GOOGLE CLOUD PLATFORM

8 DevOps tools that smoothed our migration from AWS to GCP: Tamr

Nicholas Laferriere
Lead DevOps Engineer, Tamr

March 16, 2018

Editor's note: If you recently migrated from one cloud provider to another—or are thinking about making the move—you understand the value of avoiding vendor lock-in by using third-party tools. Tamr, a data unification provider, recently made the switch from AWS to Google Cloud Platform, bringing with them a variety of DevOps tools to help with the migration and day-to-day operations. Check out their recommendations for everything from configuration management to storage to user management.

Here at Tamr, we recently migrated from AWS to Google Cloud Platform (GCP), for a

- Scale our devops team sublinearly in relation to the number of servers and services we support
- Don't be tied into one vendor/cloud ecosystem. Flexibility matters, as we also ship our entire stack and install it on-prem at our customers sites

Our first goal was well defined and simple. We wanted all operation tasks to be fully automated. Full stop. Though we would have to build our own tooling in some cases, for the most part there's a very rich set of open source tools out there that can solve 95% of our automation problems with very little effort. And by defining everything as code, we could easily review each change and version everything in git.

Treating servers as cattle, not pets is core to the DevOps philosophy. Server "pets" have names like postgres-master, and require you to maintain them by hand. That is, you run commands on it via a shell and upgrade settings and packages yourself. Instead, we wanted to focus on primitives like the amount of cores and RAM that our services need to run. We also wanted to kill any server in the cluster at any time without having to notify anyone. This makes doing maintenance much easier and streamlined, as we would be able to do rolling restarts of every server in our fleet. It also ties into our first goal of automating everything.

We also wanted to keep our DevOps team in check. We knew from the get-go that to be successful, we would be running our platform across large fleets of servers. Doing

Find an article...

Latest stories

Products

Topics

About

RSS Feed

You can use Ansible in many different ways: as a scripting language, as a parallel ssh client, and as a traditional configuration management tool. We opted to use it as a configuration management tool and set up our code base following Ansible best practices. In addition to the best practices layed out in the documentation, we went one step further and made all of our Ansible code fully idempotent—that is, we expect to be able to run Ansible at any time, and as long as everything is already up-to-date, for it to not have to make any changes. We also try and make sure that any package upgrades in Ansible have the correct handlers to ensure a zero downtime deployment.

We were able to use our entire Ansible code base in both the AWS and GCP environments without having to change any of our actual code. The only things that we needed to change were our dynamic inventory scripts, which are just Python scripts that Ansible executes to find the machines in your environment. Ansible playbooks allow you to use multiple of these dynamic inventory scripts simultaneously, allowing us to run Ansible across both clouds at once.

That said, Ansible might not be the right fit for everyone. It can be rather slow for some things and isn't always ideal in an autoscaling environment, as it's a push-based system, not pull-based (like Puppet and Chef). Some alternatives to Ansible are the afore-mentioned Puppet and Chef, as well as Salt. They all solve the same general problem (automatic configuration of servers) but are optimized for specific use cases.

2. Infrastructure configuration: Terraform

create DNS records in Google DNS that reference a resource in AWS. This allows you to

has Heat Orchestration Templates. Terraform effectively acts as a superset of all these tools and provides a universal format across all platforms.

3. Server imaging: Packer

One of the basic building blocks of a cloud environment is a Virtual Machine (VM) image. In AWS, there's a marketplace with AMI images for just about anything, but we often needed to install tools onto our servers beyond the basic services included in the AMI. For example, think Threatstack agents that monitor the activity on the server and scan packages on the server for CVEs. As a result, it was often easier to just build our own images. We also build custom images for our customers and need to share them into their various cloud accounts. These images need to be available to different regions, as do our own base images that we use internally as the basis for our VMs. Having a consistent way to build images independent of a specific cloud provider and a region is a huge benefit.

We use Packer, in conjunction with our Ansible code base, to build all of our images. Packer provides the framework to spin up machines, runs our Ansible code, then saves a copy of the snapshot of the machine into our account. Because Packer is integrated with configuration management tools, it allowed us to define everything in the AMIs as source code. This allows us to easily version images and have confidence that we know exactly what's in our images. It made reproducing problems that customers had with our images trivial, and allowed us to easily generate changelogs for images.

Find an article...

Latest stories

Products

Topics

About

RSS Feed

Containers in and of themselves don't inherently provide scale or high availability on their own. Docker itself is just a piece of the puzzle. To fully leverage containers you need something to manage them and provide management hooks. This is where a container orchestration comes in. It allows you to link together your containers and use them to build up services in a consistent, fault-tolerant way.

For our stack we use Apache Mesos as the basis of our compute clusters. Mesos is basically a distributed kernel for scheduling tasks on servers. It acts as a broker for requests from frameworks to resources (cpu, memory, disk, gpus) available on machines in the Mesos cluster. One of the most common frameworks for Mesos is Marathon, which ships as part Mesosphere's commercial DC/OS (Data Center Operating System), the main interface for launching tasks onto a Mesos cluster. Internally we deploy all of our services and dependencies on top of a custom Mesos cluster. We spent a fair amount of time building our own deployment/packaging tool on top of Marathon for shipping releases and handling deployments. (Down the road we hope to open source this tool, in addition to writing a few blog posts about it).

The Mesos + Marathon approach for hosting services is so flexible that during our migration from AWS to GCP, we were able to span our primary cluster across both clouds. As a result, we were able to slowly switch services running on the cluster from one cloud to another using Marathon constraints. As we were switching over, we simply spun up more Compute Engine machines and then deprecated machines on the AWS side. After a couple of days, all of our services were running on Compute Engine

Find an article...

Latest stories

Products

Topics

About

RSS Feed

instance. We soon consolidated to one shared key. Then we upgraded to running an

they have a hosted LDAP endpoint that we do use for other things. The JumpCloud agent syncs with JumpCloud and provisions users and groups and ssh keys onto the server automatically for us. JumpCloud also provides a self-service portal for developers updating their ssh keys. This means that we now spend almost no time actually managing access to our servers; it's all fully automated.

It's worth noting that access to machines on Compute Engine is completely different than AWS. With GCP, users can use the gcloud command line interface (CLI) to gain access to a machine. The CLI generates a ssh key, and provisions it onto the server and creates a user account on the machine (for example, here's a sample command is `gcloud compute --project "gce-project" ssh --zone "us-east1-b" "my-machine-name"`). In addition, users can upload their ssh-keys/users pairs in the console and new machines will have those users accounts set up on launch of a machine. In other words, the problem of how to provide ssh access to developers that we ran into in on AWS doesn't exist on Compute Engine.

JumpCloud solved a specific problem with AWS, but provides a portable solution across both GCP and AWS. Using it with GCP works great, however if you're 100% on GCP, you don't need to rely on an additional external service such as JumpCloud to manage your users.

8. Storage: RexRay

Given that we run a large amount of services on top of a Mesos cluster we needed a way

Find an article...

Latest stories

Products

Topics

About

RSS Feed

workflow changes, and zero downtime. Going forward, we see being able to span across and between clouds at will as a competitive advantage, and this would not be possible without the investment we made into our tooling.

Love our list of essential DevOps tools? Hate it? Leave us a note in the comments—we'd love to hear from you. To learn more about Tamr and our data unification service, visit our website.

RELATED ARTICLE

The 2019 Accelerate State of DevOps: Elite performance, productivity, and scaling

READ ARTICLE

POSTED IN: GOOGLE CLOUD PLATFORM—DEVOPS & SRE—CUSTOMERS

RELATED ARTICLES

Find an article...

Latest stories

Products

Topics

About

RSS Feed