Link: <https://www.hashicorp.com/blog/a-guide-to-cloud-cost-optimization-with-hashicorp-terraform>

If you’re reading this, chances are you are in DevOps (or some type of engineering) and you are wondering: “Why on earth do I care about cloud cost optimization? That’s not my job, I’m not in finance, right?” Wrong!

Engineers are becoming the new cloud financial controllers as finance teams begin to lose some of their direct control over new fast-paced, on-demand infrastructure consumption models driven by cloud. So the question becomes: What are the people, processes, and technologies I can use to navigate this sea change?

If you are interested in defining a new role, automating your newfound responsibility, and implementing a process for cloud cost optimization with HashiCorp Terraform Cloud, read on.

This guide will provide:

* A RASCI model assigning responsibilities for your team that manages overall cloud posture and costs
* A visualization of the cloud cost management within a Terraform provisioning workflow
* Planning recommendations for cloud cost management and forecasting with Terraform
* An introduction to using Terraform’s cost estimation features
* Instructions on how to integrate and use cloud-vendor and third-party cost optimization tools in a Terraform workflow
* Examples of Terraform’s policy as code framework, Sentinel, which can automatically block overspending with rules around cost, instance types, and tags.

The majority of features reviewed in this article will focus on Terraform paid functionality such as Cost Estimation and Governance & Policy but the core use case around cost optimization can be achieved solely with the open source version.

## A Survey of Cloud Waste

With the continuous shift to consumption-based cost models for infrastructure and operations; i.e. Cloud Service Providers (CSPs), you pay for what you use but you also pay for what you provision and don’t use. If you do not have a process for continuous governance and optimization, then there is a huge potential for waste.

A recent cloud spending <https://www.densify.com/resources/top-cloud-priorities> found that:

* 45% of the organizations reporting were over budget for their cloud spending.
* More than 55% of the respondents are using either cumbersome manual processes, or simply do not implement actions and changes to optimize their cloud resources.
* 34.15% of respondents believe they can save up to 25% of their cloud spend and 14.51% believe they can save up to 50%. Even worse, 27.46% said, “I don’t know”.

First, let’s unpack why there is an opportunity and then get to the execution.

## Why Engineers are Becoming the Financial Controllers of Cloud Spend

In moving to the cloud, most organizations have put thought into basic governance models where a team, sometimes referred to as the Cloud Center of Excellence, looks over things like strategy, architecture, operations, and, yes, cost. Most of these teams contain a combination of IT management and cloud technical specialists from common IT domains and finance. Finance is primarily charged with cost planning, migration financial forecasting, and optimization.

Due to financial pressures, they tend to say: “We need to get a handle on costs, savings, forecasting, etc.” but have no direct control over costs. It is now engineers that directly manage infrastructure and the costs.

The business case is simple, it is a financial paradigm shift where:

* Engineers are not only responsible for operations but also costs.
* Engineers now have the tools and capabilities to automate and directly manage cost controls.
* Cost planning and estimation for running cloud workloads are not easily understood or forecasted by Finance.
* Traditional forms of financial budgeting and on-prem hardware demand planning (such as contract-based budgets and capitalized purchases) do not account for cost variability in consumption-based (i.e. Cloud) models.

Finance lacks control in the two primary areas of cost-saving:

* **Pre-provisioning:** Limited governance and control in the resource provisioning phase.
* **Post-provisioning:** Limited governance and control in enforcing infrastructure changes for cost savings.

In the following sections, we will define the people, processes, and technologies associated with managing cloud financial practices with Terraform.

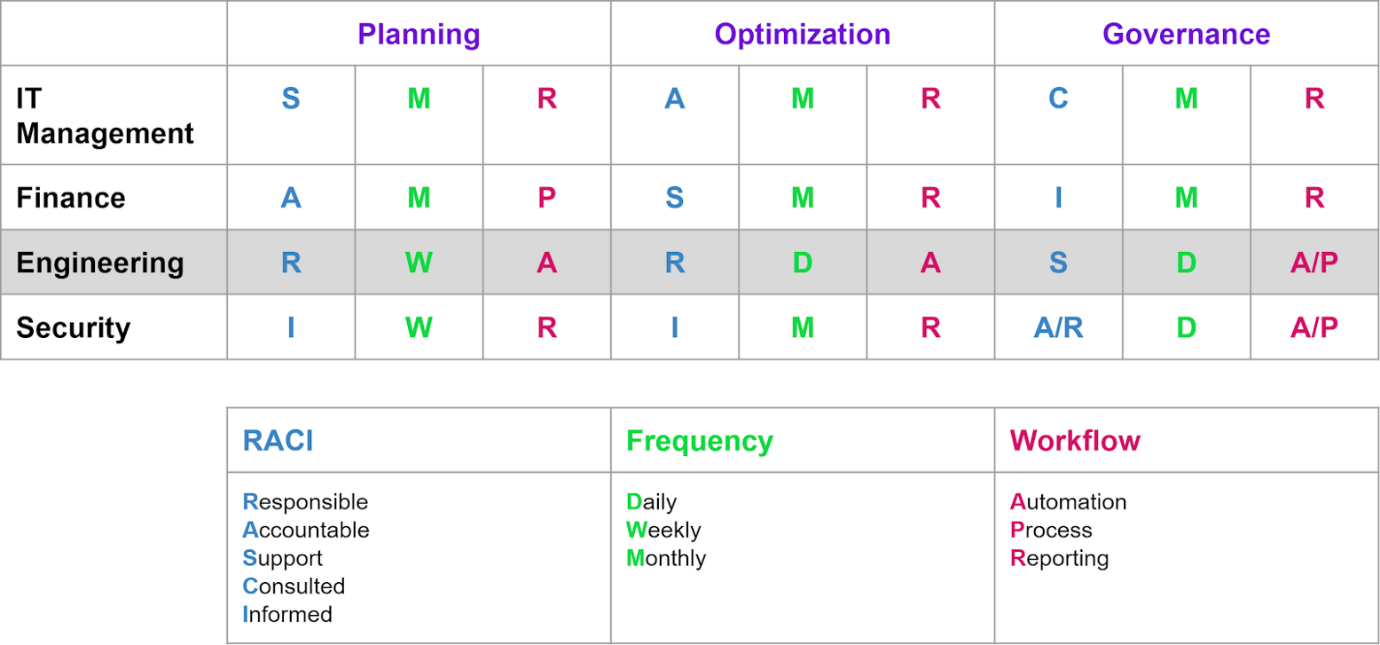
## A RASCI Chart Example

To simplify things, we will assume there is some sort of team — i.e. the Cloud Center of Excellence — that is responsible for managing the overall cloud posture.

On this team there are four core roles:

* IT Management
* Finance
* Engineering (consisting of DevOps and Infrastructure & Operations)
* Security

I’ve built a RASCI model that can be used as a baseline of expectations for this team. This is a model I have used, along with similar models, to define the roles and responsibilities of the Cloud Center of Excellence for many organizations.



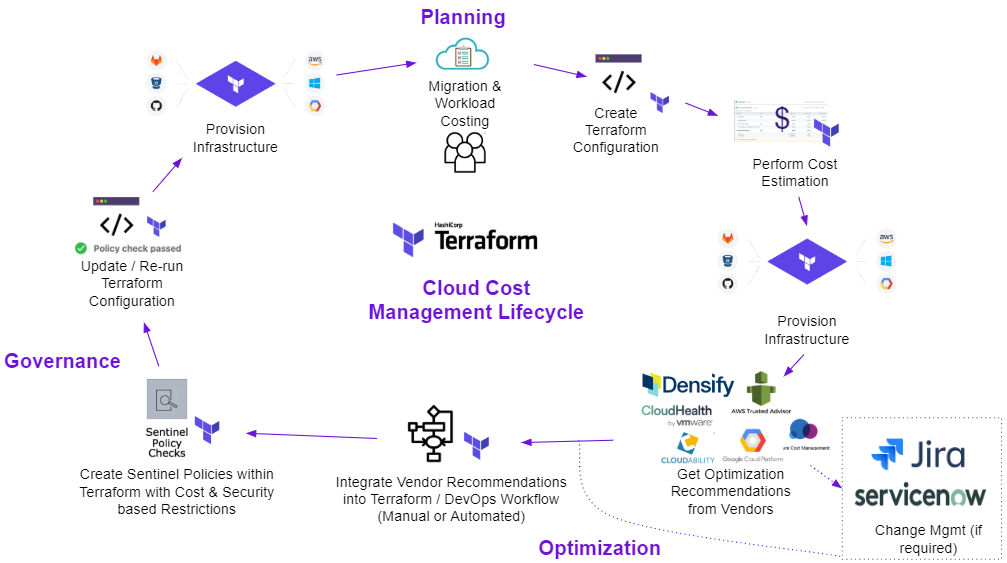
The three section headings in this table are:

* **Planning** — Relating to pre-cloud migration & ongoing cost forecasting
* **Optimizing** — Operationalizing and realizing continuous cost savings
* **Governance** — Ensuring future cost savings and waste avoidance

As you can see, engineering has a higher level of responsibility in today’s infrastructure operations, and the Frequency and Workflow columns will likely denote significant changes for many organizations.

## Planning, Optimization, and Governance

Now the next question: How can engineers use Terraform at each level of the cloud cost management process to deliver value and minimize additional work? To get started, see how the visualization illustrates Terraform’s place in the cloud cost management lifecycle. (Start at the top with the “Planning” phase)



To summarize the steps:

* Start by identifying workloads that are migrating to the cloud
* Create Terraform configuration
* Run terraform plan to perform cost estimation
* Run terraform apply to provision the resources
* Once provisioned, workloads will run and vendor tools will provide optimization recommendations
* Integrate a vendor’s optimization recommendations into Terraform and/or your CI/CD pipeline
* Investigate/analyze optimization recommendations and implement Terraform’s [Sentinel policies](https://docs.hashicorp.com/sentinel/concepts/policy-as-code/) for cost and security controls
* Update Terraform configuration and run plan & apply
* Newly optimized and compliant resources are now provisioned

### Planning — Pre-Migration and Ongoing Cost Forecasting

Cloud migrations require a multi-point assessment to determine if it makes sense to move an application/workload to the cloud. Primary factors for the assessment are:

* Architecture
* Business case
* Estimated cost for the move
* Ongoing utilization costs budgeted/forecasted for the next 1–3 years on average

Since engineers are now taking on some of these responsibilities, it makes sense to use engineering tools to handle them. Terraform helps engineers take on these new responsibilities with [cost estimation](https://www.terraform.io/docs/cloud/getting-started/cost-estimation.html), which helps calculate the infrastructure costs of each provisioning run based on the actual deployment plan.

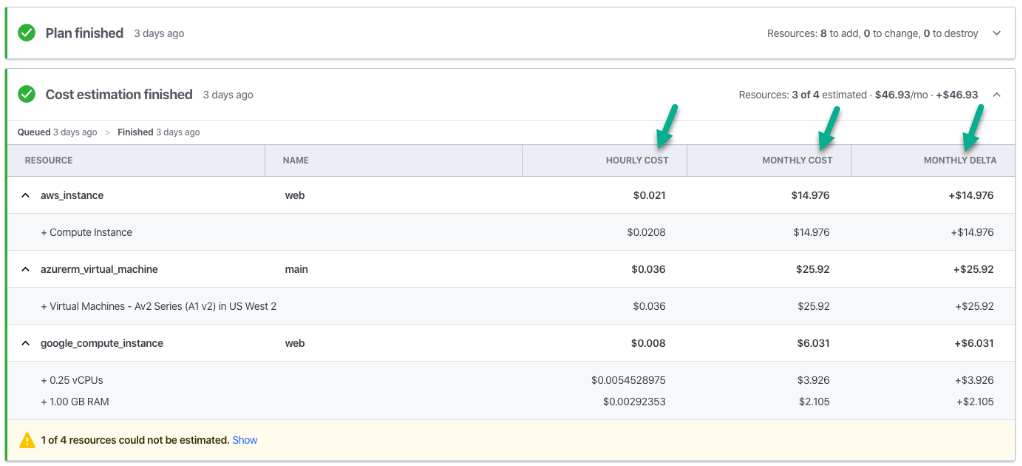
Using Terraform configuration files as a standard definition of how an application/workload’s cost is estimated, you can now use [Terraform Cloud & Enterprise APIs](https://www.terraform.io/docs/cloud/api/cost-estimates.html) to automatically supply finance with estimated cloud financial data or use Terraform’s user interface to provide finance direct access to review costs. By doing this, you can help eliminate many slower oversight processes.

#### Planning Recommendations:

* Use Terraform configuration files as the standard definition of cloud cost planning and forecasting across AWS, Azure, and GCP, and provide this information via the Terraform API or [role-based access controls](https://www.terraform.io/docs/cloud/workspaces/access.html) within the Terraform user interface to provide financial personas a self-service workflow.
* Note: Many organizations conduct planning within Excel, Google Sheets, and Web-based tools. To make data usable within these systems we would recommend using [Terraform’s Cost Estimates API](https://www.terraform.io/docs/cloud/api/cost-estimates.html) to extract the data.
* Use [Terraform modules](https://www.terraform.io/docs/configuration/modules.html) as standard units of defined infrastructure for high-level cost assessments and cloud demand planning
  + Example: Define a standard set of modules for a standard Java application, so module A + B + C = $X per month. We plan to move 5 Java apps this year. This can be a quick methodology to assess potential application run costs prior to defining the actual Terraform configuration files.
* Use Terraform to understand application/workload financial growth over time, i.e. cloud sprawl costs.
* Attempt to structurally align [Terraform Organization, Workspace, and Resource](https://www.terraform.io/docs/cloud/users-teams-organizations/organizations.html) naming conventions to the financial budgeting/forecasting process.

There is a guide for [getting started with Terraform Cost Estimation](https://learn.hashicorp.com/tutorials/terraform/cost-estimation) on the HashiCorp Learn site (this feature is available on Terraform Cloud’s Team & Governance Tier and above). Once enabled, when a Terraform plan is run, Terraform will reach out to the AWS, Azure, and/or GCP cost estimation APIs to present the estimated cost for that plan, which can be used accordingly within your financial workflow. You can also export this estimation report as JSON.

#### Example of Cost Estimation Output in Terraform



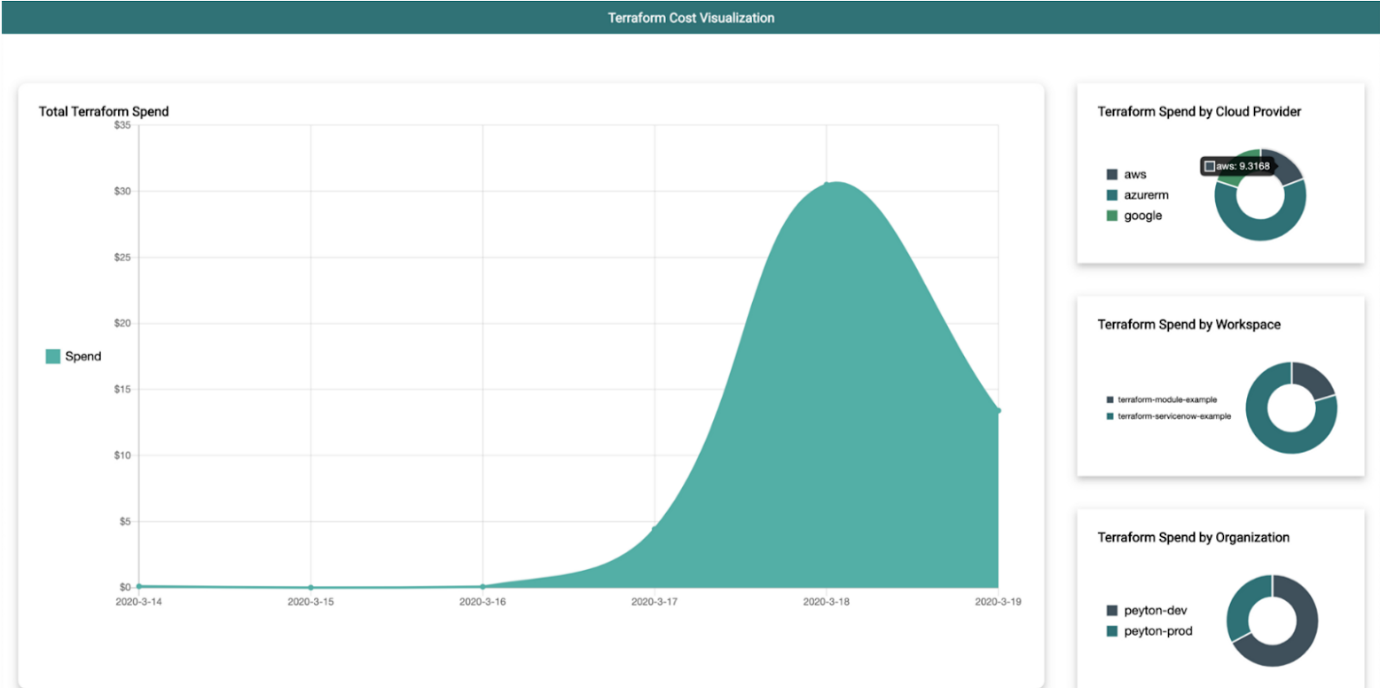
#### Example of the Cost Estimation API JSON Payload from Terraform



Be aware that Terraform Cost Estimation provides costs based on a workspace view. If you would like a higher level, cross-workspace view, you will need to leverage the Terraform Cost Estimation API in tandem with a reporting tool of your choice.

#### Tools for Multi-Cloud Cost Visualization

Peyton Casper, a HashiCorp senior solutions engineer has built a simple open source tool that can give you that higher level, cross-workspace view. The tool is called Tint



Example dashboard from Tint

If you have heavier requirements for reporting, or you already have an existing corporate reporting product (e.g. Microsoft BI, Tableau, etc.), Terraform’s cost estimation data will work with these solutions as well.

### Optimizing — Operationalizing and Realizing Continuous Cost Savings

Optimization is the continued practice of evaluating the costs and benefits ratio of your current infrastructure usage. The cloud vendors (AWS, Azure, GCP, etc.) and other third-party tools can start you off with some optimization recommendations but some organizations don’t always take advantage of the recommendations.

Engineering does not always engage with these optimization systems, leaving them with no feedback mechanism. If they are engaged with these systems, there is often a high level of manual intervention needed.

#### Automating Optimization Insights into the Provisioning Workflow

It is safe to say that the major CSPs and the vast majority of third-party tools allow you to export optimization recommendations via an API or an alternative method (references: [AWS](https://docs.aws.amazon.com/awssupport/latest/user/trustedadvisor.html), [Azure](https://docs.microsoft.com/en-us/rest/api/cost-management/), [GCP](https://cloud.google.com/recommender/docs/apis)). For the purposes of this guide, we are going to focus on the most basic approach to automate optimization data ingestion, which will come directly from the CSPs or from third parties such as [Densify](https://www.hashicorp.com/integrations/densify/terraform) who maintain a [Terraform Module](https://registry.terraform.io/modules/densify-dev/optimization-as-code/null/1.2.0). I will use Densify in my examples that follow. (Note that there are also many HashiCorp users that [create their own Terraform providers](https://www.hashicorp.com/resources/creating-terraform-provider-for-anything/) for similar processes.)



Densify EC2 optimization report example

The concepts and code can be used as a model for your own deployments. Please note that each vendor provides a different set of recommendations, but all of them provide insights on compute, so we will focus on that. Any insight that you receive (e.g. compute, storage, DB, etc.) can be consumed based on the pattern below.

#### Basic Patterns for Consuming Optimization Recommendations

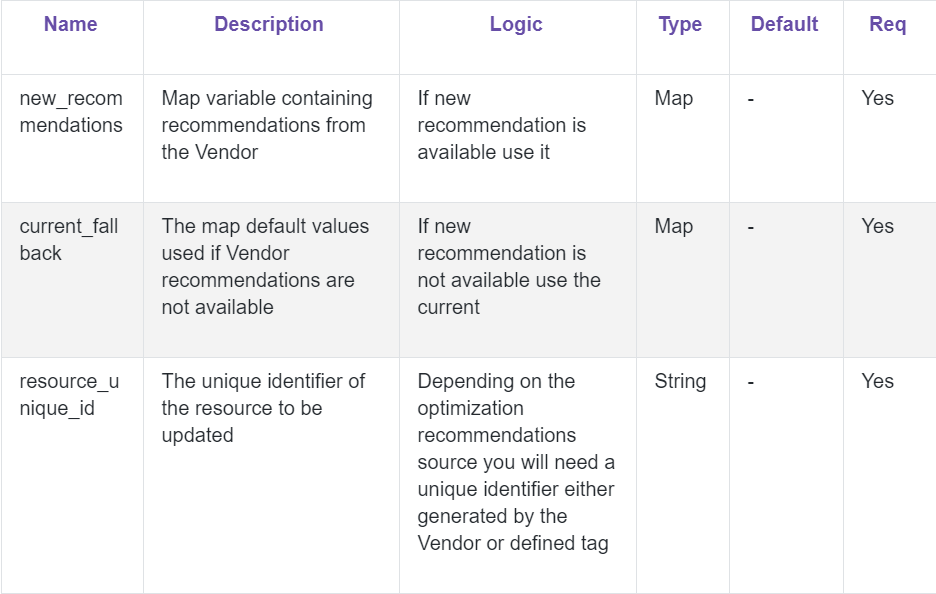
To establish a mechanism for Terraform to access the optimization recommendations, we see several common patterns:

* **Manual Workflow** — Review of optimization recommendations from the providers portal and manually update Terraform files. Since there is no automation, this is not optimal, but a feedback loop for optimization must start somewhere!
* **File Workflow** — Create a mechanism where optimization recommendations are imported into a local repository via a scheduled process (usually daily).
  + For instance, Densify customers use a script to export recommendations into a densify.auto.tfvars file and it is downloaded and stored in a locally accessible repository. Then the Terraform [lookup](https://www.terraform.io/docs/configuration/functions/lookup.html) function is used to look up specific optimization updates that have been set as variables.
* **API Workflow** — Create a mechanism for optimization recommendations to be extracted directly from the vendor and stored within an accessible data repository using Terraform’s [http data\_source](https://registry.terraform.io/providers/hashicorp/http/latest/docs/data-sources/data_source) functionality to perform the dataset import reference.
* **Ticketing Workflow** — This workflow is similar to the file and API workflows but some organizations insert an intermediary step where the optimization recommendations first go to a change control system like ServiceNow or Jira. Within these systems there is workflow and approval logic built-in where a flag is set for acceptable change and is passed as a variable to be consumed later in the process.

#### Optimization as Code: Terraform Code Update Examples

In any of these cases, especially if automation is taking place, it will be important to maintain key pieces of resource data as variables. The optimization insight tools will provide a size recommendation for resources or services (i.e. compute, DB, storage, etc.). In this example, we will use compute resources, but the example is representative of all.

At a minimum, I recommend that you have three variables set to perform the optimization update in Terraform with some basic logic: new\_recommendations, current\_fallback, and resource\_unique\_id.



As I mentioned, I’ll be using Densify. You can find the [Densify Terraform module](https://registry.terraform.io/modules/densify-dev/optimization-as-code/null/1.2.0) via the Terraform Registry and the [Densify-dev](https://github.com/densify-dev/terraform-null-optimization-as-code) GitHub repo.

In this first snippet, you will see some basic updates of Terraform code with variables and logic to get started. Below is an example of the variables created.

variable "densify\_recommendations"{

description = "Map of maps generated from the Densify Terraform Forwarder. Contains all of the systems with the settings needed to provide details for tagging as Self-Aware and Self-Optimization"

type = "map"

}

variable "densify\_unique\_id"{

description = "Unique ID that both Terraform and Densify can use to track the systems."

}

variable "densify\_fallback"{

description = "Fallback map of settings that are used for new infrastructure or systems that are missing sizing details from Densify."

type = "map"

}

In the next snippet, you will see updates with the Terraform lookup function to look up the local optimization recommendations file — densify.auto.tfvars — for changes. The optimization recommendations can also be auto-delivered by using webhooks and subscription [APIs](https://www.densify.com/dev).

locals{

temp\_map = "${merge(map(var.densify\_unique\_id, var.densify\_fallback),var.densify\_recommendations)}"

densify\_spec = "${local.temp\_map[var.densify\_unique\_id]}"

cur\_type = "${lookup(local.densify\_spec,"currentType","na")}"

rec\_type = "${lookup(local.densify\_spec,"recommendedType","na")}"

savings = "${lookup(local.densify\_spec,"savingsEstimate","na")}"

p\_uptime = "${lookup(local.densify\_spec,"predictedUptime","na")}"

ri\_cover = "${lookup(local.densify\_spec,"reservedInstanceCoverage","na")}"

appr\_type = "${lookup(local.densify\_spec,"approvalType","na")}"

recommendation\_type = "${lookup(local.densify\_spec,"recommendationType","na")}"

Lastly, you will want to insert some logic to ensure that you are properly handling the usage reference. If a recommendation is available, use it. Otherwise keep the current one.

Densify also adds some code in as part of a change control process for their customers that are using a change control/ticketing system. They have an option to first pass the optimization recommendation to be approved in one of these external systems and then pass an approval flag in as a variable to ensure that it is an approved change.

instance\_type = "${local.cur\_type == "na" ?

"na" :

local.recommendation\_type == "Terminate" ?

local.cur\_type:

local.appr\_type == "all" ?

local.rec\_type :

local.appr\_type == local.rec\_type ?

local.rec\_type :

local.cur\_type}"

For customers not using a third party approval system, the recommendation’s changes will be visible on the Terraform plan. Similarly, you can also manually update a variable such as appr\_type = false to avoid using the recommendation. Or you can use other similar methods via [Feature Flags](https://www.hashicorp.com/blog/terraform-feature-toggles-blue-green-deployments-canary-test/) and [conditional expressions](https://www.terraform.io/docs/configuration/expressions.html#conditional-expressions) in Terraform to control applied functionality.

The takeaway here is that we now have a defined process that can be partially or fully automated to make fast, code-defined changes to our infrastructure environment to optimize and save money.

### Governance — Ensuring Future Cost Savings

The last and critical component of the cloud cost management lifecycle is having guardrails to stop cost overruns and provide a feedback loop. I have had this conversation with many organizations that have done optimization exercises only to have their costs shoot back up because they didn’t put preventative controls in place from the start.

In Terraform, you can automate this feedback loop with [Sentinel](https://docs.hashicorp.com/sentinel/concepts/policy-as-code/), a policy as code framework embedded within Terraform for governance & policy (Sentinel can be used in other HashiCorp products as well).

#### Cost Compliance as Code = Sentinel Policy as Code

Sentinel includes a domain specific language (DSL) to write policy definitions that evaluate any and all data defined within a Terraform file. You can use Sentinel to ensure provisioned resources are: secure, tagged, and are within usage and cost constraints.

For costs, Terraform customers implement policy primarily around three areas: (but remember, you’re not limited to just these three… you can get creative):

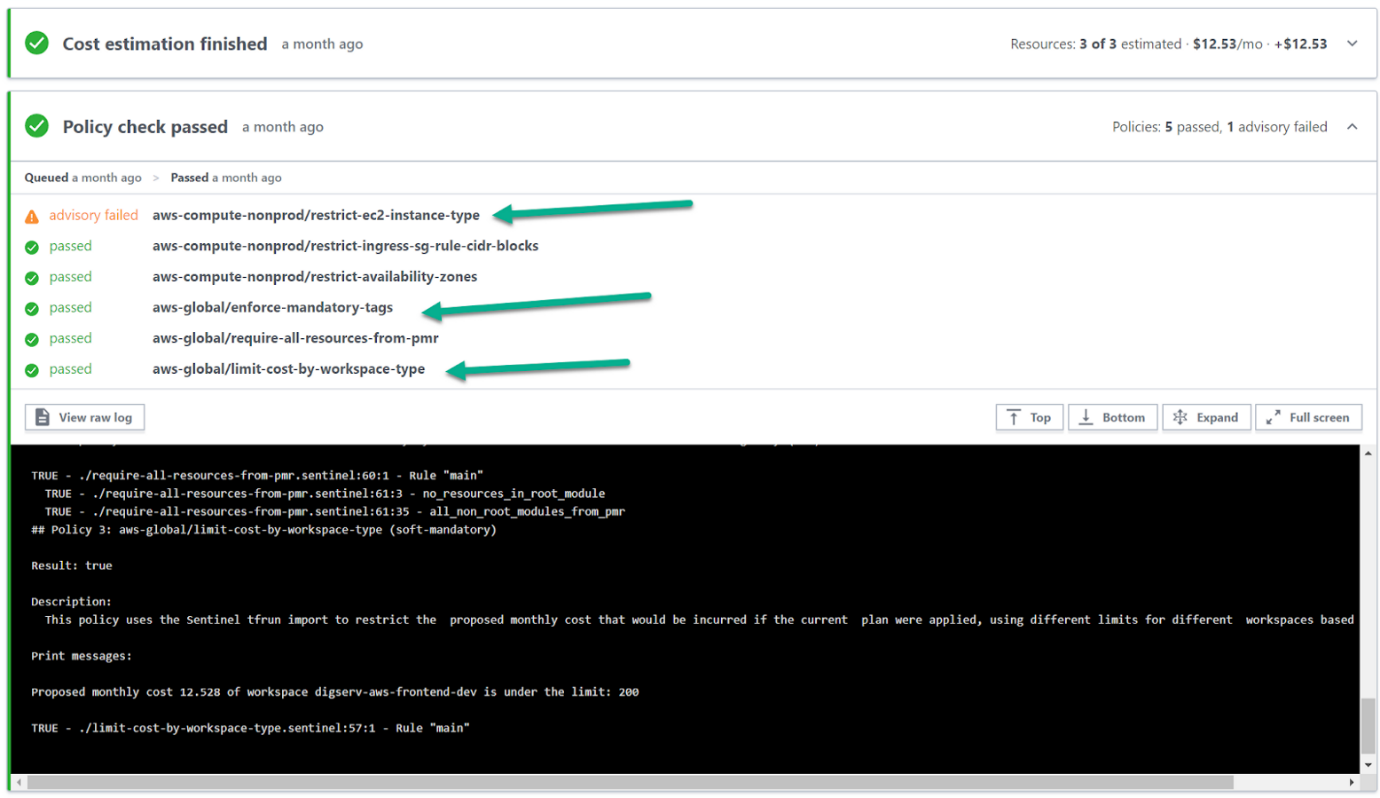
**Cost Control Areas:**

* **Amount** — Control the amount of spend
* **Provisioned size** — Control the size/usage of the resource
* **Time to live** — Control the time-to-live (TTL) of the resource

In all three of these areas, you are able to apply policy controls around things like Terraform workspaces (e.g. apps/workloads), environments (e.g. prod, test, dev), and tags to optimize resources and avoid unnecessary spending.

The following is an example Sentinel policy output when running terraform plan. We will focus on three policies:

1. [aws-global/limit-cost-by-workspace-type](https://www.hashicorp.com/blog/a-guide-to-cloud-cost-optimization-with-hashicorp-terraform#aws-global-limit-cost-by-workspace-type)
2. [aws-compute-nonprod/restrict-ec2-instance-type](https://www.hashicorp.com/blog/a-guide-to-cloud-cost-optimization-with-hashicorp-terraform#aws-compute-nonprod-restrict-ec2-instance-type)
3. [aws-global/enforce-mandatory-tags](https://www.hashicorp.com/blog/a-guide-to-cloud-cost-optimization-with-hashicorp-terraform#aws-global-enforce-mandatory-tags)



Sentinel has three [enforcement levels](https://docs.hashicorp.com/sentinel/concepts/enforcement-levels/): Advisory, Soft-Mandatory, and Hard-Mandatory — please refer to the provided link for definitions. The enforcement level will dictate workflow and resolution of policy violations.

If you’re using Terraform Cloud for Business or Terraform Enterprise, users may interact with the Terraform UI, CLI, or the API to fully integrate into their CD/CD pipelines for policy workflow control and into [VCS](https://www.terraform.io/docs/providers/type/vcs-index.html) systems such as GitLab, GitHub, and BitBucket for policy creation and management.

#### Sentinel Cost Compliance Code Examples

In the [aws-global/limit-cost-by-workspace-type](https://github.com/hashicorp/terraform-guides/blob/master/governance/second-generation/cloud-agnostic/limit-cost-by-workspace-type.sentinel) policy defined for this workspace (which can be individual or globally defined) I’ve applied monthly spending limits and an enforcement level. You can see this snippet below which shows cost limits ($200 for development, $500 for QA, and so on). I’ve set the enforcement level to soft mandatory, which means administrators can override policy failures if there is a legitimate reason, but it will block most users from spending up to that amount.

**Sentinel Cost Compliance — Monthly Limits**

##### Monthly Limits #####

limits = {

"dev": decimal.new(200),

"qa": decimal.new(500),

"prod": decimal.new(1000),

"other": decimal.new(50),

}

policy "limit-cost-by-workspace-type" {

enforcement\_level ="soft-mandatory"

Now we have a mechanism to control costs \_before\_ those resources are provisioned.

**Sentinel Cost Compliance — Instance Types**

Here’s another example — for a multitude of reasons including compliance and costs, many customers will restrict what compute instance types can be provisioned and potentially configuration limits based on environment or team. A full example of this type of policy can be seen here: [aws-compute-nonprod/restrict-ec2-instance-type](https://github.com/hashicorp/terraform-guides/blob/master/governance/second-generation/aws/restrict-ec2-instance-type.sentinel).

In the example below, we have a policy that controls instance sizes on non-prod environments to ensure lower costs in these less critical environments.

# Allowed EC2 Instance Types

# We don't include t2.medium or t2.large as they are not allowed in dev or test environments

allowed\_types = [

"t2.nano",

"t2.micro",

"t2.small",

]

policy "restrict-ec2-instance-type" {

enforcement\_level = "advisory"

**Sentinel Cost Compliance — Enforce Tagging**

Lastly, tagging is a critical factor in understanding costs. Tagging enables you to group, analyze, and create more granular policy around infrastructure instances. (See how tagging [helps our own teams at HashiCorp](https://www.hashicorp.com/resources/lower-aws-cloud-costs-terraform-lambda/) cull orphaned cloud instances)

Sentinel can enforce tagging at the provisioning phase and during updates to ensure that optimization can be targeted and governed. Tagging is managed in a simple key-value format and can be enforced across all CSPs. Here is a full [sample policy for enforcement on AWS](https://github.com/hashicorp/terraform-guides/blob/master/governance/second-generation/aws/enforce-mandatory-tags.sentinel).

### List of mandatory tags ###

mandatory\_tags = [

"Name",

"ttl",

"owner"

"cost center

"appid",

]

policy "enforce-mandatory-tags" {

enforcement\_level ="hard-mandatory"

This gives you the ability to ensure things like owner, cost center, and time-to-live for each infrastructure resource are trackable.

## The Way Forward for Cloud Cost Management

As organizations increasingly use cloud infrastructure, the DevOps philosophy can no longer be ignored. As silos between developers and operators break, so must the silos between finance and engineering.

Engineers have a lot more autonomy to deploy the infrastructure they need immediately. That means more responsibility to control those costs themselves. Technologies like Terraform and Sentinel give engineering the automated, finance-monitored workflows they need to manage costs and reclaim unused resources — all inside the tooling that most of them already use. This helps organizations avoid a cumbersome, ticket-based approach, while also avoiding the chaos and waste of Shadow IT run amok.