International Journal of Engineering & Technology, 7 (2.8) (2018) 640-651

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research Paper

Identity and access management using Boto and JSON

S. Adhirai[1], Paramjit Singh[1], R.P. Mahapatra[2]

[1]*Department of Computer Science and Engineering, PDM University, Bahadurgarh -124507 (Delhi NCR), INDIA*
[2]*Department of Computer Science and Engineering, SRM University, Modinagar, Ghaziabad -201204 (Delhi NCR), INDIA*

Abstract

Cloud computing has emerged as the important data processing tool as it tackles exponential data growth. This, in turn, makes security something of a moving target. The National Institute of Standards and Technology (NIST), has declared the Identity and Access Management (IAM) as one of the major threats to the cloud computing. The Top Threats Working Group of Cloud Security Alliance (CSA) ranks "IAM" as the second topmost threat among twelve biggest threats in cloud computing. IAM allows the cloud server for managing the web services and herby allowing the users to manage the users and corresponding permissions (user policies). Other benefits posed by the IAM are central management of users, and maintain several security qualifications. This paper focuses on Managing IAM Users, and Working with IAM Policies using JavaScript Object Notation (JSON), and Boto. The paper concludes utmost care should be given to IAM user management and IAM user policies. It is the IAM Policies which play the sole role of ensuring security. If you don't set up IAM policies properly, you will create security holes leading to security lapses.

Keywords: *Identity, Identity and Access Management, IAM Policy, Boto, JSON*

1. Introduction

Cloud computing has captured the interest of researchers as it finds wide applicability in several fields. Cloud model driven by a server allows several services such as storage, process, and retrieve large databases. As defined by the NIST, cloud has a large pool of resources shared by a closed community. Cloud provides services to various users on demand with the tools such as servers, memory, applications, and services to users [18]. Gartner suggests that by 2020, moral policing defined in Corporate "No-Cloud" will be difficult to achieve as "No-Internet" Policy as defined today. Corporate have tried to use the several model policies such as Cloud-first, and even cloud-only, for the dominant no-cloud stance policy in recent years [8]. Cloud computing has seen tremendous growth in recent years as it has lot of open resources. Along with the growing use of cloud technology, cloud security has become a major challenge for organizations. The CSA created a list for the security options for cloud, and has placed IAM to be the highest security requirement [12].
NIST declares IAM as one of the key security issues in cloud computing [22]. The Cloud Security Alliance report, released at 2016, declares twelve security threats to cloud, and they are listed as breaches in data, feeble IAM, anxious interfaces and APIs of resources, susceptible System and application, Account hijacking, Malicious insiders, Advanced persistent threats, Data loss, Insufficient due diligence, mistreatment of cloud services, Service mishandling, and other technology issues arising due to data sharing. The CSA, thus, ranks "IAM" as the second topmost threat among twelve biggest threats in cloud computing [5]. Having said so, it naturally becomes important to understand the concept of IAM.
IAM defines a discipline which provides security to the users, and groups for obtaining the right amount of resources for correct reason [1, 6, 19, 20, 24, 25]. IAM as the name suggests, aims to develop identify between the resource provider and the user. It also allows the cloud to provide correct level of security to the resources. Security provided by the IAM is more related to enterprise or business more than the technical expertise. Thus, IAM can also be defined as the framework for establishing the suitable policies for the users and the resources. IAM allows central management of resources.
This paper focuses on Managing IAM Users, and Working with IAM Policies using *JavaScript Object Notation (JSON)*, and Boto in AWS environment. The IAM Policies are expressed in JSON. So it becomes essential to understand JSON syntax. The policy submitted to the IAM model need to ensure correct JSN syntax, for making the security model to be valid.
So, accordingly, the Section 2 concentrates on JSON and its advantages, the Section 3 deals with IAM User Management, Section 4 focuses on IAM Policies. It is the IAM Policies which play the sole role of ensuring security. Simulation results achieved through the IAM policies are discussed in section 5, while section 6 concludes the paper.

2. JSON and its Advantages

JavaScript Object Notation (JSON) has the lightweight format which can be interchangable, and hence well suited for defining the IAM policies. JSON is an open standard for exchanging data on the web. JSON is language independent. JSON supports data structures such as array and objects. So it is easy to write and read data from JSON.
JSON file format can be considered as alternate to the XML language as it has the simpler readable format. Also, JSON can be enabled for transmission between the server and web application [2, 7, 9, 10, 11, 13, 14, 15, 23].
JSON uses objects and arrays. In JSON, objects and arrays can be nested.
The IAM Policies are expressed in JSON. So it becomes essential to understand JSON syntax. One of the major conditions to be satisfied before policy submission to IAM module, is checking

whether the syntax of JSON is correct or not. JSON validator discussed in section 5 checks the validation of JSON syntax.
An ***object*** refers to the collection of key and its corresponding value as the pair, and it is expressed as key:value. The key parameter in the object takes the strings format, while the values takes the JSON data types such as string, number, object, array, Boolean [13, 14]. The syntax for JSON object is given in Figure 1.

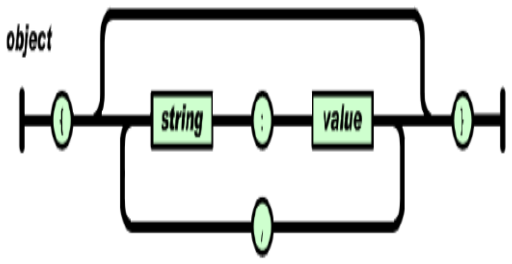


Fig. 1: Syntax of JSON Object

Here are examples of JSON Objects.
Example 1:
{"name":"Madhuri", "age":38, "car":null}

Example 2:
{
"employee": {
"name": "Ravinder",
"salary": 146000,
"married": true
}
}

Example 3:
{
"firstName": "Mohendar",
"lastName": "Uppal",
"age": 16,
"address": {
"streetAddress": "C-105, Rohini",
"city": "New Delhi",
"state": "Delhi",
"pinCode": "110063"
}
}
An ***array*** refers to the arranged sequence defined in some order and it is expressed in above figures. Figure 2 depicts the actual syntax format for the JSON array.

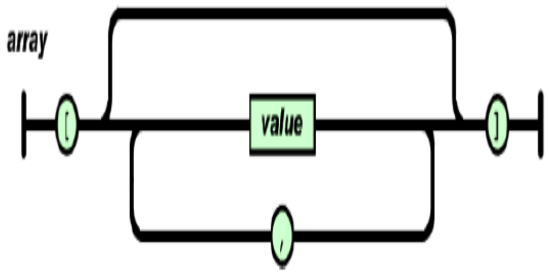


Fig. 2: Syntax of JSON Array

Here are examples of JSON arrays.

Example 1:
["Honda City", "Merc", "Audi"]

Example 2:
["Banana", "Apple", "Guava "]

Example 3:
[1, 1, 2, 3, 5, 8, 13, 21]

Example 4:
[true, true, true, false]

Example 5:
It may be noted that in a JSON array it is not necessary that the values for the array elements are all of the same type as in above examples of arrays, rather they may be of different types. So we can have mixed values as in following example:
["Banana", 1, true, null]

Example 6 (nested object and array):
{"employees":[
{"name":"Gita","email":"gita@gmail.com", "age":37},
{"name":"Mohini", "email":"mohini@hotmail.com", "age":57},
{"name":"Seema", "email":"seema@rediff.com", "age":43},
{"name":"Panda", "email":"panda@pdm.ac.in", "age":25}
]
}
JSON is the fat-free alternative to XML. The typical advantages/ benefits of JSON over XML include simplicity, extensibility, Interoperability, and Openness [2, 15, 23]. A brief description follows.

Simplicity

The first advantage posed by JSON is that it has simpler file format than the XML and the SGML. The XML has improved simplicity than SGML, but the JSON scheme has higher simplicity than other file formats. Other than the simplicity, JSON require low grammar, and have comparatively higher mapping than other file formats. Mapping of data structures in the data programming shown by JSON seems to be better than SGML.

Extensibility

As JSON does not come under the category of document markup language, it provides resistance against the extensibility. This feature makes the JSON to be compactible as it we need not define new tag for data representation.

Interoperability

Interoperability potential of the JSON and the XML language are similar.

Openness

JSON provides improved openness than the XML as it is not placed at the centre of standardization struggles.
In addition, JSON has the following features:

- JSON while compared with the XML, it can be categorized as smaller, faster and lightweight. Hence, choice between JSON and XML is easy to make during the data delivery between servers and browsers. It doesn’t take more time for execution
- For the web based applications, usage of JSON provides better results, since the JavaScript tool supports JSON language description. Another reason for the choice of JSON is that, the overhead provided during parsing of XML nodes comparatively higher than JSON.
- In the object oriented systems, mapping with JSON can be done appropriate.
- For the data exchange JSON is more suitable, but XML has been better for document exchange format.

We now concentrate on each of the following topics to access AWS Identity and IAM using the AWS for Python [3, 17, 26].

- Managing IAM Users
- Working with IAM Policies

In the example discussions that follow, Boto is the AWS for Python, for developing Amazon services like S3 and EC2. Boto is user friendly, and has object-oriented API thus has high compatibility [4].
Boto derives its name from the Portuguese name given to types of dolphins native to the Amazon River. The
Boto allows the consumers to convert the application programming interface (API) responses into Python classes. Boto3 is the latest version of the SDK, providing support for Python versions 2.6.5, 2.7 and 3.3. Boto3 includes several service-specific features to ease development. Boto supports all current AWS cloud services, including Elastic Compute Cloud, DynamoDB, AWS Config, CloudWatch and Simple Storage Service. Boto3 can be used in synchronization with Boto and hence helpful in both new and old projects.

3. Managing IAM Users

Here we describe how to create and manage users (creating user in IAM, list the various users, Changing/ updating the user name and removing the user) in IAM using Python [3, 17]. The code for managing the users is listed below:
- create_user
- get_paginator('list_users')
- update_user
- delete_user

3.1 Create a User

The example below shows how *to create a new IAM user* for AWS account using create_user method of the IAM client class. The information regarding the limitations on the number of IAM users can be visible while you create, and see Limitations on IAM Entities as mentioned in *IAM User Guide* [16].

Example: Create a new IAM user

```
import boto3
# Create IAM client
iam = boto3.client('iam')
# Create user
response = iam.create_user(
    UserName='SPECIFIED_IAM_USER'
)
print(response)
```

3.2 List IAM Users
The example for *listing the IAM users* in your **Account** using the API get_paginator('list_users') is given below:
Example: List IAM users

```
import boto3

# Create IAM client
iam = boto3.client('iam')

# List users with the pagination interface
paginator = iam.get_paginator('list_users')
for response in paginator.paginate():
    print(response)
```

3.3 Update a User's Name

Options such as AWS CLI, Tools for Windows PowerShell, or AWS API allows for renaming the users name. It provides no option for renaming the user.

The example for *updating the IAM user name using API update_user is mentioned as follows,*

Example: Update a User's Name

```
import boto3

# Create IAM client
iam = boto3.client('iam')

# Update a user name
iam.update_user(
    UserName='SPECIFIED_IAM_USER',
    NewUserName='NEW_SPECIFIED_IAM_USER'
)
```

3.4 Delete a User

The example for *deleting a specified IAM user* using delete_user is given below. The user can be deleted from the group unless the conditions such as 1) User doesn't exist in other group, and 2) Do not have any sort of access key, certificates or policies must be satisfied. The user must not belong to any groups or have any
Example: *Delete a User*

```
import boto3

# Create IAM client
iam = boto3.client('iam')

# Delete a user
iam.delete_user(
    UserName='SPECIFIED_IAM_USER'
)
```

4. Working with IAM Policies

IAM policies refer to permission provided to the user, and the policy can also be referred as document with the list of several actions. The actions may be list of users accessing the action, and the resources affecting the action. Thus, the policy can be referred as the document for stating the permission provided to the user. The actions or resources not listed in the document can be treated as invalid or denied by default. IAM policies can be created to the single user, or group of user along with their individual roles and the resources for policy [3, 17, 21].
An IAM Policy can be generally defined as the set of statements in JSON scrip, providing the information regarding the allowing or denying permissions of the object present in the AWS environment.
In the examples that follow in this section, we show how to create and get IAM policies, and along with attaching and detaching IAM policies from roles with the help of Python code. The code uses the AWS for Python using following policies,
- create_policy
- get_policy
- attach_role_policy
- detach_role_policy

4.1 Create an IAM Policy

The example below shows the procedure for creation of the IAM policy using the syntax create_policy. Using the create_policy creates the policy along with the identifier with the version v1. The version v1 was declared to be the policy's default version.
Example: Create an IAM Policy

```
import json

import boto3
```

```
# Create IAM client
iam = boto3.client('iam')

# Create a policy
my_managed_policy = {
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "logs:CreateLogGroup",
         "Resource": "RESOURCE_ARN"
      },
      {
         "Effect": "Allow",
         "Action": [
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:PutItem",
            "dynamodb:Scan",
            "dynamodb:UpdateItem"
         ],
         "Resource": "RESOURCE_ARN"
      }
   ]
}
response = iam.create_policy(
                PolicyName='myDynamoDBPolicy',
                PolicyDocument=json.dumps(my_managed_policy)
)
print(response)
```

Various components present in the permission (policy) statement is given below:

- **Effect:** Specifies the decision provided to the component. It can be Either: “Allow” or “Deny”
- **Action or NotAction:** Clearly specifies the presence of service-specific and case-sensitive commands. For example: “ec2:RunInstances”
- **Resource or NotResource:** Refers to the selected resources under the category of Amazon resource name (ARN). It can be expressed as: “arn:aws:s3:::acme_bucket/blob”
- **Condition:** Other additional constraints for policy making are specified here. One of the example is: "DateGreaterThan"

4.2 Get an IAM Policy

The procedure for obtaining the IAM policy is depicted here. IAM policy has the managed policy along with its default version. Other information such as total number of IAM users, groups, and policy attached roles are specified in IAM policy. The command list_entities_for_policy API allows the user to obtain the IAM policy and it provides the information about the list of the specific users, groups, and roles. Also, the data returned by this API is the metadata. The actual policy document can be obtained with the API get_policy_version API.
The above mentioned API returns the managed policy. Other API such as get_group_policy, get_user_policy, and get_role_policy API provides the inline policy. Example for obtaining the IAM policy with get_policy is given as,
Example: Get an IAM Policy

```
import boto3

# Create IAM client
iam = boto3.client('iam')

# Get a policy
response = iam.get_policy(
   PolicyArn='arn:aws:iam::aws:policy/AWSLambdaExecute'
)
print(response['Policy'])
```

4.3 Procedure for attaching the Managed Role Policy

The description for the attaching the managed policy with the role is expressed below: While attaching the managed policy along with the role, it acts as a part of role’s permission policy. Besides the attachment, it cannot be used as the role’s trust policy. Creation of the trust policy of the role is done the same time as the creation of role and it is done using the create_role. Also, update can be done with usingupdate_assume_role_policy.
The API allows to attach the managed policy with the role and also for embedding the inline policy put_role_policy is used.
Example for attaching the managed policy with role with the help of attach_role_policy is stated below:
Example: Attach a Managed Role Policy

```
import boto3

# Create IAM client
iam = boto3.client('iam')

# Attach a role policy
iam.attach_role_policy(

PolicyArn='arn:aws:iam::aws:policy/AmazonDynamoDBFullAcce
ss',
   RoleName='AmazonDynamoDBFullAccess'
)
```

4.4 Procedure for detaching the Managed Role Policy

For detaching the managed policy from the role can be done as follows: Initially, detach the managed policy and delete the inline policy in the role with the API delete_role_policy API.

Example: Detach a Managed Role Policy

```
import boto3

# Create IAM client
iam = boto3.client('iam')

# Detach a role policy
iam.detach_role_policy(

PolicyArn='arn:aws:iam::aws:policy/AmazonDynamoDBFullAcce
ss',
   RoleName='AmazonDynamoDBFullAccess'
)
```

5. Experimentation and Results

All the IAM Policies are expressed in JSON. Each policy must be ***validated*** for obtaining JSON with the correct syntax, and it must be ***tested*** to ensure that it meets the desired results.
All the JSON code presented in this paper, representing policies or permissions, have been validated either by using the validator entitled “The JSON Validator (JSONLint)” or the validator available as a part of AWS IAM Console. And all the JSON policies have been tested by using AWS IAM Simulator.
The JSONLint is a validator and reformatter for JSON code. In case the JSON code meets the JSON syntax (refer Figure 1, Section 2), the validator gives the result as “*Valid JSON*”, otherwise it shows the appropriate error message.
The results are discussed and shown below.

5.1 The initial JSONLint Window

Here is the opening window of JSONLint before we give any input.

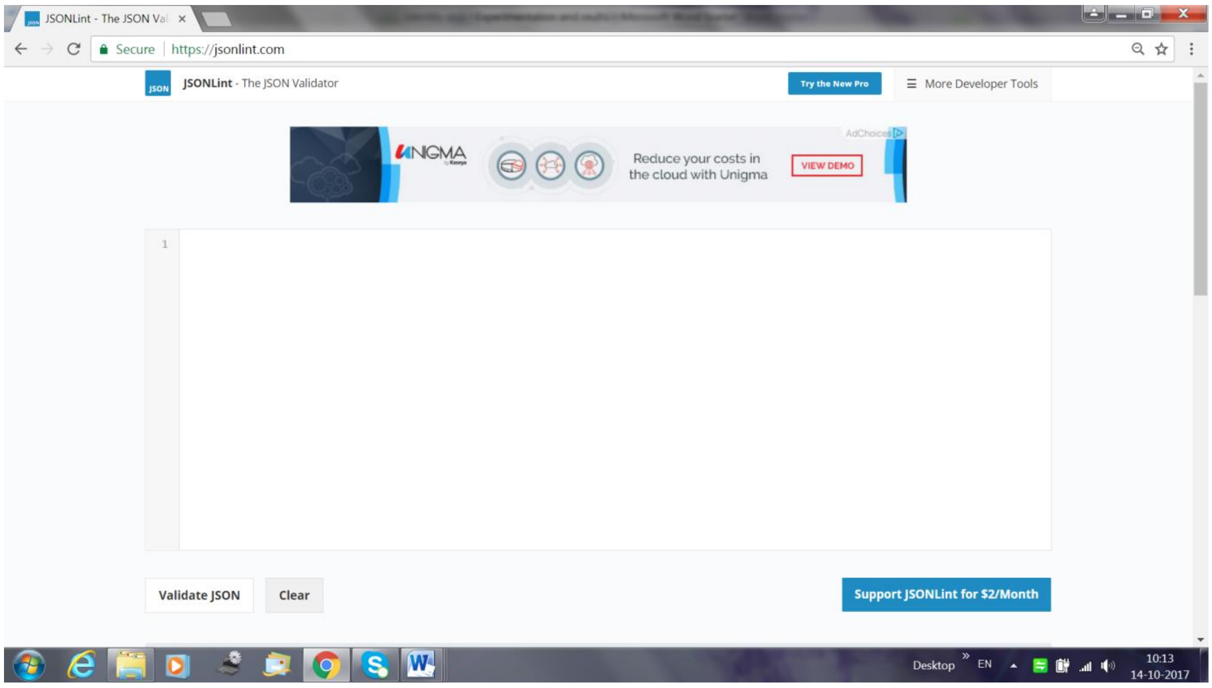
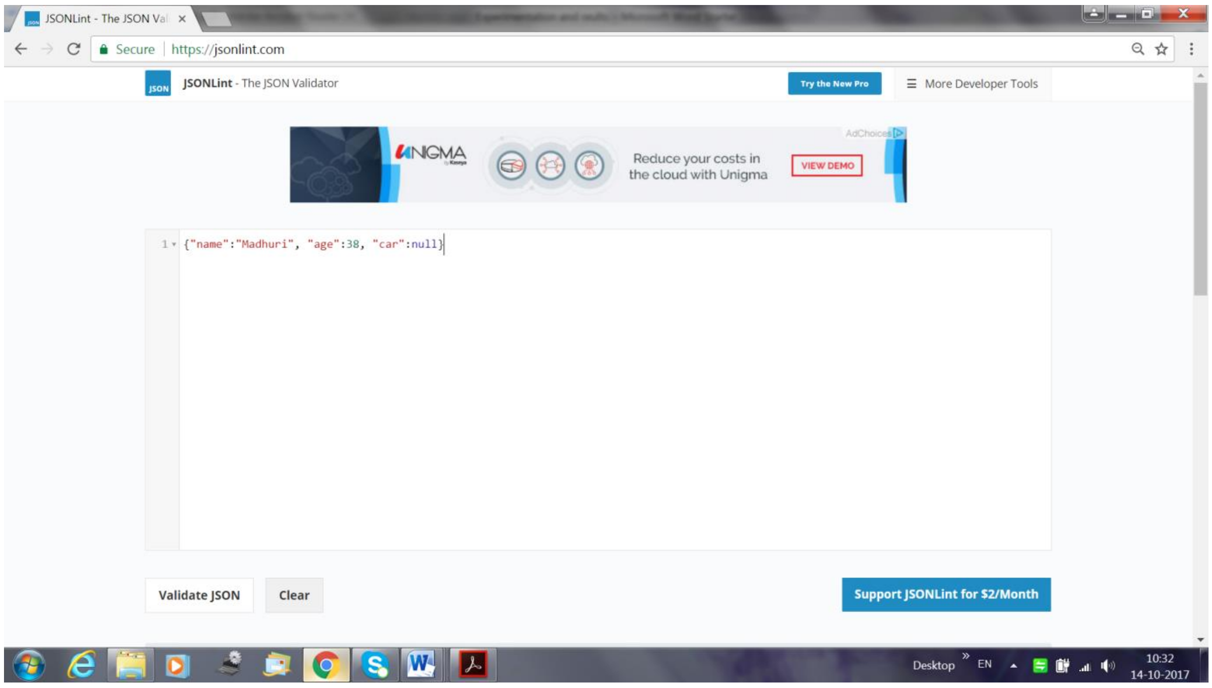


Fig. 5.1: The JSON Validator (JSONLint) before any input (the initial screenshot)

5.2 Testing of a JSON Objects

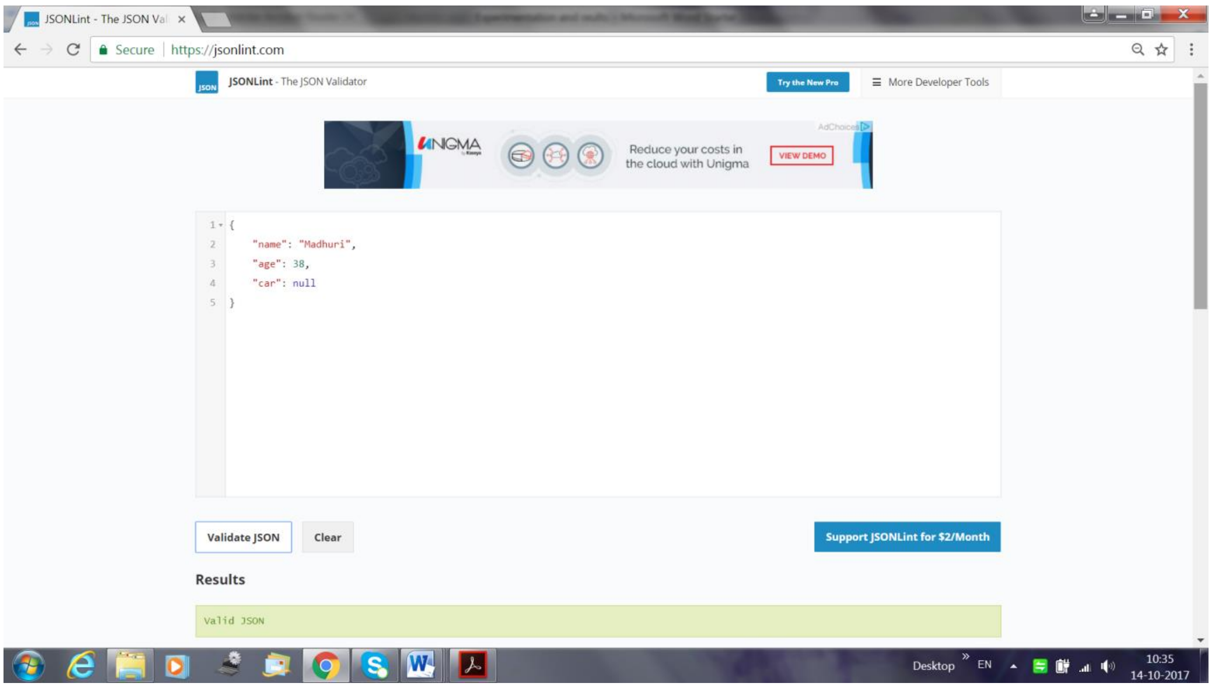
The screenshot when we give the JSON object to JSONLint looks as follows:



```
{"name":"Madhuri", "age":38, "car":null}
```

Fig. 5.2: Checking of JSON Object – the input

The results after JSON Validation look as follows:



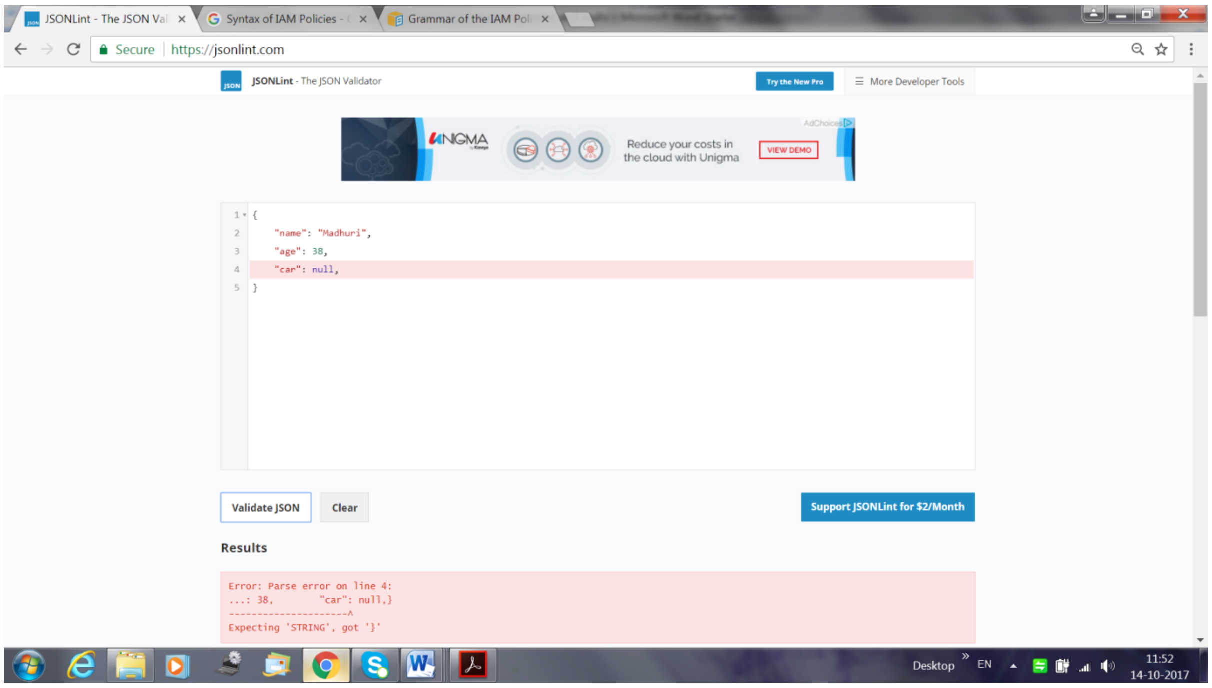
```
1 {
2     "name": "Madhuri",
3     "age": 38,
4     "car": null
5 }
```

```
Valid JSON
```

Fig. 5.3: Checking of JSON Object – the output

There are two observations: (1) The JSON Validator declares the JSON object to be "Valid", i.e. it meets with JSON syntax, and (2) it reformats the initial code as shown.

In case, there is a syntax error (extra "," at the end of line 4), the results look like:

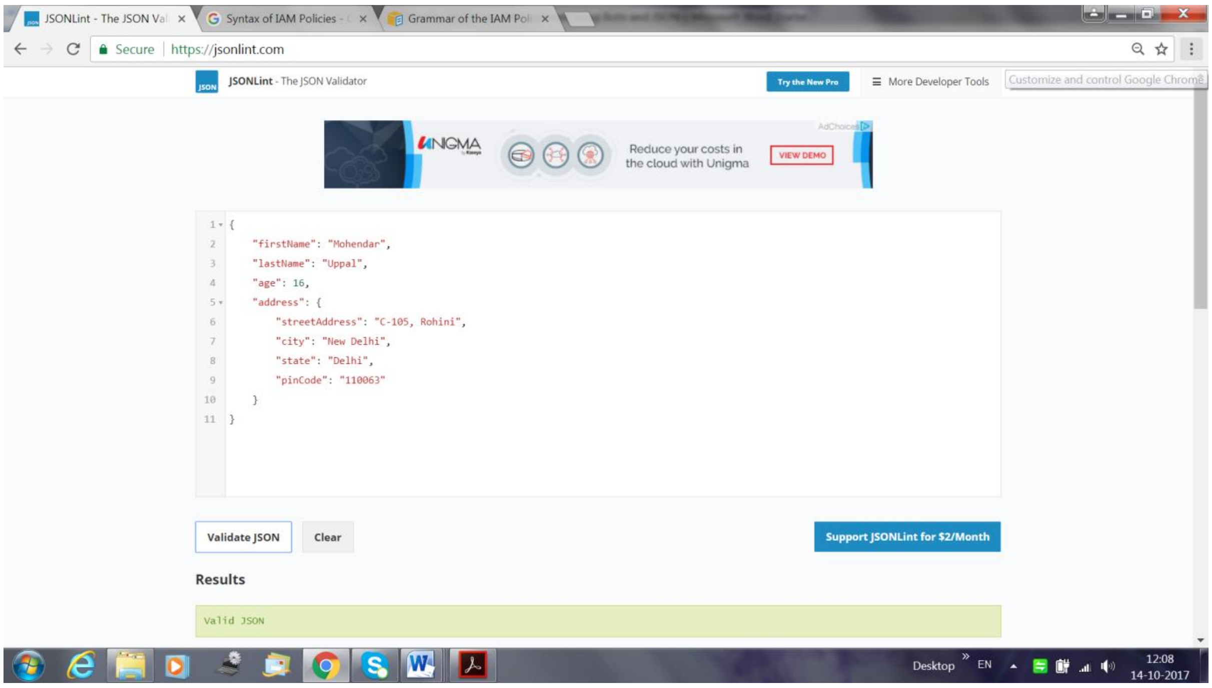


```
1 {
2     "name": "Madhuri",
3     "age": 38,
4     "car": null,
5 }
```

```
Error: Parse error on line 4:
...: 38,        "car": null,}
---------------------^
Expecting 'STRING', got '}'
```

Fig. 5.4: JSON Object with Syntax Error

Another Valid JSON Object (refer Section 2, Example 3):



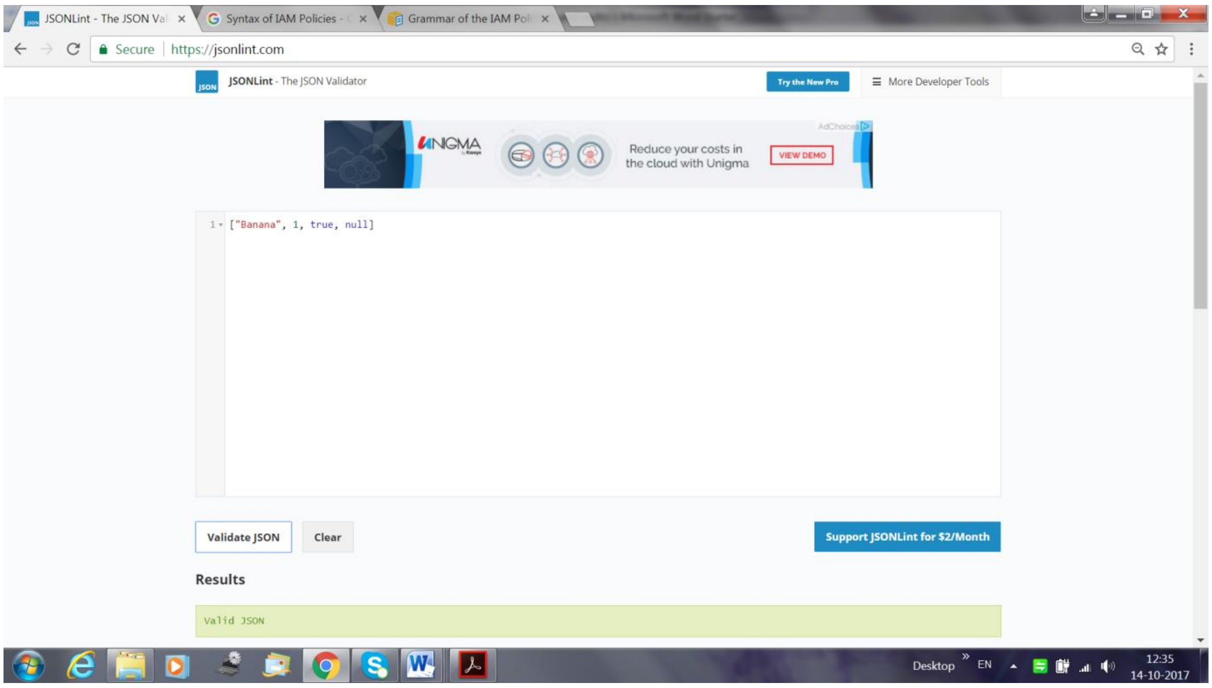
```
{
    "firstName": "Mohendar",
    "lastName": "Uppal",
    "age": 16,
    "address": {
        "streetAddress": "C-105, Rohini",
        "city": "New Delhi",
        "state": "Delhi",
        "pinCode": "110063"
    }
}
```

```
Valid JSON
```

Fig. 5.5: A Valid JSON object

5.3 Testing of a JSON Arrays

The JSON arrays must meet the Syntax of JSON Arrays defined in Figure 2, Section 2.
Here are the experimentation results.

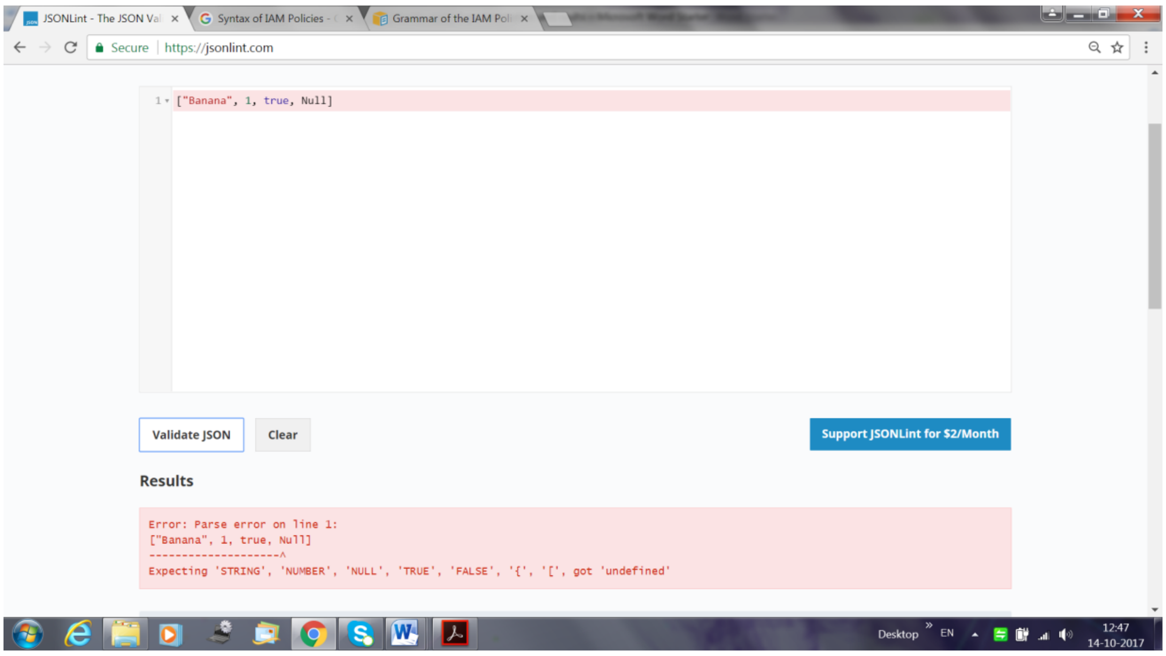


```
["Banana", 1, true, null]
```

```
Valid JSON
```

Fig. 5.6: A Valid JSON Array

It may be noted that we can have mixed data type values of array elements, as shown in this example. It may be noted the JSON data types can be either of string, number, object, array, Boolean or null. The value ‘null’ cannot be written as ‘Null’or ‘NULL’. In this case, the Validator shows the syntax error as shown below:

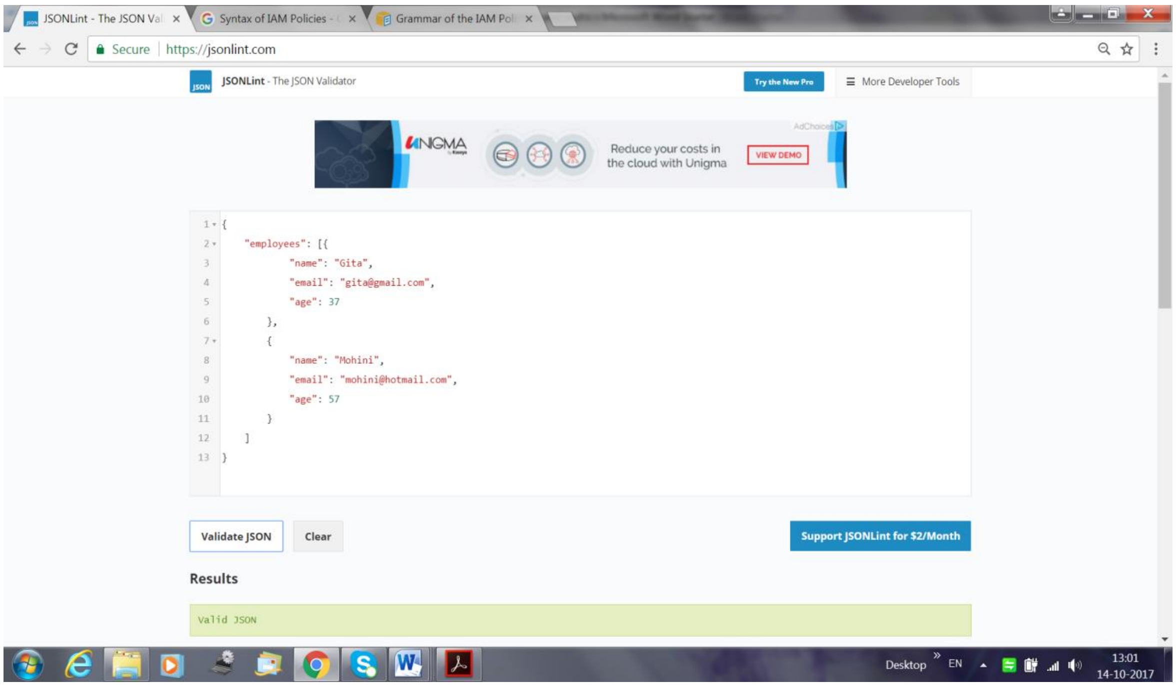


```
["Banana", 1, true, Null]
```

```
Error: Parse error on line 1:
["Banana", 1, true, Null]
--------------------^
Expecting 'STRING', 'NUMBER', 'NULL', 'TRUE', 'FALSE', '{', '[', got 'undefined'
```

Fig. 5.7: A JSON Array with syntax error.

As discussed in Section 2, we have nested JSON objects and JSON arrays. Here follows an example:



```
{
    "employees": [{
            "name": "Gita",
            "email": "gita@gmail.com",
            "age": 37
        },
        {
            "name": "Mohini",
            "email": "mohini@hotmail.com",
            "age": 57
        }
    ]
}
```

```
Valid JSON
```

Fig. 5.8: A Nested JSON Object and JSON Array.

5.4 Testing of IAM Policy

Procedure for testing the validity of the IAM policy is given below: Here, the experimentation uses the Amazon Elastic Compute Cloud (Amazon EC2) as it has high computing capacity. Also, the Amazon EC2 allows the development and deployment of applications in high speed as you don't need to investigate the hardware tolls.
Keeping these advantages in view, we write the following test policy that allows a user access to the ***all*** actions in the service named say EC2 during the month of October 2017.

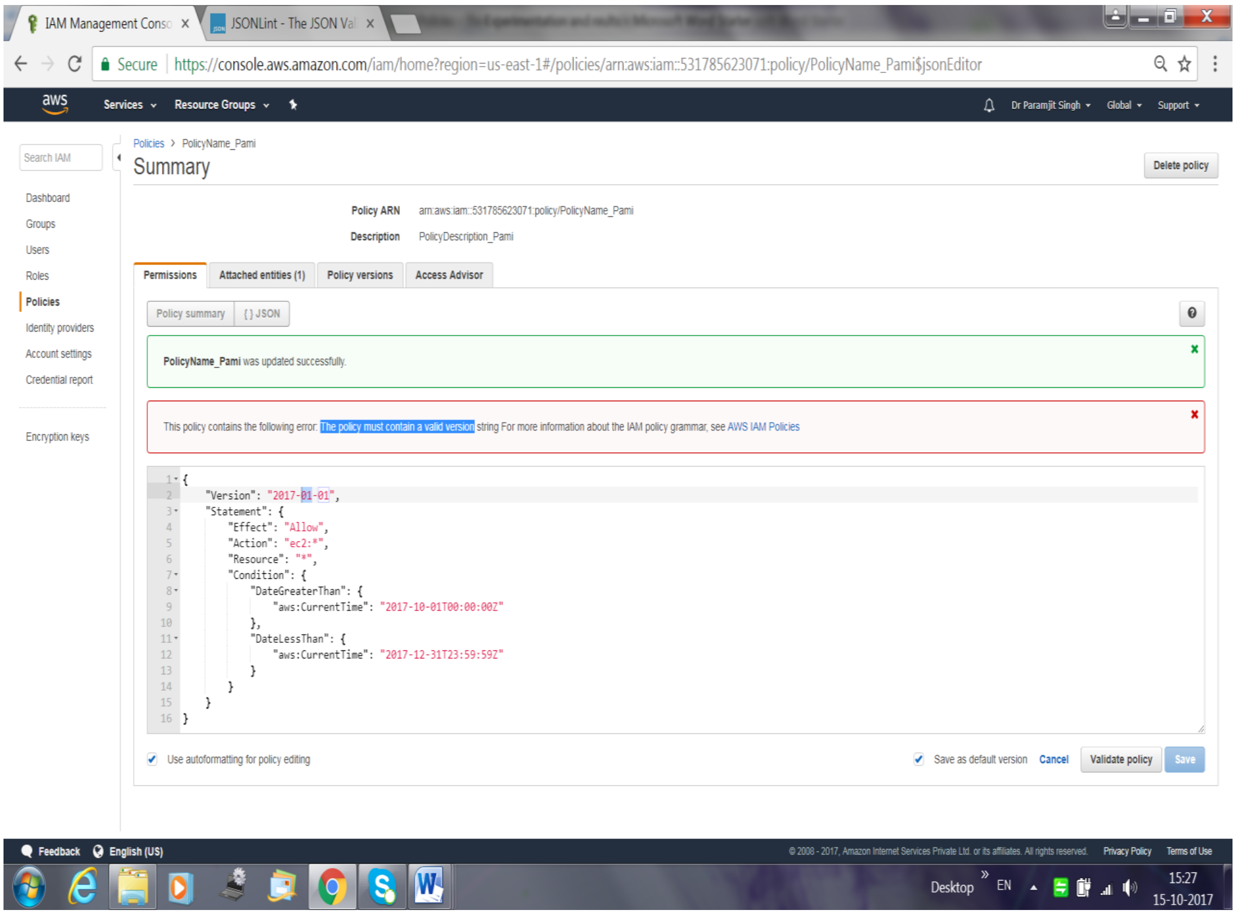
```
{
  "Version": "2017-01-01",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:*",
    "Resource": "*",
    "Condition": {
      "DateGreaterThan": {"aws:CurrentTime": "2017-10-01T00:00:00Z "},
      "DateLessThan": {"aws:CurrentTime": "2017-10-31T23:59:59Z "}
    }
  }
}
```

Fig. 5.9: A policy that allows access during a specific range of dates

If we validate this policy using JSONLint, we get the results as:

```
Valid JSON
```

meaning by that the policy complies with grammar rules of the policy. However, if we validate the same policy using AWS IAM Console, we get the result as



Summary

Policy ARN arn:aws:iam::531785623071:policy/PolicyName_Pami

Description PolicyDescription_Pami

PolicyName_Pami was updated successfully.

This policy contains the following error: The policy must contain a valid version string For more information about the IAM policy grammar, see AWS IAM Policies

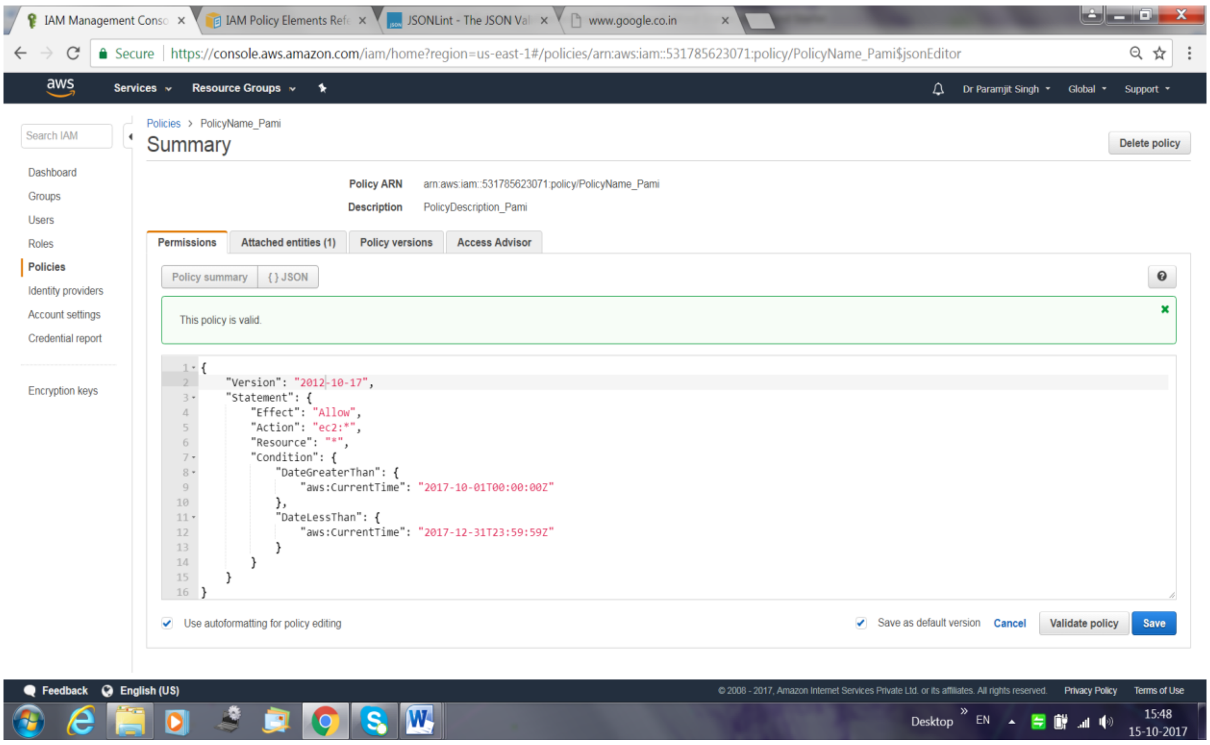
```
{
    "Version": "2017-01-01",
    "Statement": {
        "Effect": "Allow",
        "Action": "ec2:*",
        "Resource": "*",
        "Condition": {
            "DateGreaterThan": {
                "aws:CurrentTime": "2017-10-01T00:00:00Z"
            },
            "DateLessThan": {
                "aws:CurrentTime": "2017-12-31T23:59:59Z"
            }
        }
    }
}
```

[✓] Use autoformatting for policy editing

© 2008 - 2017, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use

Fig. 5.10: The Simulated results from Policy in Figure 5.9

The value for “Version” policy element must be 2012-10-17 only, and it cannot be anything else. Within the AWS IAM Console we validated this policy with Version value “2012-10-17”. The result is:


https://console.aws.amazon.com/iam/home?region=us-east-1#/policies/arn:aws:iam::531785623071:policy/PolicyName_Pami$jsonEditor

Policies > PolicyName_Pami

Summary

Policy ARN arn:aws:iam::531785623071:policy/PolicyName_Pami

Description PolicyDescription_Pami

This policy is valid.

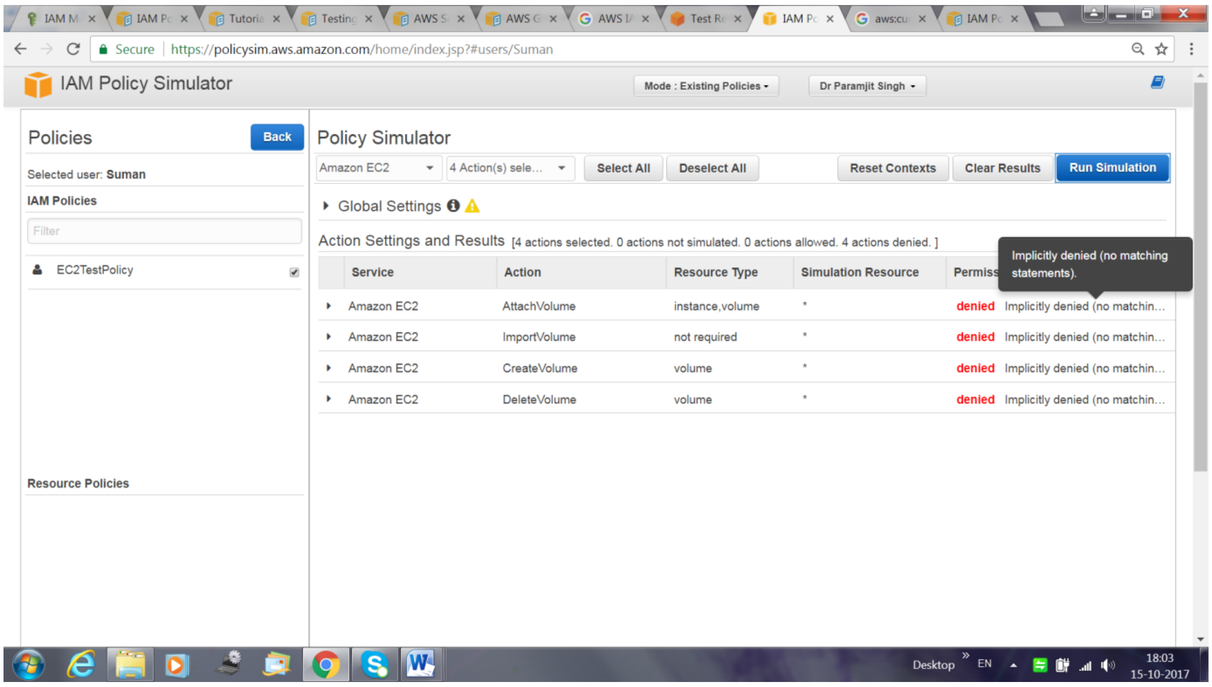
```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "ec2:*",
        "Resource": "*",
        "Condition": {
            "DateGreaterThan": {
                "aws:CurrentTime": "2017-10-01T00:00:00Z"
            },
            "DateLessThan": {
                "aws:CurrentTime": "2017-12-31T23:59:59Z"
            }
        }
    }
}
```

☑ Use autoformatting for policy editing ☑ Save as default version

Note, further, that though the AWS documentation mentions the allowed values for Version to be “2012-10-17” and “2008-10-17”, but experimentally only the current version of the policy language works.

Let us now simulate the results of the policy under consideration.

In the present context, we create a single user named Suman, and attach the above written policy named EC2TestPolicy. The simulation results look as follows:


https://policysim.aws.amazon.com/home/index.jsp?#users/Suman

IAM Policy Simulator

Policies

Selected user: Suman

IAM Policies

EC2TestPolicy ☑

Resource Policies

Policy Simulator

Amazon EC2 | 4 Action(s) sele...

Global Settings

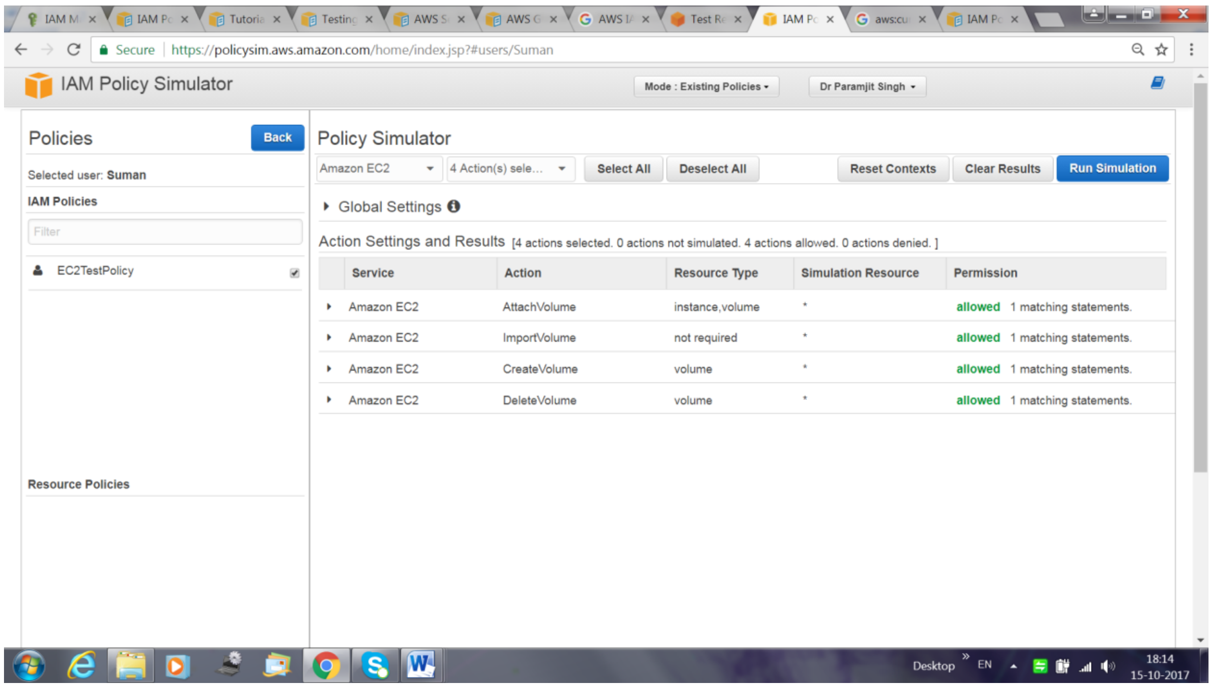
Action Settings and Results [4 actions selected. 0 actions not simulated. 0 actions allowed. 4 actions denied.]

Implicitly denied (no matching statements).

Service	Action	Resource Type	Simulation Resource	Permiss	
Amazon EC2	AttachVolume	instance,volume	*	denied	Implicitly denied (no matchin...
Amazon EC2	ImportVolume	not required	*	denied	Implicitly denied (no matchin...
Amazon EC2	CreateVolume	volume	*	denied	Implicitly denied (no matchin...
Amazon EC2	DeleteVolume	volume	*	denied	Implicitly denied (no matchin...

Fig. 5.11: Testing of EC2TestPolicy Policy

It may be noticed that all permissions are denied (contrary to our expected results). The reason being that we are yet to assign the ***Global Settings***. Once we assign the Global Settings value as “2017-10-01T00:00:00Z”, we get the correct results as follows (all the permissions become *Allowed*, as desired).



Action Settings and Results [4 actions selected. 0 actions not simulated. 4 actions allowed. 0 actions denied.]

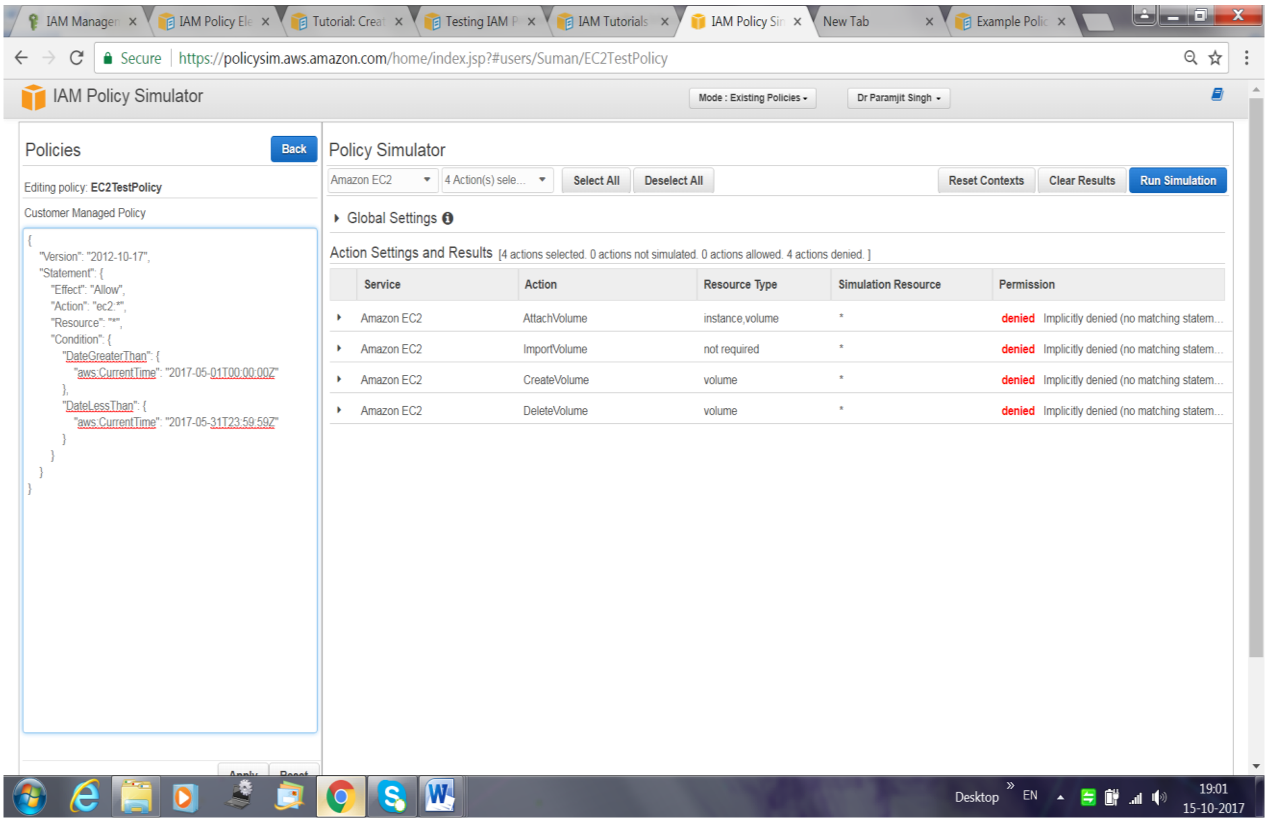
Service	Action	Resource Type	Simulation Resource	Permission
Amazon EC2	AttachVolume	instance,volume	*	allowed 1 matching statements.
Amazon EC2	ImportVolume	not required	*	allowed 1 matching statements.
Amazon EC2	CreateVolume	volume	*	allowed 1 matching statements.
Amazon EC2	DeleteVolume	volume	*	allowed 1 matching statements.

Fig. 5.12: Results of EC2TestPolicy Policy as expected

In case we change the condition in our policy as follows

```
"Condition": {
  "DateGreaterThan": {"aws:CurrentTime": "2017-05-01T00:00:00Z "},
  "DateLessThan": {"aws:CurrentTime": "2017-05-31T23:59:59Z "}
}
```

i.e. the time period is already over, we get the following results as desired.



```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:*",
    "Resource": "*",
    "Condition": {
      "DateGreaterThan": {
        "aws:CurrentTime": "2017-05-01T00:00:00Z"
      },
      "DateLessThan": {
        "aws:CurrentTime": "2017-05-31T23:59:59Z"
      }
    }
  }
}
```

Action Settings and Results [4 actions selected. 0 actions not simulated. 0 actions allowed. 4 actions denied.]

Service	Action	Resource Type	Simulation Resource	Permission
Amazon EC2	AttachVolume	instance,volume	*	denied Implicitly denied (no matching statem...
Amazon EC2	ImportVolume	not required	*	denied Implicitly denied (no matching statem...
Amazon EC2	CreateVolume	volume	*	denied Implicitly denied (no matching statem...
Amazon EC2	DeleteVolume	volume	*	denied Implicitly denied (no matching statem...

Fig. 5.13: Results of EC2TestPolicy Policy when the “condition” is not satisfied

6. Conclusion

The organizations are shifting from "No-Cloud" Policy to “Cloud-First”, and even “Cloud-Only”. The IAM is the biggest cloud security challenge. Utmost care should be given to IAM user management and IAM user policies. IAM policies are imperative when setting up permissions to ensure cloud security. Understanding how IAM policies work and how to set them up is crucial. If you don’t set up IAM policies properly, you will create security holes resulting in security compromise, or you won’t have the correct permissions for your users. It is the IAM policies which assume the sole part of guaranteeing security. The IAM policy must satisfy your application’s actual access needs.

References

[1] “Identity and Access Management (IAM)”, IT Glossary, Gartner, https://www.gartner.com/it-glossary/identity-and-access-management-iam/

[2] Advantage and Disadvantage of JSON, http://candidjava.com/advantage-and-disadvantage-of-json/.

[3] AWS Identity and Access Management Examples, http://boto3.readthedocs.io/en/latest/guide/iam-examples.html

[4] Boto 3 Documentation, https://boto3.readthedocs.io/en/latest/.

[5] CLOUD SECURITY ALLIANCE, 2016, “The Treacherous 12 - Cloud Computing Top Threats in 2016”.

[6] Gilchrist, Alasdair, An Executive Guide to Identity Access Management, Kindle Edition, RG Consulting, 2015.

[7] http://www.tutorialspoint.com/json/, JSON Tutorial.

[8] https://www.gartner.com/newsroom/id/3354117, “Gartner Says By 2020, a Corporate "No-Cloud" Policy Will Be as Rare as a "No-Internet" Policy Is Today”, STAMFORD, Conn., June 22, 2016.

[9] https://www.javatpoint.com/json-tutorial, JSON Tutorial.

[10] https://www.json.org/, Introducing JSON.

[11] https://www.w3schools.com/js/js_json_intro.asp, JSON – Introduction.

[12] Jerry Archer, Alan Boehme, Dave Cullinane, Nils Puhlmann, Paul Kurtz, Jim Reavis. CLOUD SECURITY ALLIANCE SecaaS DEFINED CATEGORIES OF SERVICE, 2011.

[13] JSON – DataTypes, https://www.tutorialspoint.com/json/json_data_types.htm.

[14] JSON Data Types, https://www.w3schools.com/js/js_json_datatypes.asp.

[15] JSON: The Fat-Free Alternative to XML, http://json.org/xml.html.

[16] Limitations on IAM Entities and Objects, http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_iam-limits.html.

[17] Managing IAM Users, http://boto3.readthedocs.io/en/latest/guide/iam-example-managing-users.html.

[18] Mell, Peter, and Grance, Timothy, The NIST Definition of Cloud Computing, Special Publication 800-145, September 2011.

[19] Orondo, Omondi, Identity & Access Management: A Systems Engineering Approach, Second Edition, IAM Imprints, Boston, MA, 2016.

[20] Osmanoglu, Ertem , Identity and Access Management: Business Performance Through Connected Intelligence, First Edition, Syngress, London, New York, 2014.

[21] Overview of IAM Policies, http://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

[22] Wayne Jansen, Timothy Grance, 2011, Guidelines on Security and Privacy in Public Cloud Computing, Special Publication 800-144.

[23] What are the advantages of JSON over XML?, https://www.quora.com/What-are-the-advantages-of-JSON-over-XML.

[24] Williamson, G., Yip D., Identity Management: A Primer, 1st Edition, MC Press, Texas, 2009.

[25] Witty R., Allan A., Enck J., Wagner R., Identity and Access Management Defined, Gartner Research, SPA-21-3430, 4 November 2003, Available online: http://www.bus.umich.edu/KresgePublic/Journals/Gartner/research/118200/118281/118281.pdf

[26] Working with IAM Policies, http://boto3.readthedocs.io/en/latest/guide/iam-example-policies.html

[27] T. Padmapriya and V. Saminadan, “Improving Throughput for Downlink Multi user MIMO-LTE Advanced Networks using SINR approximation and Hierarchical CSI feedback”, International Journal of Mobile Design Network and Innovation-Inderscience Publisher, ISSN : 1744-2850 vol. 6, no.1, pp. 14-23, May 2015.

[28] S.V.Manikanthan and K.srividhya "An Android based secure access control using ARM and cloud computing", Published in: Electronics and Communication Systems (ICECS), 2015 2nd International Conference on 26-27 Feb. 2015,Publisher: IEEE,DOI: 10.1109/ECS.2015.7124833.