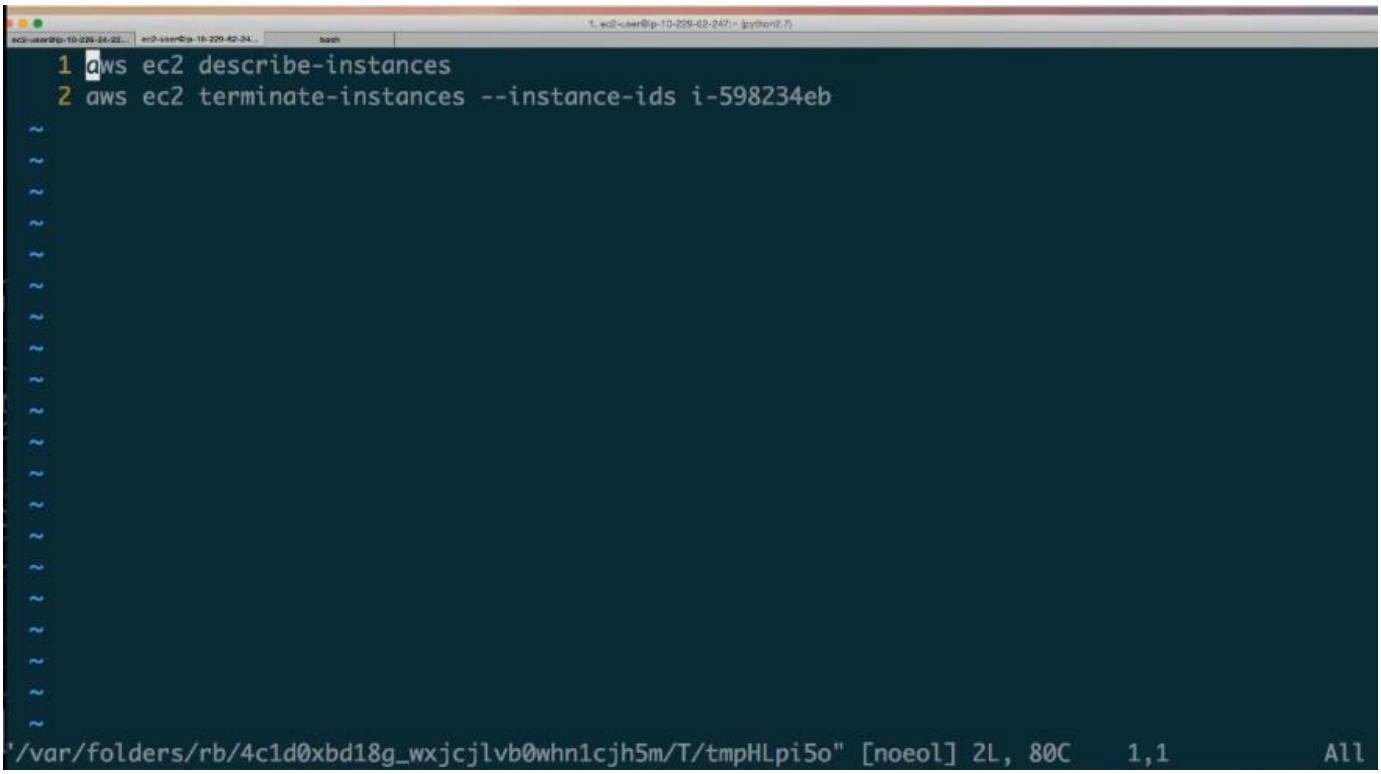

{
"TerminatingInstances": [
{
"InstanceId": "i-598234eb",
"CurrentState": {
"Code": 32,
"Name": "shutting-down"
},
"PreviousState": {
"Code": 16,
"Name": "running"
}
}
]
}
aws>


{
"TerminatingInstances": [
{
"InstanceId": "i-598234eb",
"CurrentState": {
"Code": 32,
"Name": "shutting-down"
},
"PreviousState": {
"Code": 16,
"Name": "running"
}
}
]
}
aws> iam
add-client-id-to-open-id-connect-provider
add-role-to-instance-profile
add-user-to-group
attach-group-policy
attach-role-policy
attach-user-policy
change-password
create-access-key
create-account-alias
create-group
create-instance-profile
create-login-profile
create-open-id-connect-provider
create-policy

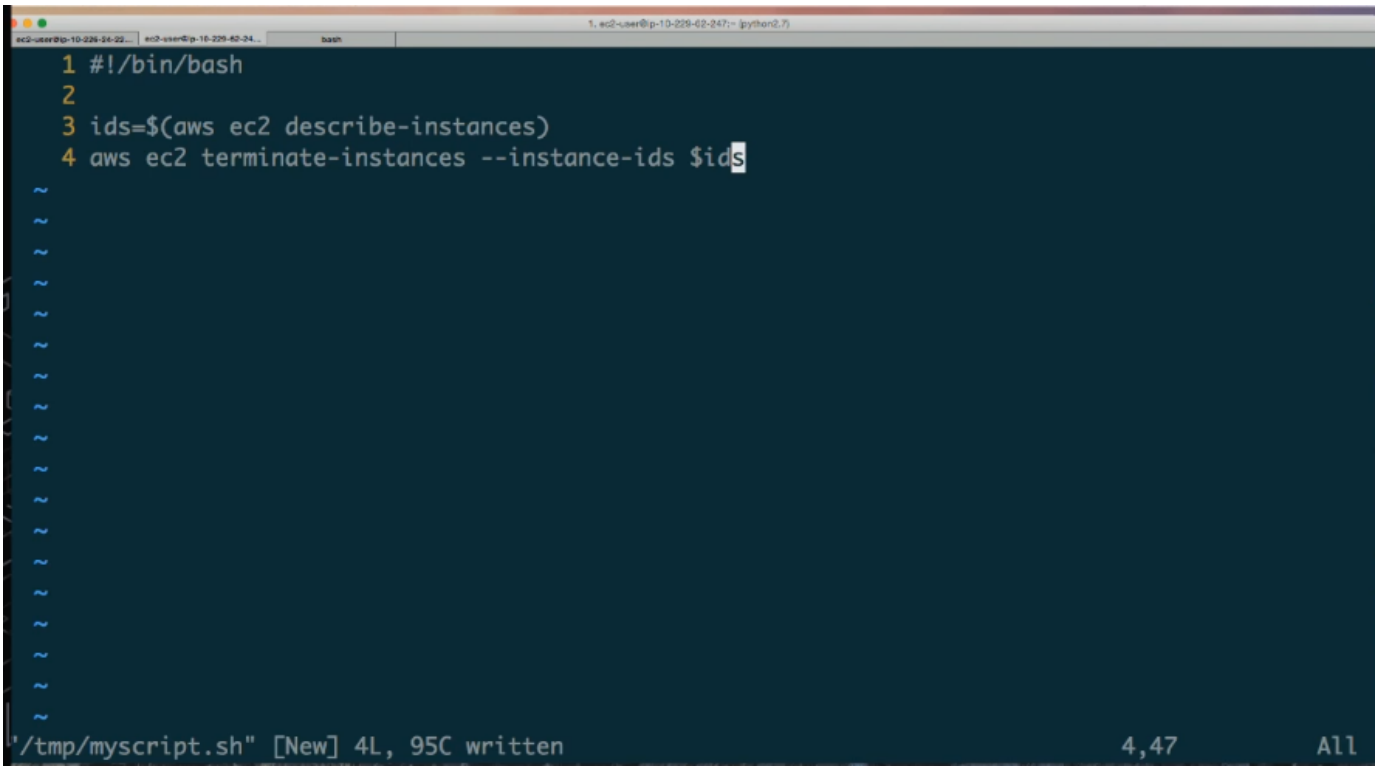

aws> iam delete-user --user
--user-name (required)
AWS Identity and Access Management (IAM) is a web service that you can use to manage users and user permissions under your AWS account. This guide provides descriptions of IAM actions that you can call programmatically. For general information about IAM, see AWS Identity and Access Management (IAM) . For the user guide for IAM, see Using IAM .
NOTE:
AWS provides SDKs that consist of libraries and sample code for various programming languages and platforms (Java, Ruby, .NET, iOS, Android, etc.). The SDKs provide a convenient way to create programmatic access to IAM and AWS. For example, the SDKs take care of tasks such as cryptographically signing requests (see below), managing errors, and retrying requests automatically. For information

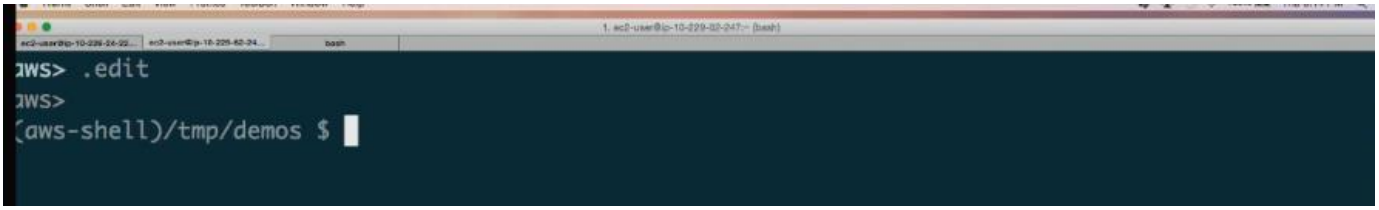

aws> iam delete-user --user-name
admin jenkinsci s3syncer
james jenkinsNotifier ses-smtp-user.20141201-154808
james-mac jenkinsNotifierLinux testuser1
--user-name (string)
The name of the user to delete.


aws> .edit



This will take all the commands that I have ran and put them into the edit file as above, I can start converting them into a shell script to use for my work.





Summary

- Common patterns
- Shell script examples
- Deployment
- Tools

All these codes will be on AWS Labs website

Remember to complete your evaluations!

AWS re:Invent

Thank you!

- User guide: http://docs.aws.amazon.com/cli/latest/userguide/
- Reference docs: http://docs.aws.amazon.com/cli/latest/reference/
- JMESPath: http://jmespath.org/
- re:Invent 2014 - https://www.youtube.com/watch?v=vP56I7qThNs
- re:Invent 2013 - https://www.youtube.com/watch?v=qiPt1NoyZm0