Sourcegraph

Cloud cost optimization

Part 1: Create visibility

There is a growing trend for organizations of all sizes to get out of the business of running their own data centers and move their infrastructure and application workloads to the cloud. This typically includes a transition to software as a service (SaaS) as well as leveraging infrastructure as a service (IaaS). IaaS providers are often referenced as "Hyperscalers" and include Google Cloud Platform (GCP), Amazon AWS, and Microsoft Azure. The advantages of and motivations for adopting cloud applications and infrastructure are often to achieve cost savings through a shift from a CapEx to an OpEx financial model and the opportunity to only pay for the resources and infrastructure you need or use.

The problem

However, most organizations are finding that there is actually a higher cost than expected for running their legacy workloads in a hyperscale environment. In fact, Gartner estimates that as much as 70% of cloud costs are wasted. As a result, there has been a need to adopt best practices for optimizing cloud costs (examples: What is cloud cost optimization? 15+ best practices for 2022, Cloud Cost optimization: definition and strategies, 9 Best practices for cloud cost optimization). While these articles are helpful if you need to drive alignment around an initiative, what they miss are practical steps you can take today to start optimizing your cloud costs.

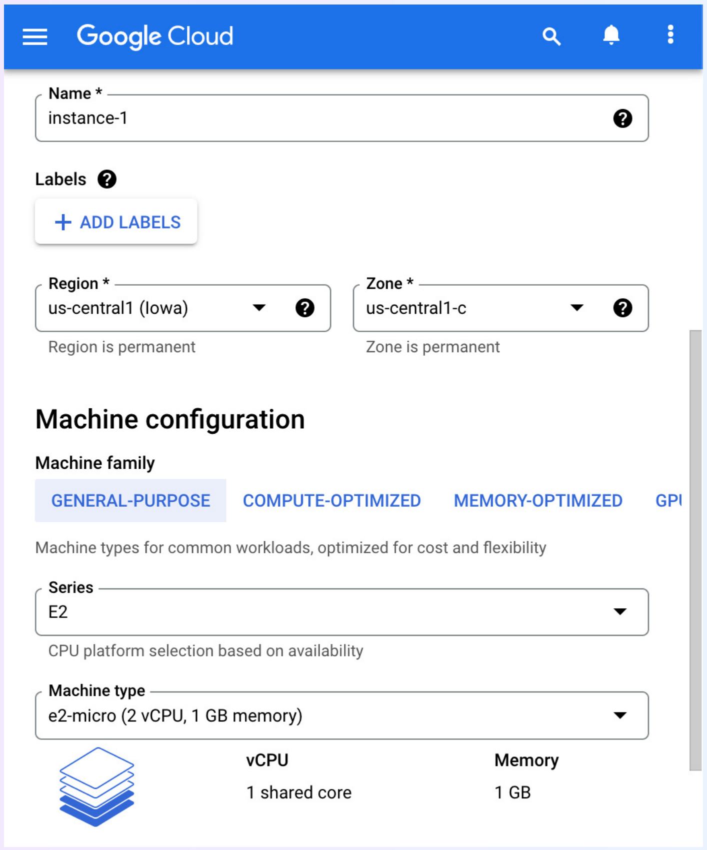
Whether you are starting your journey to the cloud, or just looking to better optimize your current cloud costs, it helps to break the problem down into two different buckets.

1. **Infrastructure configuration, optimization, and automation** – Create awareness and visibility into cloud infrastructure usage using infrastructure as code (covered here in part 1). Then review how your infrastructure can be automated and optimized to find cost savings (covered in part 2 – coming soon).

2. **Application refactoring** – Review application architecture and resource utilization. Often applications are monolithic and not designed to be cloud native. Hyperscalers provide compute, memory, and storage, as well as new standardized services, technologies, and approaches that were not available in the traditional data center environments. This requires insights into the portions of your code that can leverage the newly available technologies and the ability to rapidly update patterns in code across the organization. At the very least, organizations need to identify and optimize their workloads to run more efficiently in the new hyperscale environment (covered in parts 3 and 4 – coming soon).

Infrastructure as code

Infrastructure as code (IaC) is the process of managing and provisioning cloud infrastructure through machine-readable definition files like Terraform (open source repositories with examples, example files) and AWS CloudFormation (open source repositories with examples, example files), rather than physical hardware configuration or interactive configuration tools1. These definition files make it possible to have visibility of all your provisioned infrastructure, especially if you have a universal code search solution inside your organization.



Name *
instance-1

Labels

Region *
us-central1 (Iowa)
Region is permanent

Zone *
us-central1-c
Zone is permanent

Machine configuration

Machine family

Machine types for common workloads, optimized for cost and flexibility

Series
E2
CPU platform selection based on availability

Machine type
e2-micro (2 vCPU, 1 GB memory)

vCPU	Memory
1 shared core	1 GB

```
provider "google" {
  project = "{{YOUR GCP PROJECT}}"
  region  = "us-central1"
  zone    = "us-central1-c"
}

resource "google_compute_instance" "vm_instance" {
  name         = "terraform-instance"
  machine_type = "e2-micro"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-9"
    }
  }

  network_interface {
    # A default network is created for all GCP projects
    network = google_compute_network.vpc_network.self_link
    access_config {
    }
  }
}

resource "google_compute_network" "vpc_network" {
  name                    = "terraform-network"
  auto_create_subnetworks = "true"
}
```

According to Gartner research, less than 5% of server provisioning utilized IaC in 2020, and only 40% is expected to do so by 2023. This means that the vast majority of cloud infrastructure is manually provisioned, built on a huge amount of untraceable scripts, or manually configured in the cloud provider interface.

If your organization has not adopted IaC (which is the majority – you're not alone!), then the first step is to create visibility into your cloud infrastructure configuration by exporting it into a version control system so that you can search over the current state and see the history of changes.

Create visibility

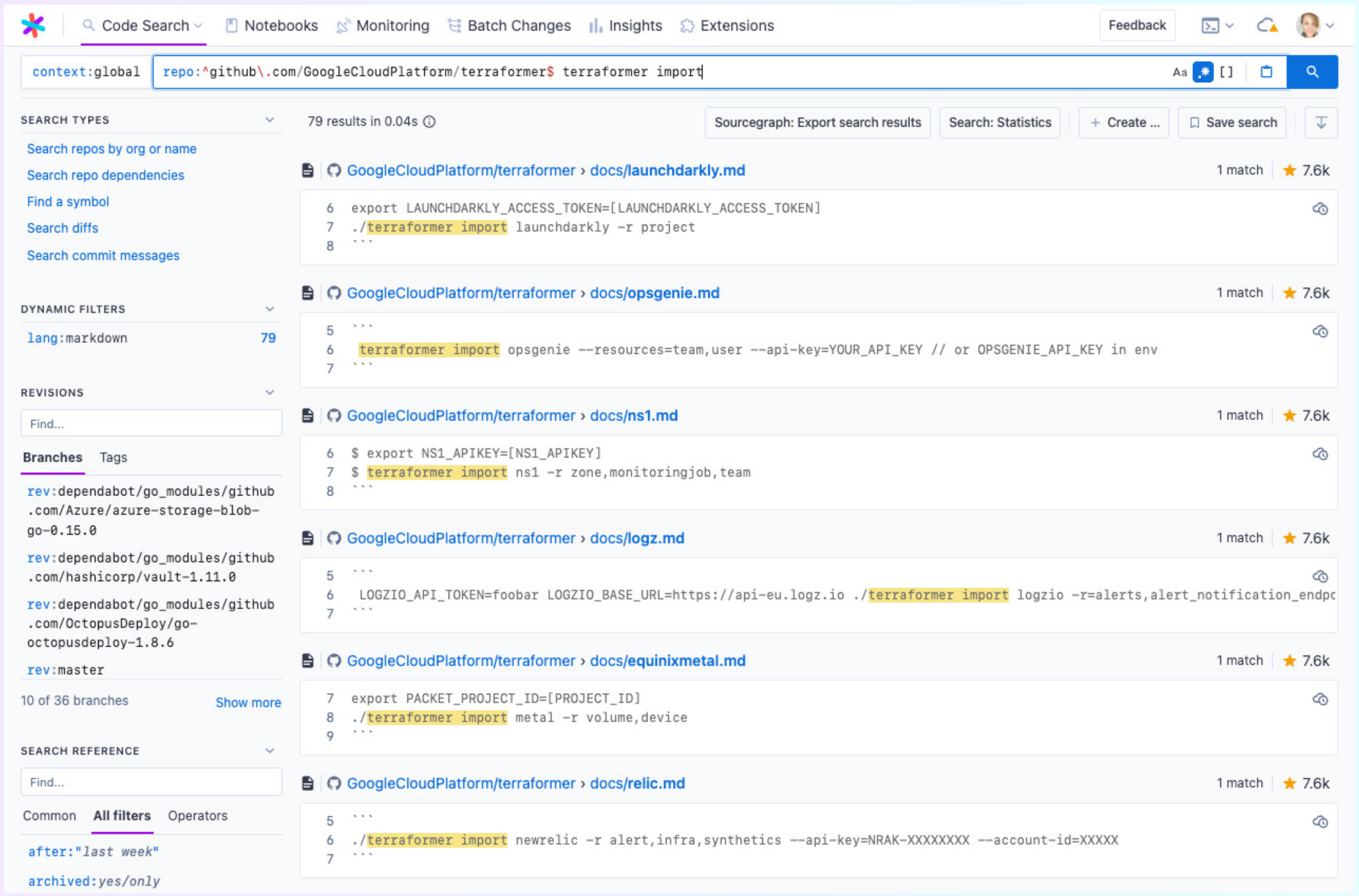
Doing a one-time audit of your cloud infrastructure is beneficial for cutting costs, but if you want to continually review for cost reduction, you'll want to have automation in place to regularly export your config into a Git repository. We've pulled together some helpful resources that should make this setup a bit easier.

1. Export all cluster config into a Git repository – use a tool like CloudFormation or Terraform (provisioning tools).

    a. First, choose the provisioning tool you want to export to. We recommend Terraform. Here is a helpful blog series on choosing Terraform over other provisioning tools.

    b. Terraformer (https://github.com/GoogleCloudPlatform/terraformer) is a CLI to export config to Terraform on any cloud provider.

Example export commands for all cloud providers from Terraformer docs



```
context:global  repo:^github\.com/GoogleCloudPlatform/terraformer$ terraformer import
```

79 results in 0.04s

lang:markdown 79

rev:dependabot/go_modules/github.com/Azure/azure-storage-blob-go-0.15.0
rev:dependabot/go_modules/github.com/hashicorp/vault-1.11.0
rev:dependabot/go_modules/github.com/OctopusDeploy/go-octopusdeploy-1.8.6
rev:master

10 of 36 branches

after:"last week"
archived:yes/only

GoogleCloudPlatform/terraformer › docs/launchdarkly.md — 1 match ★ 7.6k

```
export LAUNCHDARKLY_ACCESS_TOKEN=[LAUNCHDARKLY_ACCESS_TOKEN]
./terraformer import launchdarkly -r project
```
```

GoogleCloudPlatform/terraformer › docs/opsgenie.md — 1 match ★ 7.6k

```
```
 terraformer import opsgenie --resources=team,user --api-key=YOUR_API_KEY // or OPSGENIE_API_KEY in env
```
```

GoogleCloudPlatform/terraformer › docs/ns1.md — 1 match ★ 7.6k

```
$ export NS1_APIKEY=[NS1_APIKEY]
$ terraformer import ns1 -r zone,monitoringjob,team
```
```

GoogleCloudPlatform/terraformer › docs/logz.md — 1 match ★ 7.6k

```
```
 LOGZIO_API_TOKEN=foobar LOGZIO_BASE_URL=https://api-eu.logz.io ./terraformer import logzio -r=alerts,alert_notification_endpc
```
```

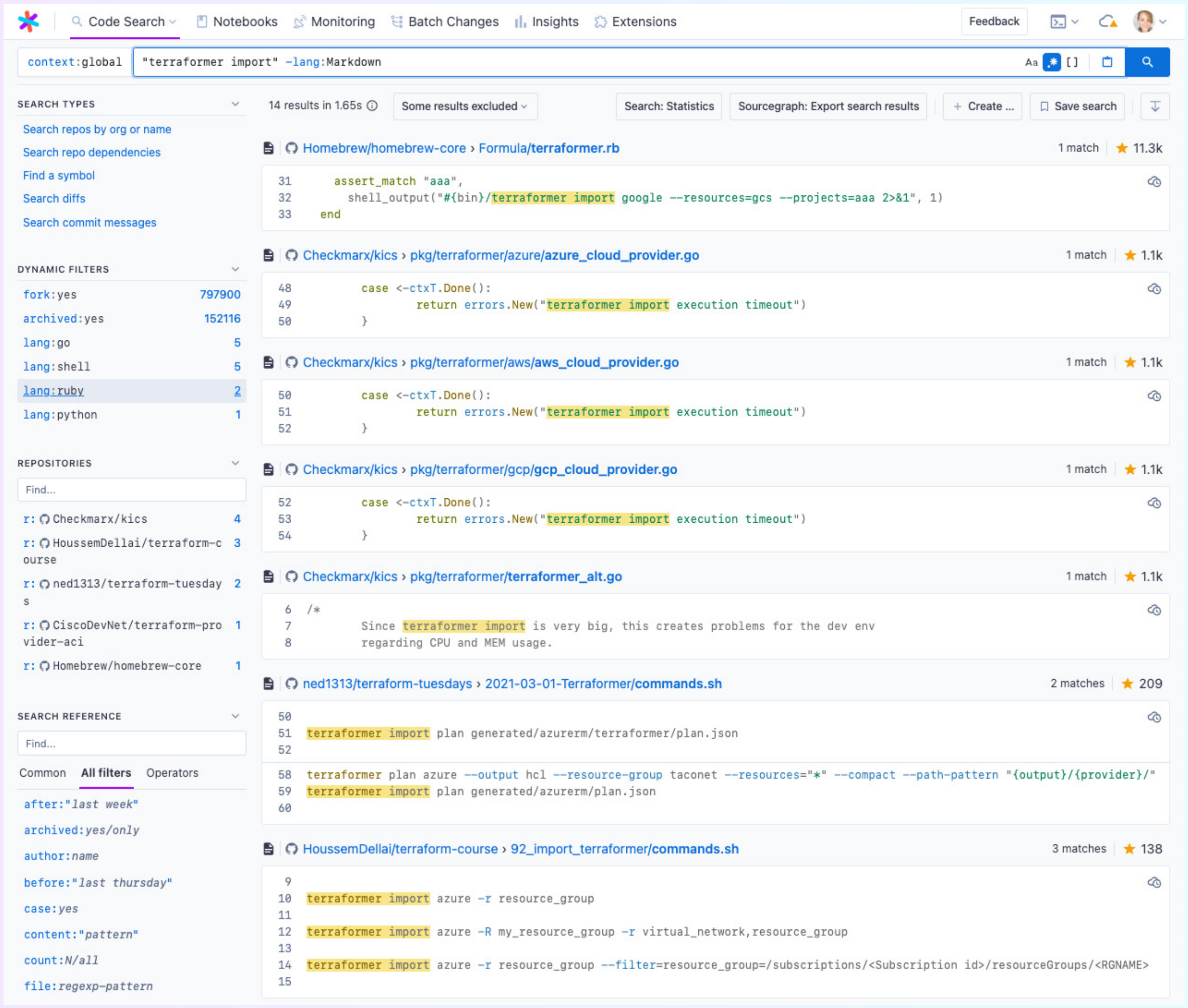
GoogleCloudPlatform/terraformer › docs/equinixmetal.md — 1 match ★ 7.6k

```
export PACKET_PROJECT_ID=[PROJECT_ID]
./terraformer import metal -r volume,device
```
```

GoogleCloudPlatform/terraformer › docs/relic.md — 1 match ★ 7.6k

```
```
./terraformer import newrelic -r alert,infra,synthetics --api-key=NRAK-XXXXXXXX --account-id=XXXXX
```
```
```

Examples of scripts that reference `terraformer import`



```
context:global  "terraformer import" -lang:Markdown
```

SEARCH TYPES
- Search repos by org or name
- Search repo dependencies
- Find a symbol
- Search diffs
- Search commit messages

DYNAMIC FILTERS

fork:yes	797900
archived:yes	152116
lang:go	5
lang:shell	5
lang:ruby	2
lang:python	1

REPOSITORIES

r: Checkmarx/kics	4
r: HoussemDellai/terraform-course	3
r: ned1313/terraform-tuesdays	2
r: CiscoDevNet/terraform-provider-aci	1
r: Homebrew/homebrew-core	1

SEARCH REFERENCE

Common | All filters | Operators

```
after:"last week"
archived:yes/only
author:name
before:"last thursday"
case:yes
content:"pattern"
count:N/all
file:regexp-pattern
```

14 results in 1.65s · Some results excluded · Search: Statistics · Sourcegraph: Export search results · Create ... · Save search

Homebrew/homebrew-core › Formula/terraformer.rb — 1 match ★ 11.3k

```
    assert_match "aaa",
      shell_output("#{bin}/terraformer import google --resources=gcs --projects=aaa 2>&1", 1)
  end
```

Checkmarx/kics › pkg/terraformer/azure/azure_cloud_provider.go — 1 match ★ 1.1k

```
	case <-ctxT.Done():
		return errors.New("terraformer import execution timeout")
	}
```

Checkmarx/kics › pkg/terraformer/aws/aws_cloud_provider.go — 1 match ★ 1.1k

```
	case <-ctxT.Done():
		return errors.New("terraformer import execution timeout")
	}
```

Checkmarx/kics › pkg/terraformer/gcp/gcp_cloud_provider.go — 1 match ★ 1.1k

```
	case <-ctxT.Done():
		return errors.New("terraformer import execution timeout")
	}
```

Checkmarx/kics › pkg/terraformer/terraformer_alt.go — 1 match ★ 1.1k

```
/*
	Since terraformer import is very big, this creates problems for the dev env
	regarding CPU and MEM usage.
```

ned1313/terraform-tuesdays › 2021-03-01-Terraformer/commands.sh — 2 matches ★ 209

```

terraformer import plan generated/azurerm/terraformer/plan.json


terraformer plan azure --output hcl --resource-group taconet --resources="*" --compact --path-pattern "{output}/{provider}/"
terraformer import plan generated/azurerm/plan.json

```

HoussemDellai/terraform-course › 92_import_terraformer/commands.sh — 3 matches ★ 138

```

terraformer import azure -r resource_group

terraformer import azure -R my_resource_group -r virtual_network,resource_group

terraformer import azure -r resource_group --filter=resource_group=/subscriptions/<Subscription id>/resourceGroups/<RGNAME>

```

2. Export all services config into a Git repository – use a tool like Chef, Puppet, Ansible, or Kubernetes (configuration management tools) to export the services configuration.

    a. We deploy Sourcegraph.com with Kubernetes, and use kube-backup (https://github.com/pieterlange/kube-backup) to export and backup this configuration.

3. Set up a CronJob to regularly snapshot the configuration files from the first two steps into the Git repos. By regularly updating, you will be able to have better traceability of changes over time, and it will allow you to better understand the entire system. Additionally, you have now moved one small step closer to infrastructure as code.

Visibility and awareness are the first steps toward greater cost savings across your organization and extend to include both direct costs of cloud infrastructure as well as operational savings. The next step is to identify targets for optimization and to monitor changes over time. We will cover approaches and practical examples of this in part 2.

Sourcegraph's code intelligence platform is more than simply search. The platform drives velocity by helping development teams quickly get unblocked, save time resolving issues, and gain insights to make better decisions. Request a demo to learn more about ways we can help accelerate your development team.

Request a demo