Google Cloud

Optimizing high-speed market analysis in the cloud with infrastructure as code

Improving Google Cloud's
STAC-M3™ benchmark results by up to 18x

Executive summary

As the financial markets move faster and remain volatile, quantitative analysts grapple with ever-growing data sets and the complex infrastructure needed to generate alpha and manage risk. With the need to process more data and act on opportunities faster, increasing compute speed, performance, and scalability in the cloud is crucial to gaining a competitive advantage.

The Securities Technology Analysis Center (STAC®), an organization that improves technology discovery and assessment in the finance industry through dialog and research, recently audited the STAC-M3™ benchmark suite on Google Cloud (SUT ID KDB211210). These enterprise tick-analytics benchmarks assess the ability of a solution stack such as database software, servers, and storage, to perform a variety of I/O-intensive and compute-intensive operations on historical market data.

The latest STAC-M3™ tick history analytics benchmark results demonstrate Google Cloud's high-performance computing capabilities. Access to the latest technologies in Google Cloud can help hedge funds, investment banks, and other key players in the asset management industry achieve four crucial objectives:

1. Accelerate the speed of running a growing ledger of analysis while producing faster results.
2. Maximize high-performance compute clusters and storage with end-to-end automation and infrastructure as code (IaC) techniques.
3. Increase the flexibility and capacity to scale infrastructures to match market opportunities.
4. Simplify operations and reduce the cost of running calculations on massive financial data sets.

In this white paper, we highlight our methodology for achieving significant performance improvements, explain the use and advantages of IaC for infrastructure reproducibility, and provide code samples illustrating how an IaC deployment works.

Optimizing Google Cloud for high-speed market tick history analysis

Google Compute Engine processors have advanced since our last STAC-M3™ audit in 2018. The first way that we improved performance in the latest audit was by upgrading from the N1 machine type to the newer N2 machine type. Intel Cascade Lake powered the benchmarked cluster, increasing computational performance over the Skylake architecture used in the previous audit. A newer Intel Ice Lake is now available, offering over 30% better price-performance compared to Cascade Lake.

N2 machine types are general-purpose virtual machines (VMs), offering an excellent balance of performance and functionality. Compute-optimized C2 and C2D machine types were another option we considered, as they are well suited for computationally-intensive workloads, offering superior performance, higher frequencies, and control over non-uniform memory access (NUMA) settings. In this case, however, more significant performance gains came from rethinking how to handle disk input/output (I/O). C2 and C2D machine types can have up to 8 high-speed local NVMe SSDs per instance, but general-purpose N2 machine types can have up to 24. As shown below, choosing I/O capacity over processor performance was crucial to improving overall performance on the benchmark.

The cluster used 12 nodes, which was the sweet spot for performance given the way data was distributed. Each node had 32 vCPUs, 160GiB of memory, and 9 TiB of local NVMe solid state disks (SSDs).

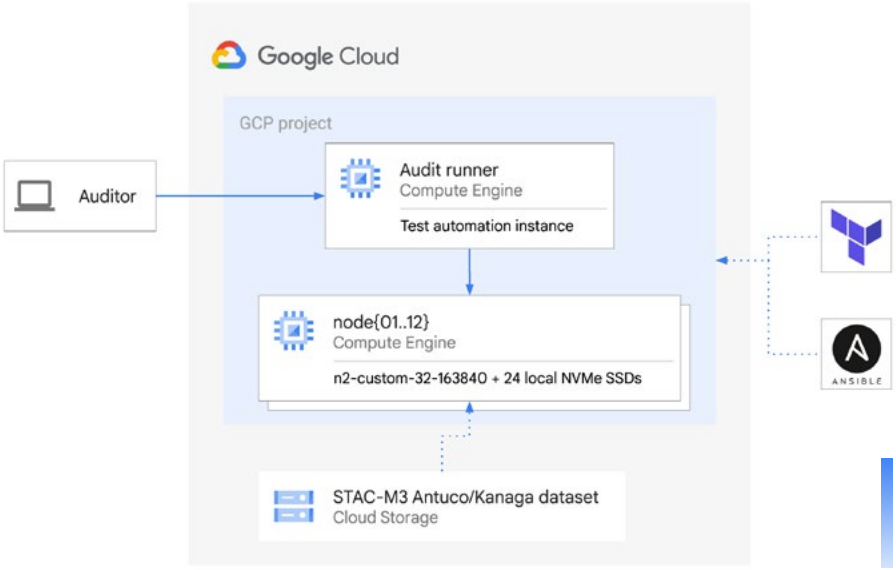


Figure 1. Sharded kdb+ 4.0 STAC-M3™ architecture on Google Cloud.

Changing our approach to disk I/O was made possible by changes in how queries were issued across the cluster. Some workloads benefit from vertical scaling, keeping computations on a single large node with nanosecond-scale latency between processor cores. Other workloads benefit from scaling out horizontally across a greater number of smaller nodes that communicate over a high-speed network.

Workloads designed for horizontal scaling are able to easily take advantage of the elasticity of cloud resources. Freed from the constraints of fixed-size, on-premises clusters, hundreds of thousands of cores can be provisioned in hours rather than months to accommodate demanding workloads and vast data sets.

The STAC-M3™ benchmark workloads on kdb+ 4.0 benefit to a certain extent from faster processors, but they are biased towards I/O performance and gain more from improvements to throughput and latency at the storage layer. By using more nodes with fewer cores each, both total storage capacity and throughput increase across the cluster in aggregate. The same concept applies to workloads that are limited by network throughput.

Working with KX, a Google Cloud Global Technology Partner, we took advantage of a new option in their STAC-M3™ implementation to "shard" tick history data across many nodes in a compute cluster. Sharding distributes unique segments of the complete data set to individual nodes, making each responsible for a range of the full data set. The downside of sharding is that it generally requires querying applications to be aware of the data layout scheme. Since the benchmark implementation's code supported this style of execution already, adapting our infrastructure was simple.

Sharding the data set removes the need to have a complete shared data set visible to all nodes and allows us to redesign the cluster to take advantage of low-latency, high-throughput local NVMe SSD storage on Google Cloud. This is exceptionally fast, ephemeral storage that is tied to the lifecycle of the node. It's excellent for high-speed local computations, but when the node is shut down, the data is lost as well. That meant we needed a persistent store from which to download the tick history data set, as generating it on demand takes several hours.

Fortunately, Google Cloud Storage provides secure, reliable, and cost-effective object storage of vast amounts of data. Most importantly, accessing it can be done in a highly parallel manner. The tick history data set was divided into hundreds of files, each individually downloadable by the node responsible for that segment of the sharded data set.

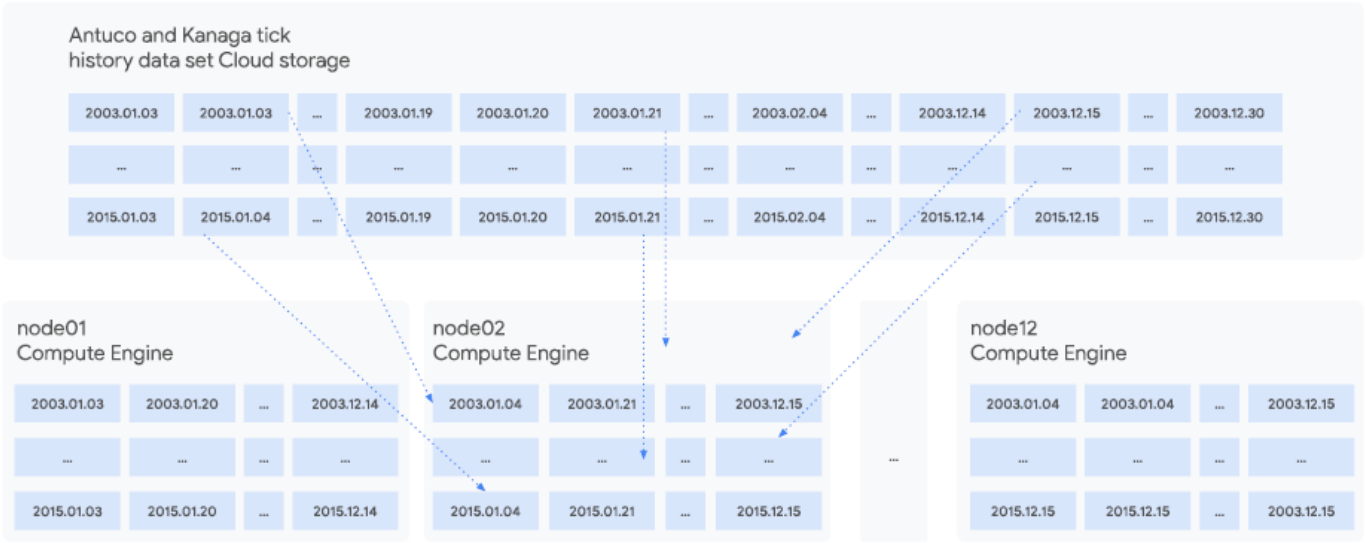


Figure 2. STAC-M3™ sharded data set.

Not only was each node responsible for downloading its own range of the entire data set, each range consisted of multiple files which could be downloaded in parallel to the local file systems of each node. This parallelism, both across and within nodes, was the key to optimizing the data transfer.

A script to calculate the necessary files in the shard ran on each node, streaming multiple files simultaneously from Cloud Storage at the 32Gbps limit of each node's network interface. While it wasn't used for this particular setup, workloads that require an even faster population of local data will benefit from our faster network tiers up to 100Gbps. Scaled out across 50 or 100 nodes, even petabyte-level data sets can be staged in a distributed NVMe caching layer within minutes.

Other options exist for the data storage architecture. In situations where high-performance workloads can't take advantage of sharded data storage, fast shared storage can still be used to present a unified view of a large data set across many nodes in a cluster. Filestore High Scale, a fully-managed Google Cloud product, allows the provision of up to 100TiB of storage per Filestore instance with 920,000/260,000 read/write IOPS and 26,000/8,800 read/write MiB/s at the maximum volume size.

Bringing workloads to Google Cloud is easier if you use NetApp or Powerscale systems on-premises; our partner solutions offer the same features and scale available in on-premises systems today. For workloads that demand the most storage performance, the Lustre parallel file system is an ideal solution. Cloud Edition for Lustre from DDN Storage, available in the Google Cloud Marketplace, offers integration with Google Cloud projects and billing.

Managing complexity with code

We also honed our approach to performance optimization. As we iterated through the cluster configuration possibilities, we used infrastructure as code (IaC) techniques to provision reproducible clusters based on definitions written in code.

IaC techniques use modern, cloud-oriented tools to interact with provisioning APIs which dynamically create, configure, and shut down networks, VMs, storage, and other resources or services based on code and configuration written by administrators. Because the tools use text-based code and configuration as inputs, everything necessary to manage infrastructure, operating systems, and applications can be stored in source code management tools like Git. This enables other activities like continuous integration and deployment of not just application code, but the underlying infrastructure itself. It also provides a full history of infrastructure state through the code repository, supporting auditing and troubleshooting activities.

Two of the most powerful and popular tools in this space are Terraform and Ansible. While there are some shared capabilities between the two tools, Terraform is *primarily* responsible for creating infrastructure resources such as networks and VMs, whereas Ansible is most commonly used to drive the operating system configuration for VMs and VM image templates to a known state in a controlled manner.

They also differ in their approaches to code structure. Terraform code is largely declarative in nature, using dependencies between resources to determine the preferred way to create and shut them down. Ansible uses declarative steps called tasks, but the playbooks that tie those tasks together are procedural. This turns out to be a good fit for their respective strengths. Cloud APIs usually follow RESTful patterns that lend themselves well to declarative management via Terraform; however, operating systems are a mix of code, configuration, user data, and persistent state that are more easily managed with a pragmatic approach of declarative steps coordinated by procedural code with Ansible.

Google Cloud collaborates with Hashicorp to maintain the Google Terraform provider and with Red Hat to maintain an extensive set of Ansible modules. There is also a large collection of Google-authored, open-source Terraform blueprints and modules that simplify many common infrastructure creation tasks.

For our latest audited STAC-M3™ stack, Terraform managed the lifecycle of the compute cluster and its storage. Ansible configured the operating system, deployed the kdb+ application, synchronized data shards from Cloud Storage, set up the benchmarking environment, and ran the tests.

The final version of the code repository represents the configuration of the cluster used to produce the results in the STAC report. This concept of infrastructure reproducibility via code is powerful for a number of reasons:

1. It dramatically simplified iteration through cluster configurations to help optimize performance. By enforcing that every deployment use IaC to reach a known configuration, comparing different benchmark runs was as easy as looking at the code for each to see which parameters changed.

2. It completely eliminated the configuration drift. Rather than manually tuning the cluster to find the optimal configuration, the IaC tools automatically drove the cluster to the correct state each time a parameter change was needed.

3. It removed the possibility of human error at deployment time. Using tools to create infrastructure based on a predefined "blueprint" removed the possibility of accidental misconfigurations.

4. It reduced time to create a given cluster to minutes from hours or days since none of the tasks were performed manually. Automated tools can push a deployment forward without human interaction, running each step immediately after the previous one without distractions or breaks.

5. It can free up administrators' time so they can manage more resources. Once the code definition of the infrastructure is created, it can be applied to multiple production and non-production environments, scaling out to clusters of thousands of nodes that would be infeasible to manage by hand.

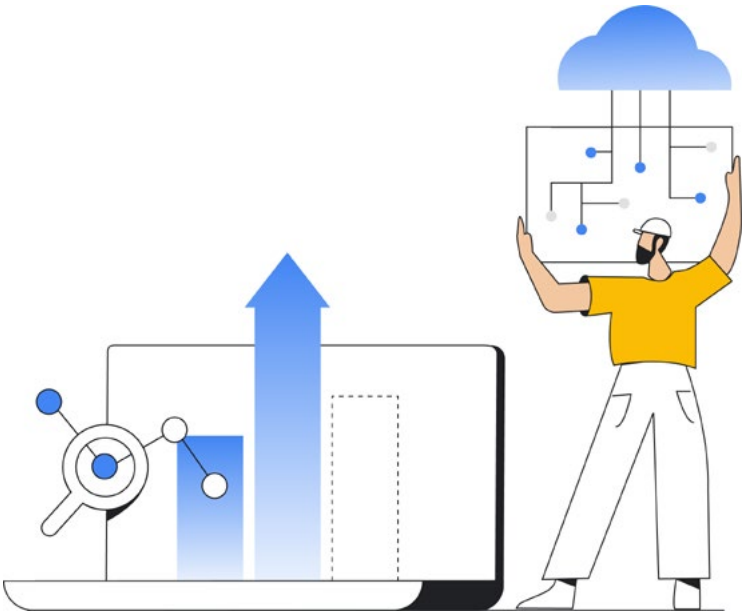
6. It can dramatically increase the auditability of an environment. In its purest form, IaC can remove the need for any routine human interaction with a cluster or Google Cloud project, delegating all resource management and configuration changes to automated processes. Deployment of infrastructure can be driven by code commits; since all the code changes are stored in the history of the code repository, a full audit trail of the current and past desired state is always available for review and analysis. Taking this further, automated code scanning tools can search repositories for insecure patterns and alert administrators or halt deployments until corrective actions are taken.

These benefits don't just apply to an ephemeral benchmarking environment. One can use the same techniques to iterate on the infrastructure, control configuration drift, reduce human error, speed up deployments, scale operations, and secure and audit all cloud assets.

Consider a research platform for quantitative analysts. In the on-premises model, high-performance computing resource managers and job schedulers are used to share scarce physical resources in a data center, boosting utilization and return on investment. The physical resources are fixed in the timeframe of days or weeks during which substantial changes to scale are infeasible. It can take months to meaningfully scale them, during which time competition for those scarce resources can slow down research teams and limit opportunities to react to market opportunities while researchers wait through long job queues.

On Google Cloud, teams can perform their research independently on dedicated clusters, each provisioned on demand using IaC tools when a hypothesis needs to be tested against market data. Those dedicated clusters experience no contention for fixed hardware, can be precisely sized for each workload to strike the right balance between cost and duration, and can be shut down or scaled down immediately upon job completion.

IaC can drive this process end to end, sizing infrastructure to workloads, matching jobs to dedicated clusters, and trimming the time that cloud resources are in use. This results in cost savings compared to huge on-premises hardware investments, and provides greater agility to react to market conditions and opportunities.



IaC for STAC-M3™ on Google Cloud

The following code samples illustrate how an IaC deployment works. Download the code bundle from the STAC report page for complete details. STAC subscribers can even replicate our results using the bundle, a kdb+ 4.0 license, and a pregenerated STAC-M3™ data set.

Starting from scratch with an empty project, a small cluster of Compute Engine instances are created with a Terraform resource definition in `automation/terraform/main.tf`:

```
resource google_compute_instance node {
  count = var.instance_count
  name = "${var.name}${format("%02s", count.index + 1)}"
  machine_type = var.machine_type
  zone = var.zone
  allow_stopping_for_update = true
  min_cpu_platform = var.cpu_platform
(...)
```

The beginning of this block builds a uniform compute cluster of variable size with node names based on the size of the cluster and using a number of other user inputs for settings like the location and type of hardware. All of the usual options for Compute Engine instances are available here, but written as text rather than accessible interactively through the Google Cloud console, such as:

```
(...)
  boot_disk {
    initialize_params {
      image = var.boot_disk_image
      size = var.boot_disk_size
      type = var.boot_disk_type
    }
  }
(...)
```

The above block, for example, initializes the boot disk to a user-specified size, type, and image. To create a cluster with this definition in your Google Cloud project, change to the `automation/terraform` directory and run:

```
terraform init
terraform apply
```

Changing any of the values in `automation/terraform/vars.tf` and running `terraform apply` will drive the cluster towards the new desired state. The cluster might need to be shut down and recreated depending on the type of change, so always review Terraform's plan carefully before accepting it.

This declarative resource management – stating your intent and using automation to drive resources towards it – is a powerful concept behind modern, cloud-native tools like Terraform. Even in cases like this one where the cluster itself is relatively simple, treating the infrastructure as committed code from the beginning enables all the benefits discussed earlier, such as rapid iteration and control over configuration drift.

One last example from the Terraform definition illustrates how resources relate to each other:

```
resource google_dns_record_set node {
  count = var.instance_count
  managed_zone = var.dns_zone
  name = "${var.name}${format("%02s", count.index + 1)}.${var.dns_domain}."
  type = "A"
  rrdatas = ["${google_compute_instance.node[count.index].network_interface[0].access_config[0].nat_ip}"]
  ttl = 5
}
```

The DNS record set described here uses two sources of inputs: User selections from variables, and dynamic information sourced from the newly-created Compute Engine instances. Terraform understands the implicit dependency between the record set and the node's address it points to. Therefore, it will create the node first, discover the address, and record that in DNS without any special handling by the user requesting the resources.

After applying the Terraform, you will have a running cluster, but each VM is created from a vanilla operating system image without any custom applications or data. The next step is to use Ansible to configure the operating system, data, and applications needed to run the STAC-M3™ benchmark:

```
- hosts: all
  gather_facts: no
  any_errors_fatal: yes
  tasks:
    - wait_for_connection:
(...)
- hosts: all
  any_errors_fatal: yes
  roles:
    - stac-m3
  tags: stac-m3
```

This block from `automation/ansible/install.yml` waits for connectivity to the cluster and maps Ansible roles (code modules) to host groups. Digging into the `stac-m3` role more, we find tasks like these in `automation/ansible/roles/stac-m3/tasks/setup.yml`:

```
(...)
- name: find mount points
  find:
    paths:
      - '{{ stac_m3_dir }}/data'
    file_type: directory
  register: mounts
```

```
# list of partitions for each worker
- name: generate par.txt
  template:
    src: par.txt.j2
    dest: '{{ path }}/db{{ item }}/par.txt'
  loop: '{{ range(stac_m3_worker_count) }}'
(...)
```

The code above locates mount points and stores them for later use, then uses a template to generate a configuration file needed for the benchmark to run. Most of the tasks are declarative, much like Terraform definitions, and playbooks should be written to be idempotent. In other words, running the same playbook again should drive the operating system, data, and applications toward a desired state, much like Terraform does with cloud infrastructure.

One final note about synchronization of sharded data as described earlier:

```
(...)
- name: sync data from GCS
  loop: '{{ stac_m3_years }}'
  script: data.py -y {{ item }} -n {{ inventory_hostname }} {{ '
'.join(groups['stac']) }}
(...)
```

This task from `automation/ansible/roles/nvme/main.yml` runs a custom script to pull data from the Cloud Storage bucket in which the full Antuco and Kanaga data set is stored. With some basic knowledge of cluster size and layout, the script can determine which data segments the local node is responsible for and cache them to the fast local NVMe storage used when running tick history calculations. As a flexible and pragmatic tool, Ansible allows arbitrary scripts to run – just make sure the execution of the script is idempotent across multiple runs of the Ansible role or playbook.

Empowering financial services with on-demand scale

Processing larger and larger data sets is becoming a major challenge - and opportunity - for the financial services industry where managing risk and generating returns are vital. Our latest benchmark results translate into real-world advantages that typically would be difficult for investment firms to achieve in complex and costly on-premise environments: Immediate answers when markets are turbulent, more thoroughly explored research theories, and reduced costs by releasing cloud resources more quickly.

To learn more or explore how Google Cloud fits into your long-term innovation roadmap, please contact your Google Cloud account manager or check our website.

Blog STAC Vault

We would like to thank author Paul Mibus and contributors Aaron Walters and Christin Brown for their help, support, and technical and domain expertise in writing this paper.