Using Terraform to Orchestrate Distributed Firewalling of AWS Workloads with Cisco Tetration

Contents

The need

Developers are reacting to the need for business agility with ever more frequent software deployments, but are often hampered by delays in IT policy and security reviews. IT and security operations need to maintain data and systems security in both on-premises and public cloud data centers. Fundamentally there exists a conflict of goals that may drive developers to compromise security in the name of agility, thereby creating operational risk.

Proposed solution

We are proposing the use of Cisco® Tetration coupled with HashiCorp Terraform and AWS to meet the needs of both the development and operations teams. Cisco Tetration will allow security and IT operations to set policy, and developers can implement the policy by consuming the proper components while also creating their own zero-trust security settings that allow their application to operate. The key to this solution is the ability for different teams to consume Cisco Tetration in the manner that easily integrates into their normal workflow. This reduces adoption time and smooths integration. In this solution we allow the developers to use their existing HashiCorp Terraform workflows to deploy their workloads to AWS and create policies that allow required communication. The security and IT teams are free to set global policy via Terraform or the Cisco Tetration administrative interface.

Key concepts

The first key concept that makes the proposed solution work is the manner in which Cisco Tetration applies a centrally computed firewall ruleset to each application's workload. This allows microsegmented (zero-trust) firewall rules specific to each Virtual Machine (VM) or Kubernetes node to be automatically loaded and updated from a centralized system, Cisco Tetration. Workloads receive the appropriate firewall ruleset no matter whether they are deployed to public or private clouds using application-centric metadata provided by external orchestration systems like AWS EC2 or a Kubernetes cluster. If a workload moves, it automatically receives the correctly updated ruleset based on the new environment. Using this capability, we can easily protect development, test, and production environments from communicating with each other through annotations for each workload progressing through a CI/CD pipeline utilizing HashiCorp Terraform.

The second key concept that makes the proposed solution work is how Cisco Tetration applies an absolute security policy ruleset from the top of an organizational structure down a scope tree. This allows IT operations and security policy to enforce global rules for the organization. It also allows corporate governance of policy to be applied as required before the lower branches of the scope tree where specific application communications are allowed. Once the absolute policies are applied, the default policies per scope are applied from the most granular component of the scope back up to the top. By doing this, application owners can open specific required communication for permitted communication before rules higher in the scope stop all remaining network traffic.

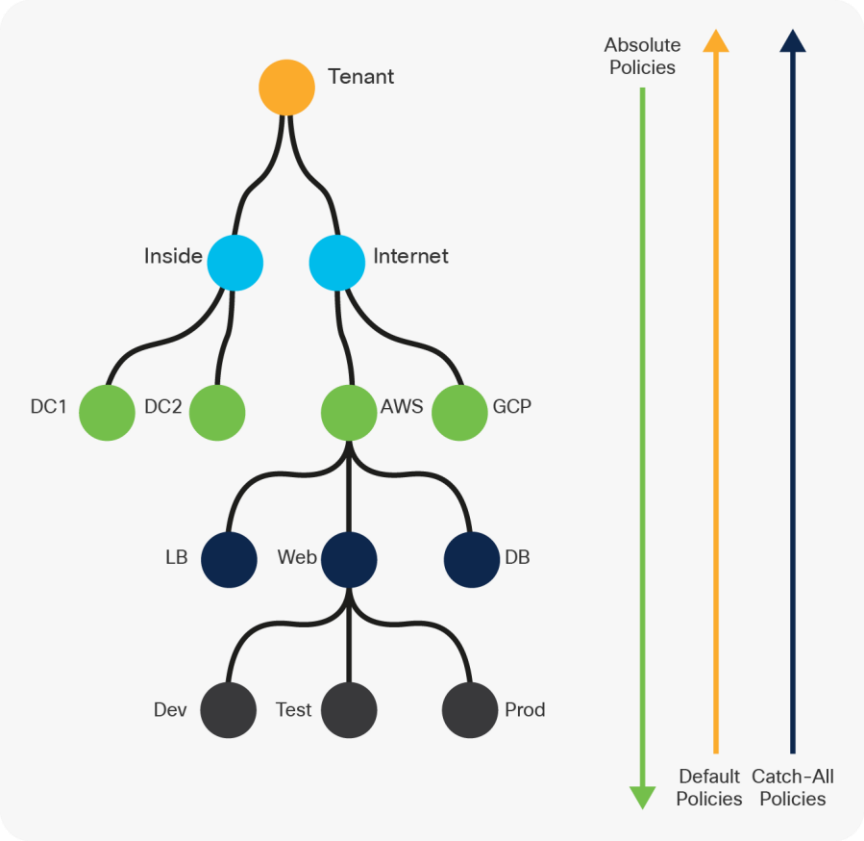

Figure 1.
Building a workload policy from scope tree in Cisco Tetration

By combining these capabilities of centralized policy governance with the workload annotation and the ability to create specific zero-trust policy requirements via Terraform, the solution allows developer flexibility and agility, retains IT operations and security governance, and provides for workload portability. These combined capabilities allow business to move fast while controlling risk.

Why choose Cisco Tetration for microsegmented firewalling of workloads

Cisco Tetration offers holistic workload protection for multicloud data centers by enabling a zero-trust model using segmentation. This approach allows for identifying security incidents faster, contains lateral movement, and reduces the attack surface available to today's sophisticated security attacks. Cisco Tetration's infrastructure-agnostic approach supports both on-premises and public cloud workloads.

Cisco Tetration provides:

- Users the ability to create a ubiquitous security policy across all assets in their on-premises and public cloud workloads
- Application dependency mapping with a machine-learning algorithm to constantly assess appropriate rules
- Visibility to test rule changes before committing them, increasing confidence in changes, and allowing work to happen faster
- Zero-trust microsegmentation regardless of workload location

- Forensic review of past events for analysis
- Connectivity to Kubernetes, VMware vCenter, AWS, on-premises traffic flow devices, load balancers, firewalls, VPNs, and identity services engines for annotations and information ingestion
- Outbound alerting to endpoints via email, Slack, PagerDuty, syslog, and Amazon Kinesis as well as bidirectional connectivity via a Kafka message bus

For the developer, Cisco Tetration provides a robust set of RESTful API calls to allow developers to interact with the platform via their native toolsets. Cisco provides a Terraform Provider, Ansible Modules, and a Python library for interacting with the Tetration platform. Cisco also provides learning labs for each of the management methods on Cisco DevNet. Cisco maintains an open source community organization for the development and sharing of tooling to integrate Tetration into your environment on GitHub.

Why choose HashiCorp Terraform

Terraform is a broadly used tool that developers use to provision infrastructure as code workloads across multiple infrastructure providers. It allows development teams to write, test, and deploy workloads into hybrid cloud data centers such as AWS and on-premises using HashiCorp Configuration Language directly from their CI/CD pipelines. By leveraging the Terraform provider for Cisco Tetration, developers can push annotations and zero-trust policy rules with the workload at the time of deployment.

Why choose Amazon Web Services

Amazon Web Services (AWS) allow the business to meet their need for flexibility and agility in workload deployment. AWS and Cisco Tetration have integrated multiple components of the solution to provide users with additional capabilities. AWS workload annotations can automatically be fed into Cisco Tetration via a native integration into AWS APIs to be able to enforce security policy based on AWS data such as Auto Scaling group name or EC2 tags. AWS VPC Flow Logs can also be consumed by the Cisco Tetration Ingest Appliance to use VPC information, including flows allowed or denied from the VPC, within Tetration for insights into your AWS instances. In addition, the Cisco Tetration Kinesis connector enables streaming of Tetration alerts on Amazon Kinesis for automating queries and analysis of the alerts.

How the solution works

The solution relies on the Terraform provider for Cisco Tetration, which is available at https://github.com/tetration-exchange/terraform-provider as well as accounts on AWS and Cisco Tetration. The Cisco Tetration Terraform provider is based on HashiCorp Terraform v0.12.X.

The first step is to include the Cisco Tetration agent in the image build for the VM to be deployed by Terraform. Next, development and IT Ops/security need to agree on the appropriate annotations to be applied for proper scope placement and policy application. Some examples of annotations can be tagging for Dev, Test, and Prod or by tagging desired locality such as AWS or DC1. Additionally, fields such as application name and application role like frontend or DB can be applied to further help in scope placement. Tags from external orchestrators such as AWS, Kubernetes, or VMware VCenter are applied if those systems are integrated.

Finally, to allow the developers to be able to open permitted ports for their applications before IT operations and security close all further un-required ports, Terraform needs to be able to push policy requirements to the Cisco Tetration platform when running the terraform apply command. For this, Cisco provides filter and policy controls in their Terraform provider, which we will detail below.

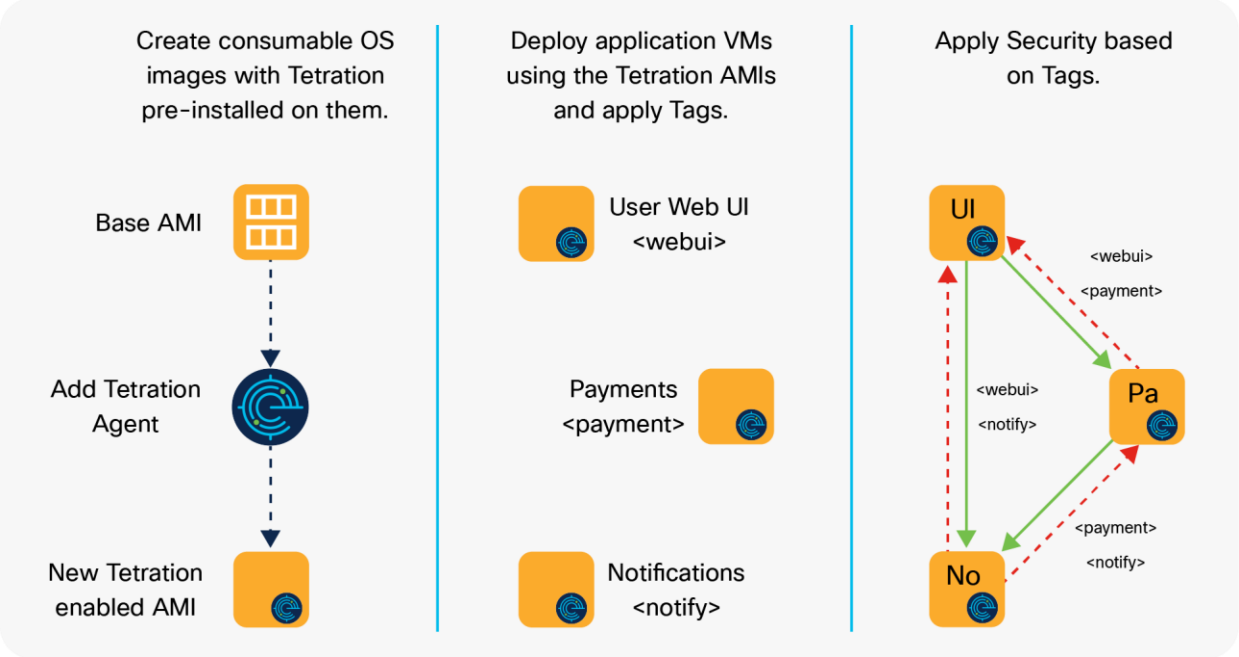

Figure 2.
Steps to automated policy deployment

The high-level review of the solution, below, is further detailed in the Cisco DevNet Learning Lab. Here we are assuming you understand the basic format of a Terraform module and secrets management and have working accounts for Cisco Tetration and AWS. We also assume you are comfortable creating and deploying workloads with the Cisco Tetration agent in AWS using Terraform via a CI/CD pipeline. For more background and information on this and other topics, please see all the available Cisco DevNet Learning Labs.

Tagging annotations in Tetration using Terraform

Adding annotations to workloads is as simple as adding the list of tags to a workload deployment into AWS. Consider the following code and see how we can add the annotations in AWS using Terraform for the application 'Name' and the 'Project' name when we are pushing the application:

Table 1. Code Block 1

```
provider "aws" {
  region = "us-west-2"   # Or another region of your choice
}

data "aws_ami" "cisco" {
  most_recent = true
  owners = [
    "self"]

  filter {
    name = "name"
    values = ["cisco-tetration-image"]
# Example name of the AMI image with the Tetration agent installed.
  }
}

resource "aws_instance" "sample_app_web" {
  ami                    = data.aws_ami.cisco.id
# Matches the "data" line above and gets the ID of the AMI called "cisco-tetration-image"
  instance_type          = "t2.micro"
  key_name               = "cisco-terraform-tetration-provider"
# Key created in AWS - specific ssh keys for our instances (optional)
  security_groups        = ["cisco-terraform-tetration-provider"]
#  This is a security group created in AWS (optional)

  tags = {
    Name                 = "sample-app-web"
#  The name of the instance
    Project              = "cisco-tetration-automation-poc"
#  Provide a tag so you can find your instances later
  }
}

resource "aws_instance" "sample_app_db" {
  ami                    = data.aws_ami.cisco.id
  instance_type          = "t2.micro"
  key_name               = "cisco-terraform-tetration-provider"
  security_groups        = ["cisco-terraform-tetration-provider"]

  tags = {
    Name                 = "sample-app-db"
    Project              = "cisco-tetration-automation-poc"
  }
}
```

Code Block 1 example shows tag annotations applied to the EC2 in AWS. Such tags will need to be pulled into Cisco Tetration via an External Orchestrator connection to AWS. We can also use the Terraform provider for Tetration from Cisco to push the annotations natively to the Cisco Tetration system. By adding the following code to our main.tf file, we can add the tags specific to the application for '**Environment**,' '**Datacenter**,' '**Application**,' and '**Tier**.'

Table 2. Code Block 2

```
resource "tetration_tag" "tag_app" {
  tenant_name = "Orgname"
# This is the tenant name on Tetration (usually your organization name)
  ip          = aws_instance.sample_app_web.private_ip
# This gathers the IP Address from the Web VM resource - previous page.
  attributes = {
    Environment = "Dev"
    Datacenter  = "AWS"
    Application = "Sample_App"
    Tier = "Web"
  }
}
resource "tetration_tag" "tag_db" {
  tenant_name = "Orgname"
  ip                  = aws_instance.sample_app_db.private_ip
  attributes = {
    Environment = "Dev"
    Datacenter  = "AWS"
    Application = "Sample_App"
  }
}
```

The IT operations and security teams can use these attributes to create firewall policies for the workload in the Cisco Tetration system to create appropriate firewalling policies and have them pushed to the workload for enforcement.

Creating allow-list policy in Tetration using Terraform

While the above **Code Block 2** example allows for policy creation based on attributes that are pushed to Cisco Tetration via the Terraform deployment, it still relies on the IT operations and security teams to create the policy for the application workload outside of the deployment. To remove this dependency, we have included the capability to push actual policy rules along with the code deployment for a fully automated, zero-trust micro-segmented application architecture. This deployment methodology allows for automated testing as well as a review step in the CI/CD pipeline automation via the merge/pull request review to check if the policy is correct before deployment.

The Terraform provider for Cisco Tetration allows for us to create filters and use these filters to apply zero-trust policy to our workload firewall. For example, we can create filters based on the IP address of the workload we receive back from AWS at the EC2 instance creation time.

Table 3. Code Block 3

```
resource "tetration_filter" "filter_app_web" {
  name         = "web_tier"
# This is the name of the Tetration filter and will be searchable.
  query_type   = "eq"
  query_field  = "ip"
  query_value  = aws_instance.sample_app_web.private_ip
# We are using the IP Address that was given to the web server.
  app_scope_id = "5be415034123452e23af29b3"
# This is the main scope ID or the ID of your tenant name in Tetration
}

resource "tetration_filter" "filter_app_db" {
  name         = "db_tier"
  query_type   = "eq"
  query_field  = "ip"
  query_value  = aws_instance.sample_app_db.private_ip
  app_scope_id = "5be4150312345f2e23af29b3"
}
```

The **Code Block 3** example shows us that once the filters are created, we can apply them to traffic to create policy rules for our deployment. In Cisco Tetration, policies are applied in a directional manner using filters as either Consumer (source) or Provider (destination). In this example, we are applying four policies for our workload to accomplish the following policies:

- Create an absolute policy that denies traffic from the web app to reach the database on port 23
- Create a default policy that allows traffic from the web app to reach the database on port 443
- Create a default policy that allows traffic from the web app to reach the database on port 3306
- Create a catch-all policy that denies all other traffic for the workload, thereby creating the zero-trust nature of the policy to provide microsegmentation from all other workloads.

As policies are unidirectional, you would also need rules with the consumer as the '**db**' and the provider as the '**web app**' if your application architecture required the 'db' to originate requests to the 'web app' on a particular port before applying the catch-all deny. These are omitted from this example.

Table 4. Code Block 4

```
resource "tetration_application" "sample_application_policy" {
  app_scope_id         = "5be415034912345623af29b3"
# This is an example of the main scope ID or the ID of your tenant
  name                 = "My Sample Application"
# This name will show up in the Tetration GUI.
  description          = "A sample application policy!"
  alternate_query_mode = false
  strict_validation    = true
  primary              = false

  catch_all_action = "DENY"
# Your default action for all other traffic.

  absolute_policy {
    consumer_filter_id = tetration_filter.filter_app_web.id
# The Terraform object name and the ID that was assigned from Tetration when it was created
above.
    provider_filter_id = tetration_filter.filter_app_db.id
    action             = "DENY"
    layer_4_network_policy {
      port_range = [23, 23]
      protocol   = 6
    }
  }

  default_policy {
    consumer_filter_id = tetration_filter.filter_app_web.id
    provider_filter_id = tetration_filter.filter_app_db.id
    action             = "ALLOW"
    layer_4_network_policy {
      port_range = [443, 443]
      protocol   = 6
    }
  }

  default_policy {
    consumer_filter_id = tetration_filter.filter_app_web.id
    provider_filter_id = tetration_filter.filter_app_db.id
    action             = "ALLOW"
    layer_4_network_policy {
      port_range = [3306, 3306]
      protocol   = 6
    }
  }
}
```

The **Code Block 4** example uses the AWS-provided IP addresses to create policy rules, but we can also use annotations to create policy dynamically. If we wish to consider our example of using '**tag_app**' and '**tag_db**' in the **Code Block 2** example, we can use our self-defined tag for 'Environment' as follows:

Table 5. Code Block 5

```
resource "tetration_filter" "filter_app_dev" {
  name         = "dev_tier"
# This is the name used inside of Tetration and will show up on the screen.
  query_type   = "eq"
  query_field  = "Environment"
  query_value  = "Dev"
  app_scope_id = "5be41512345d8f2e23af29b3"
# This is the main scope ID or the ID of your tenant name in Tetration.
}

resource "tetration_filter" "filter_app_prod" {
  name         = "prod_tier"
  query_type   = "eq"
  query_field  = "Environment"
  query_value  = "Prod"
  app_scope_id = "5be4150123458f2e23af29b3"
# This is the main scope ID or the ID of your tenant name in Tetration.
}
```

The **Code Block 5** example shows that we can then apply the filters as we did in the **Code Block 4** example, but we would replace the name of the **'consumer_filter_id'** or **'provider_filter_id'** with the new filter names of **'tetration_filter.filter_app_dev.id'** and **'tetration_filter.filter_app_prod.id'**.

Figure 3.
Automated policy by tags in Cisco Tetration GUI

In our examples for **Code Block 3** and **Code Block 5**, we used simple queries with a single layer of matching based on a key equaling a tag. The Terraform provider for Cisco Tetration allows us to use not only a broad set of query **'type'** fields, but also **'and/or'** notation to make more complex queries.

The available query 'type' fields for comparison are:

- "**eq**"—meaning equality
- "**ne**"—meaning inequality
- "**lt**"—meaning less than
- "**lte**"—meaning less than or equal to
- "**gt**"—meaning greater than
- "**gte**"—meaning greater than or equal to
- "**range**"—meaning within a list of ['**start value**', '**end value**']
- "**in**"—meaning a member of a list of values
- "**contains**"—meaning the key is a subset of the value
- "**subnet**"—meaning address resides with a CIDR block using slash notation

In addition to the comparators, the system allows the logical use of 'type' fields:

- "**not**"—which inverts a filter
- "**and**"—which compounds two or more filters
- "**or**"—which matches any of a set of filters

By using the logical operators with the comparator operators, we can create a complex query that allows fine-grained placement of workloads in the appropriate security context.

Table 6. Code Block 6

```
resource "tetration_filter" "filter" {
  name         = "Terraform created filter"
  query        = <<EOF
            {
               "type": "and",
               "filters": [
                  {
                     "field": "vrf_id",
                     "type": "eq",
                     "value": 733059
                  },
                  {
                     "type": "or",
                     "filters": [
                        {
                           "field": ""Environment"",
                           "type": "eq",
                           "value": "Dev"
                        },
                        {
                           "field": "ip",
                           "type": "subnet",
                           "value": "10.0.0.0/8"
                        }
                     ]
                  }
               ]
            }
          EOF
  app_scope_id = "5ed61234567d4f55eb5c585c"
  primary      = true
  public       = false
}
```

Using the example code in **Code Block 6**, we can create complex filter queries, as we are creating a filter that checks that a flow is within our **'vrf id' and** is either tagged as being in our **'Dev'** environment **or** is in the 10.0.0.0/8 network.

Conclusion

With the proposed solution discussed above, we can take security policy and “shift left” to allow a developer to open the necessary permitted ports while inheriting the IT operations and security policies using Terraform and Cisco Tetration. With the proper policy definitions and agreed-upon annotations combined with appropriate review and approval processes in our deployment pipelines, we can now create the security requirements for the workload no matter the deployment location. We can integrate with the AWS tooling to make security decisions based on AWS annotations, flow logs, and even output to AWS management tools. This solution allows the business to achieve their goals of agility and scalability while IT operations and security can secure the workloads wherever they may reside.

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at **https://www.cisco.com/go/offices**.

Printed in USA C11-744318-00 02/21