Introduction

Welcome to DEVWKS-2034 - **Enabling Multi-Cloud Network Assurance with Terraform and ThousandEyes**!

In this workshop, you will learn how the ThousandEyes Terraform Provider can be used to create tests in a structured and automated way. Applications are becoming more dynamic which can result in blind spots and outages due to the resources not being monitored but by automating testing with Terraform it makes it easy to declaratively create, manage and monitor infrastructure resources.

Background Information

What is ThousandEyes?

ThousandEyes is a *Digital Experience Monitoring* (DEM) SaaS platform. DEM is the evolution of application performance monitoring, network performance monitoring, and end user experience monitoring into a comprehensive tool that analyzes the efficacy of an enterprise's applications and services.

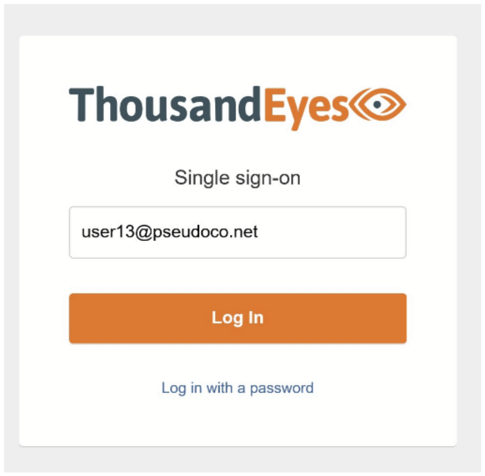
What is Terraform?

Terraform is an open-source infrastructure as code (IaC) software tool that provides a consistent CLI workflow for managing and provisioning cloud service, public and private infrastructure, and service providers. A key feature of Terraform is its ability to manage a wide variety of service providers as well as custom in-house solutions. For instance, Terraform can manage low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

Section One: Intro to ThousandEyes

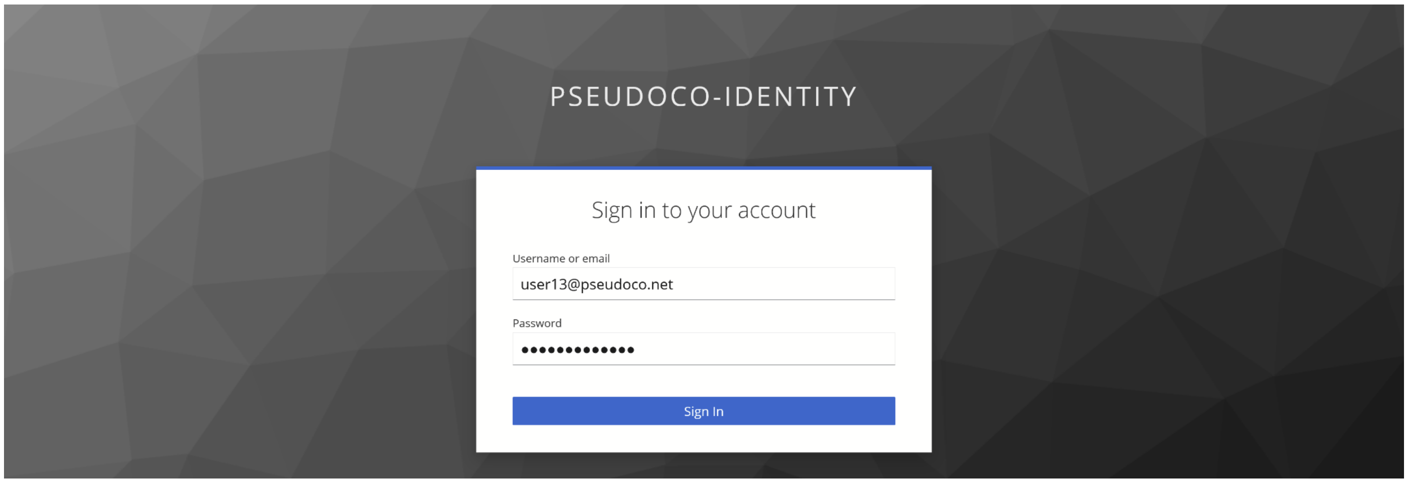
Step One: Logging In

Navigate to https://app.thousandeyes.com/login/sso



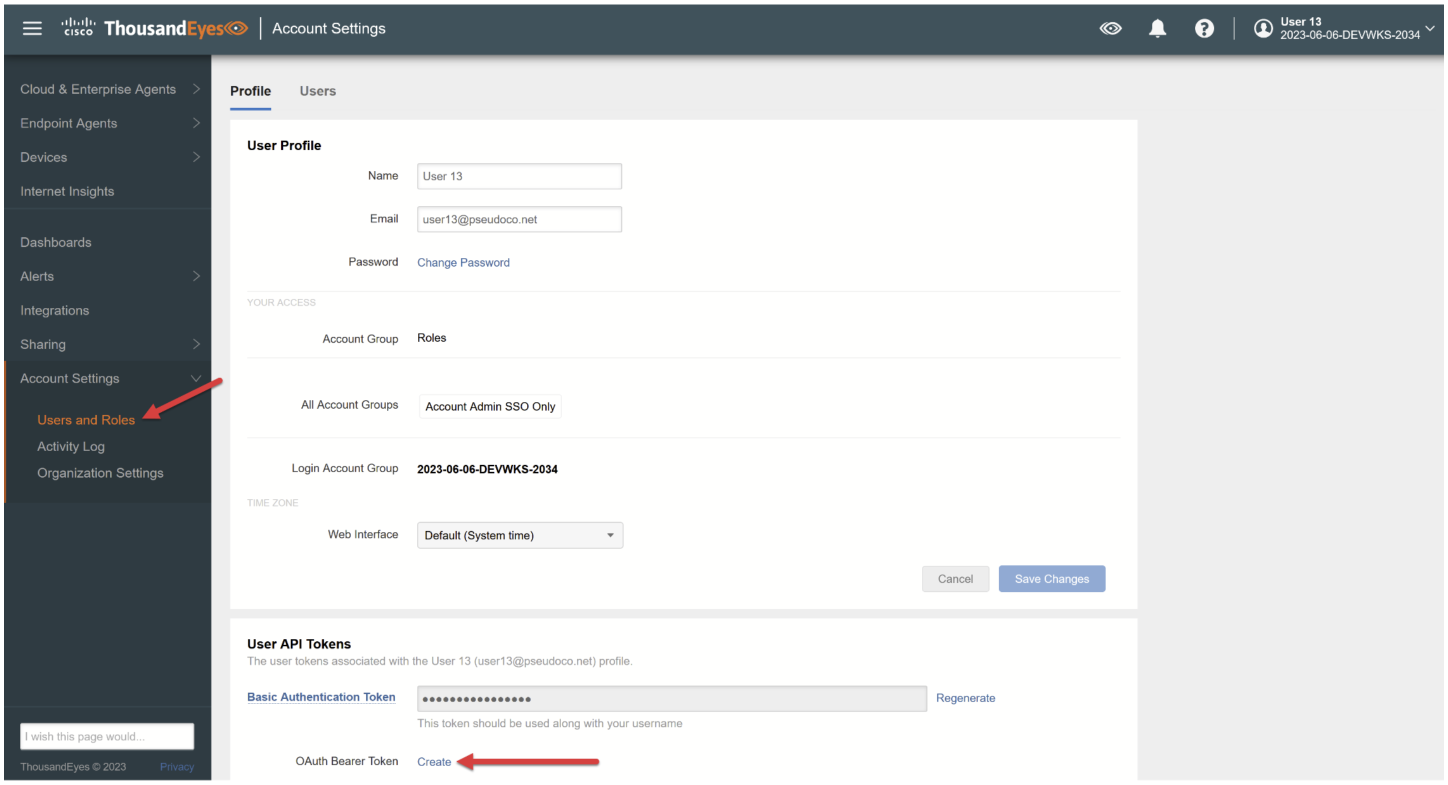
Type the e-mail address you were given, click "Log In", and when you are redirected to the IDP, login with the username and password provided, e.g.

user[NUMBER]@pseudoco.net : !MPaCT24



Step Two: Get an API key

Navigate to **Account Settings > Users and Roles > Profile > User API Tokens**



User Profile

Name	User 13
Email	user13@pseudoco.net
Password	Change Password

YOUR ACCESS

Account Group	Roles
All Account Groups	Account Admin SSO Only
Login Account Group	2023-06-06-DEVWKS-2034

TIME ZONE

Web Interface	Default (System time)

Cancel Save Changes

User API Tokens

The user tokens associated with the User 13 (user13@pseudoco.net) profile.

Basic Authentication Token	••••••••••••••••	Regenerate
	This token should be used along with your username	
OAuth Bearer Token	Create	

Scroll to the bottom and find "OAuth Bearer Token". Click the "Create" link to open a modal dialog with your new token. **Copy it and save it in a text file on your laptop, you will not be able to view the token later.**

If, when you click "Create", the page refreshes without opening the modal dialog containing the token, click the "Create" link again. If this still does not create and show a token, please ask for help from the workshop lead.

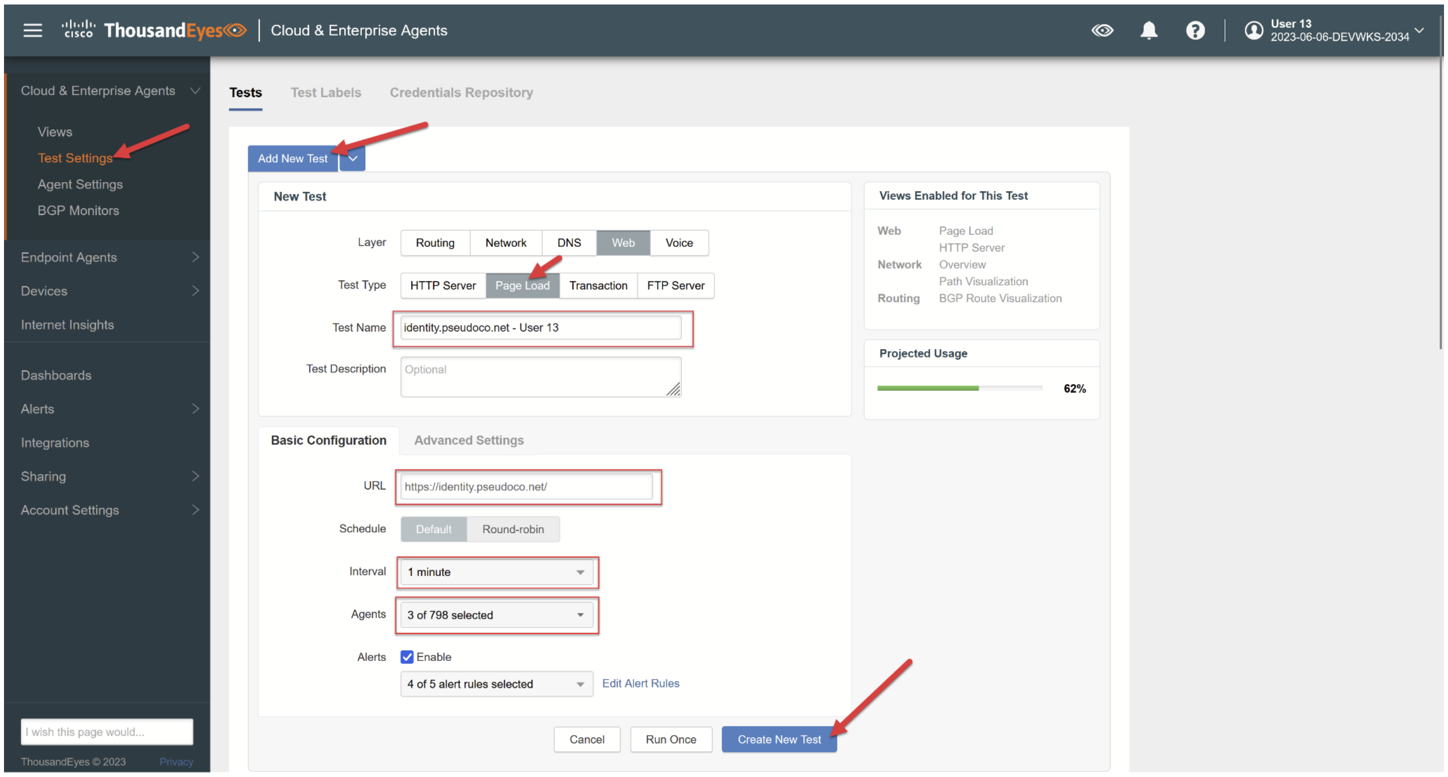
Step Three: Create a Test, Manually

Navigate to **Cloud & Enterprise Agents > Test Settings**. Click on the **Add New Test** button.

Create a **Page Load** test targeting URL https://identity.pseudoco.net:

- Name the test "identity.pseudoco.net - User **<#>**",
- Set the interval at 1 minute,
- Select up to 3 Cloud Agents for this test.
- Lastly, click "Create New Test". We'll let it run for a few test rounds and come back to look at the data.

A screenshot of this configuration is shown below.



New Test

Layer	Routing / Network / DNS / [✓ Web] / Voice
Test Type	HTTP Server / [✓ Page Load] / Transaction / FTP Server
Test Name	identity.pseudoco.net - User 13
Test Description	Optional
URL	https://identity.pseudoco.net/
Schedule	[✓ Default] / Round-robin
Interval	1 minute
Agents	3 of 798 selected
Alerts	✓ Enable; 4 of 5 alert rules selected

Views Enabled for This Test

Web	Page Load HTTP Server
Network	Overview Path Visualization
Routing	BGP Route Visualization

Projected Usage 62%

Section Two: Introduction to Terraform

Step One: Configuring the ThousandEyes Terraform Provider

On your workshop laptop, there is a “wsdn26” directory on the desktop. Inside this directory is a file named “main.tf” – open this file in your plaintext editor of choice, such as Sublime Text, TextEdit, vim, or nano. (**note: do not use LibreOffice Writer, as it is not a plaintext editor)**.

Terraform relies on plugins called "providers" to interact with remote systems. Terraform configurations must declare which providers they require, so that Terraform can install and use them. At the top of the “main.tf” file, add the following block of code - this tells Terraform to use the ThousandEyes Terraform Provider.

```
terraform {
  required_providers {
      thousandeyes = {
      source = "thousandeyes/thousandeyes"
      version = ">= 1.3.1"
      }
  }
}
```

Next, there are two configuration options we need to define:

1. The OAuth Bearer Token you created in Section One, and
2. The ThousandEyes Account Group ID

We can use the ThousandEyes API to identify our Account Group ID. In your terminal window, run curl against the ThousandEyes API with your OAuth Bearer token.

```
curl https://api.thousandeyes.com/v6/account-groups.json \
  -H "Authorization: Bearer Your-OAuth-Token-Here"
```

The response output should look like this; note the “aid” value.

```
{
      "accountGroups": [
      {
            "accountGroupName": "IMPACTFY24-WSDN26",
            "aid": 1712921,
            "current": 1,
            "default": 1,
            "organizationName": "IMPACTFY24-WSDN26"
      }
      ]
}
```

Back in your “main.tf” file, add a new block beneath the block you’ve already added:

```
provider "thousandeyes" {
  token = "insert-your-OAuth-bearer-token-here"
  account_group_id = "1712921"
}
```

Step Two: Defining a ThousandEyes test in Terraform code

Now that we've included and configured the ThousandEyes Terraform Provider, let's define a new ThousandEyes test using Terraform code. Add the following block to your "main.tf" file.

```
resource "thousandeyes_http_server" "api_thousandeyes_http_test" {
  test_name      = "ThousandEyes API Test - User <#>"
  interval       = 60
  alerts_enabled = false

  url = "https://api.thousandeyes.com/status.json"

  agents {
     agent_id = 61
  }
}
```

Make sure to save your "main.tf" file at this point.

Step Three: Initializing Terraform

In your terminal, change your working directory to the DEVWKS-2034 directory:

```
cd ~/Desktop/wsdn26/terraform
```

Then, run:

```
terraform init
```

You should see the following output indicating that the initialization was successful:

```
Initializing the backend...

Initializing provider plugins...
- Finding thousandeyes/thousandeyes versions matching ">= 1.3.1"...
- Installing thousandeyes/thousandeyes v2.0.1...
- Installed thousandeyes/thousandeyes v2.0.1 (self-signed, key ID
02BB91CE566497C7)

Partner and community providers are signed by their developers.
```

```
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Step Four: Planning

Before you run Terraform to apply the changes to the remote system (i.e, ThousandEyes SaaS platform), you should first use the "terraform plan" command to view what changes will be made and validate that everything looks good with the configuration:

```
terraform plan
```

The output of this command will show the actions that Terraform will take based on your Terraform configuration file:

```
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # thousandeyes_http_server.api_thousandeyes_http_test will be created
  + resource "thousandeyes_http_server" "api_thousandeyes_http_test" {
      + alerts_enabled         = false
      + api_links              = (known after apply)
      + auth_type              = "NONE"
      + bandwidth_measurements = false
      + content_regex          = ".*"
      + created_by             = (known after apply)
      + created_date           = (known after apply)
      + enabled                = true
      + follow_redirects       = true
      + groups                 = (known after apply)
      + http_target_time       = 1000
      + http_time_limit        = 5
      + http_version           = 2
      + id                    = (known after apply)
      + interval              = 60
      + live_share            = (known after apply)
      + modified_by           = (known after apply)
      + modified_date         = (known after apply)
      + network_measurements    = true
      + path_trace_mode       = "classic"
      + probe_mode            = "AUTO"
      + protocol              = "TCP"
      + saved_event           = (known after apply)
      + ssl_version           = (known after apply)
      + ssl_version_id        = 0
      + test_id               = (known after apply)
      + test_name             = "ThousandEyes API Test - User 13"
      + type                  = (known after apply)
      + url                   = "https://api.thousandeyes.com/status.json"
      + verify_certificate  = true

      + agents {
           + agent_id       = 61
           + agent_type    = (known after apply)
           + ip_addresses = (known after apply)
      }
      }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Review the output of the `terraform plan` command, then proceed to the next step.

Step Four: Applying

Now that we understand what Terraform will do, let's apply the plan:

```
terraform apply
```

Again, you will see a preview of the actions Terraform will apply, and Terraform will prompt you to confirm to perform these actions. **Be sure to type "yes" at the prompt, and hit your enter key**.

```
...
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

thousandeyes_http_server.api_thousandeyes_http_test: Creating...
thousandeyes_http_server.api_thousandeyes_http_test: Creation complete after 2s
[id=3767890]
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

You've now successfully deployed a ThousandEyes test via Terraform!

Navigate to **Cloud & Enterprise Agents > Test Settings** to see your new test.

Section Three:

Importing Existing Resources

So far, we've used Terraform to create new resources in our cloud environments. What if we want to leverage Terraform to manage an existing resource? Recall in the first section of this exercise that we created a ThousandEyes test manually. Let's explore the process to import that test into our Terraform code.

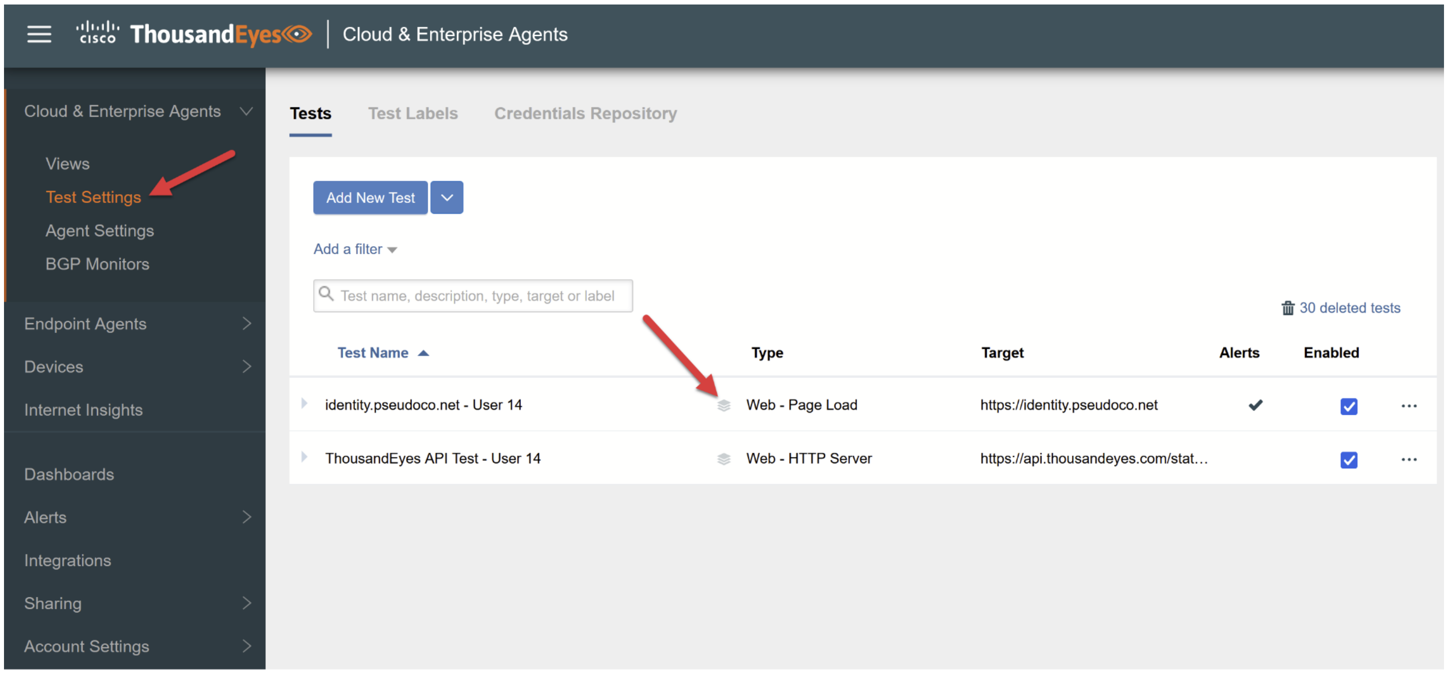
Step One: Importing Resource State

First, add a new code block to "main.tf" represent the existing resource:

```
resource "thousandeyes_page_load" "identity_pseudoco_net_test" {
}
```

Before we import the existing resource, we will need the ID of the resource we are importing. You can easily find your test ID from the ThousandEyes platform, by opening the Views for the test and looking for the `testId` URL parameter. To open the View for your "identity.pseudoco.net" test that you created manually in Section

One, navigate to **Cloud & Enterprise Agents > Test Settings**, find your test in the list, and click on the button:



Test Name	Type	Target	Alerts	Enabled
identity.pseudoco.net - User 14	Web - Page Load	https://identity.pseudoco.net	✓	✓
ThousandEyes API Test - User 14	Web - HTTP Server	https://api.thousandeyes.com/stat…		✓

Clicking on the button will open the test view, and you can look at the browser's URL bar to find the test ID, **for example:**

https://app.thousandeyes.com/view/cloud-and-enterprise-agents/?testId=**3770885**

Then, run the following command to import the existing state into Terraform:

```
terraform import thousandeyes_page_load.identity_pseudoco_net_test <TEST_ID>
```

You should see output like the following:

```
thousandeyes_http_server.identity_pseudoco_net_test: Importing from ID
"3767796"...
thousandeyes_http_server.identity_pseudoco_net_test: Import prepared!
  Prepared thousandeyes_http_server for import
thousandeyes_http_server.identity_pseudoco_net_test: Refreshing state...
[id=3767796]

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

Step Two: Importing Configuration

We have now imported the *state* for this resource, but next we need to import the *configuration*, so that we can manage this resource via Terraform going forward.

Run the `terraform show` command to display the current state, and look for the "# thousandeyes_http_server.identity_pseudoco_net_test" section in the output. It should look something like this:

```
resource "thousandeyes_http_server" "identity_pseudoco_net_test" {
      alerts_enabled         = true
      api_links              = [
      {
            href = "https://api.thousandeyes.com/v6/tests/3767796"
            rel  = "self"
      },
      {
            href = "https://api.thousandeyes.com/v6/web/http-server/3767796"
            rel  = "data"
      },
      {
            href = "https://api.thousandeyes.com/v6/web/page-load/3767796"
            rel  = "data"
      },
      {
            href = "https://api.thousandeyes.com/v6/net/metrics/3767796"
            rel  = "data"
      },
      {
            href = "https://api.thousandeyes.com/v6/net/path-vis/3767796"
            rel  = "data"
      },
      {
            href = "https://api.thousandeyes.com/v6/net/bgp-metrics/3767796"
            rel  = "data"
      },
      ]
      auth_type              = "NONE"
      bandwidth_measurements = false
      bgp_measurements       = true
      created_by             = "User 13 (user13@pseudoco.net)"
      created_date           = "2023-06-06 12:03:26"
      enabled                = true
      follow_redirects       = true
      http_target_time       = 1000
      http_time_limit        = 5
      http_version           = 2
      id                     = "3767796"
      interval               = 60
      live_share             = false
      modified_by            = "User 13 (user13@pseudoco.net)"
      modified_date          = "2023-06-06 12:37:00"
      mtu_measurements       = false
      network_measurements    = true
      num_path_traces        = 3
      path_trace_mode        = "classic"
      probe_mode             = "AUTO"
      protocol               = "TCP"
      saved_event            = false
      ssl_version            = "Auto"
      ssl_version_id         = 0
      test_id                = 3767796
      test_name              = "identity.pseudoco.net - User 13"
      type                   = "page-load"
      url                    = "https://identity.pseudoco.net"
      use_ntlm               = false
      verify_certificate     = true

      agents {
      agent_id               = 317746
      enabled                = false
      ip_addresses           = []
      keep_browser_cache     = false
      utilization            = 0
      verify_ssl_certificate = false
      }
      agents {
      agent_id               = 552901
      enabled                = false
      ip_addresses           = []
      keep_browser_cache     = false
      utilization            = 0
      verify_ssl_certificate = false
      }
      agents {
      agent_id              = 68
      enabled               = false
      ip_addresses          = []
      keep_browser_cache    = false
      utilization           = 0
      verify_ssl_certificate = false
      }
}
```

Copy this block and overwrite the empty "identity_pseudoco_net_test" block in your “main.tf” file, and run `terraform plan`. You will see multiple “Value for unconfigurable attribute” errors. This is because the `terraform show` command shows the complete state, including read-only attributes. Our Terraform configuration must not contain any read-only attributes. Remove the following fields from the “identity_pseudoco_net_test” block:

- api_links
- created_by
- created_date
- id
- live_share
- modified_by
- modified_date
- saved_event
- ssl_version
- test_id
- type

Next, re-run `terraform plan` and validate there are no errors. Now you are able to use Terraform to manage a test that you had initially created manually!

Section Four: Variables

Recall that when we initially set up “main.tf” with the ThousandEyes provider, we put our OAuth Bearer Token directly in the code. While this is fine for learning and experimenting, it is an anti-pattern to include secrets directly in your code files.

Step One: Declare the Variable

Let’s replace the token configuration to use a *variable* that we will define outside of the code. Open “main.tf”, add a new block to declare the variable, and update the ThousandEyes provider section like so:

```
variable "te_oauth_token" {
  type = string
}

provider "thousandeyes" {
  token = var.te_oauth_token
  account_group_id = "1712921"
}
```

Be sure to save the file.

Step Two: Define the Value of the Variable

```
export TF_VAR_te_oauth_token="YOUR-OAUTH-TOKEN"
```

Now, run `terraform plan` to validate that everything is working correctly.

Section Five: Deploy and Monitor a Web Application

Let’s deploy a web application to AWS and, at the same time, define a new ThousandEyes test to run against the application.

Step One: Add a New AWS EC2 Resource

First, open “aws.tf” and add a new EC2 resource:

```
resource "aws_instance" "web_application_user<#>" {
  ami = "ami-024e6efaf93d85776"
  instance_type = "t2.small"
  key_name = "impactfy24-wsdn26"

  user_data = <<-EOL
  #!/bin/bash -x
  sudo apt update && sudo apt -y install nginx
  EOL

  tags = {
    Name = "web_application_user<#>"
  }
}
```

Step Two: Add a New ThousandEyes HTTP Server Test Resource

Next, open “main.tf” and add a new ThousandEyes HTTP Server test resource:

```
resource "thousandeyes_http_server" "ec2_http_test" {
  test_name = "EC2 Web Server Test User<#>"
  interval = 60
  alerts_enabled = false
  url = "http://${aws_instance.web_application_user<#>.public_dns}"

  agents {
      agent_id = 61
  }
}
```

Note the value that we use for the url:

```
"http://${aws_instance.web_application_user<#>.public_dns}"
```

This tells Terraform that the Target of our ThousandEyes test will be the new EC2 instance we are deploying.

Step Three: Plan and Apply

Run `terraform plan` to validate everything is correct, then run `terraform apply`.

Navigate to **Cloud & Enterprise Agents > Views** to view your new test and ensure it is successfully reaching your new web application.

Section Six: Deploy Additional Tests for the Application Backend

We've now deployed a new web application, and a test that runs from ThousandEyes Cloud Agents towards that new web app. This serves as a representation for how users on the public Internet reach our app.

However, most applications today are dependent upon third party services and external APIs. Let's presume that our web application is dependent on the following three APIs:

- api.github.com
- api.slack.com
- api.twilio.com

Because these API requests are made from *within* our AWS VPC, for the tests we set up against these targets, we want to use the Enterprise Agent we deployed earlier. This Enterprise Agent will represent our web application making third-party API calls to external dependencies across the Internet.

Step One: Declare your Enterprise Agent as a Data Source

First, add the Enterprise Agent as a new data object in "main.tf", immediately after the "provider" block:

```
data "thousandeyes_agent" "arg_useast2_enterprise_agent" {
  agent_name = "te-agent-user<#>"
}
```

This will allow us to reference the Enterprise Agent by name, rather than its numeric ID.

Step Two: Add New ThousandEyes HTTP Server Test Resources

Next, add three new HTTP Server test resources at the bottom of "main.tf":

```
resource "thousandeyes_http_server" "api_github_http_test" {
  test_name      = "Github API Test - User<#>"
  interval       = 60
  alerts_enabled = false

  url = "https://api.github.com/"

  agents {
      agent_id = data.thousandeyes_agent.arg_useast2_enterprise_agent.agent_id
  }
}

resource "thousandeyes_http_server" "api_slack_http_test" {
  test_name      = "Slack API Test - User<#>"
  interval       = 60
  alerts_enabled = false

  url = "https://api.slack.com"

  agents {
      agent_id = data.thousandeyes_agent.arg_useast2_enterprise_agent.agent_id

  }
}

resource "thousandeyes_http_server" "api_twilio_http_test" {
  test_name      = "Twilio API Test - User<#>"
  interval       = 60
  alerts_enabled = false

  url = "https://api.thousandeyes.com/status.json"

  agents {
      agent_id = data.thousandeyes_agent.arg_useast2_enterprise_agent.agent_id
  }
}
```

Step Three: Plan and Apply

Run `terraform plan` to validate there are no issues in your Terraform code files, then run `terraform apply` to apply the changes and create the three new tests in the ThousandEyes platform.

Conclusion

In this workshop, we used Terraform to define our Infrastructure as Code. Specifically, we used Terraform with the ThousandEyes provider to define our ThousandEyes tests in code. We created “Outside-In” tests, from Cloud Agents to our Website.

For more information, read the blog post announcing the ThousandEyes Terraform Provider. Complete documentation is available on the Terraform registry. To learn more about how we use Terraform at ThousandEyes, check out Ricard Bejarano’s “Scaling Terraform at ThousandEyes” presentation from SREcon23 Americas.

You can also learn more about public cloud performance in the ThousandEyes Cloud Performance Report. The Cloud Performance Report analyzes the performance and connectivity architectures of the top three public cloud providers: Amazon Web Services (AWS), Microsoft Azure, and Google Cloud.

Thank you for joining today’s session!