ArtifactDB Platform: cloud infrastructure

Deploying an ArtifactDB Platform in AWS

Sébastien Lelong / DSSC / Genentech

Contents

Introduction

ArtifactDB is a cloud-based data management solution, used to securely store arbitrary data along with searchable metadata. The ArtifactDB Platform allows to create and manage ArtifactDB instances, as a self-service. This document describes the cloud foundations on AWS required to deploy the third iteration[1] of the platform.

The main components behind that platform are:

- a Kubernetes cluster: all APIs are packaged as Helm chart and deployed within a dedicated Kubernetes cluster.
- cluster-wide controllers, such as an ingress controller, and secrets manager controller.
- a Application Load Balancer as the main entry point, forwarding traffic to that Kubernetes cluster
- an Elasticsearch, or OpenSearch cluster, usually shared across instances, handling metadata indexing.

Deployments details are explained in the next section. Implementation is based on Terraform modules. The last section shows an example of deployment in a public AWS cloud account, using Terragrunt.

Once the platform is operational, ArtifactDB instances can automatically be registered and deployed, using Olympus, the self-service component of the platform.

[1] What about the first and second iteration? There's no official description… The first iteration is a manual deployment. The second iteration brings some level of automation, while this third iteration increases reliability, is more generic, and push the automation cursor further up.

Components

This section describes the cloud components/services required to run an ArtifactDB Platform. The configuration itself is not described in details, the corresponding Terraform module holds that information, but some general information is given, as well as design details when necessary.

EKS: Kubernetes cluster

One central required service is AWS EKS, to deploy a managed Kubenetes cluster. Recent versions are supported, as the time of this writing, Kubernetes `1.25` is the current version being targetted.

Custom networking

The cluster itself is usually never exposed but rather deployed in private subnets, while an Application Load Balancer (ALB, see below) is used to proxy/forward traffic to the cluster. Because each pods running in a Kubernetes cluster will be assigned a private IP, address exhaustion is frequent, specially in the context of a corporate company where traditional private networks (class A `10.0.0.0/8`, B `172.16.0.0/12` or C `192.168.0.0/16`) must be shared across numerous private resources. To ovecome this limitation, non-routable subnets, IPv4 shared address space, are used specifically for pods. This address space provides 100.64.0.0/16 (100.64.0.0/10 officially, but AWS CIDR blocks can't be bigger than /16), which means a total of 65536 private IPs available for pods.

These non-routable subnets are not directly used during the deployment of the cluster (otherwise worker nodes could have an IP assigned in this space which would complexify routing in ingress traffic), a custom networking configuration is used instead, described here. This involves creating custom ENIConfig manifests, one for each availability zone the non-routable subnets are created in, to register and expose these subnets to the cluster, and tagging the EKS `aws-node` daemonset with a special label. This instruct the node to use a different subnet than the one it currently is deployed on.

When a worker node joins the cluster, it must also be labelled with the ENIConfig name corresponding to the right availability zone the node is deployed in. Without this label, pods assigned to the node will stay in `Pending` status, because no IP address can be assigned.

Nodegroup

This custom networking configuration impacts the way Kubernetes nodes must be created and join the cluster. To fully automate the procedure of annotating a node, a nodegroup is created based on a Launch Template. This launch template has multiple benefits.

- It allows to specify a custom EKS AMI image in place of the AWS provided ones, for instance to use a hardened image (a scenario commonly found in companies).
- Custom User-Data script can be declared. When the node starts, this custom scripts is executed. The EC2 instance self-discover the availability zone in which it runs, and annotate itself with the correct ENIConfig matching the zone. It then starts the EKS bootstrapping process and joins the cluster. Finally, if the nodegroup is declared as "ingressable", meaning it runs services that should be exposed through an ALB (as opposed to a batch job for instance, which doesn't need to exposed), it also register itself in one or more Target Groups that are connected to ALBs.

IRSA: IAM Role for Service Account

IRSA is enabled on the cluster, to allow Kubernetes Service Account to map and assume IAM roles. This setup is convenient when it comes to allow/restrict which cloud resources an ArtifactDB instance can access and modify (eg. full access to its dedicated S3 buckets), without having to use IAM users (and rotate access/secret keys frequently).

KMS: encryption at rest

A custom KMS key is created when deploying the platform. It is used for encryption-at-rest in several shared resources, such as EBS volumes attached to worker nodes, S3 buckets collecting ALB traffic logs, etc… Note this KMS key is unique per platform deployment, and is not reused for each ArtifactDB instance deployment, which has its own dedicated KMS key.

Cluster-wide controllers

Ingress controller

An ingress controller must be deployed on the cluster, to route the traffic towards to the proper namespace and service. Traefik is the preferred one, though NGINX has growing support. The ingress controller is deployed on each ingressable nodes, and is exposed through the NodePort 30080. Note the k8s services themselves, created when deploying an ArtifactDB instance, do *not* lead to the creation of a ALB, but rather declare ingress rules to Traefik (or NGNIX) to allow traffic exposition.

Secrets controller

An secret controller can be deployed to manage secrets declared in ArtifactDB instances. Sealed Secrets is currently the only option supported. Using this controller, secrets are encrypted against that controller, meaning they can only be unsealed in the cluster in which it was deployed. Sealed secrets information can safely be committed in git for instance, to facilitate instance deployment.

Secrets are also stored in AWS Secret Manager, but not directly used during the deployment. The ArtifactDB Platform uses it a the source of truth to generated sealed secrets.

ALB: Load Balancer

An ALB is used to “capture” incoming traffic and forward it to the ingress controller. A HTTPS listener can be used by declaring an SSL certificate, in which case the HTTP listener is configured to redirect traffic to HTTPS, instead of forwaring traffic.

Multiple ALB can be used, for instance to declare an Internet-facing one, as well as a private one. Each have their own Target Groups, with special tags to identify which ALB and Target Groups must be used when the “ingressable” nodes self-register themselves when booting (see above). Target Groups are configured to forward traffic to port 30080, the same port used by Traeffik.

Opensearch cluster

A shared Opensearch (or Elasticsearch cluster) is deployed within the platform. While an instance could have its own dedicated cluster, a shared instance is recommended to reduce the cost. More, indices produced by ArtifactDB instances are usually small enough to allow sharing resources, yet maintaining a good performance.

ElasticSearch 7.4 and Opensearch 1.3 are currently supported, though Opensearch >2.3 is being targetted. In the context of AWS and a shared indexing engine, Opensearch provides per-index permissions, meaning it’s possible to restrict an instance to a set of specific indices assigned during its deployement, while restricting it from accessing other instances’ indices.

Following one important principle in ArtifactDB, the whole content of an index (or more) in Opensearch can be fully restored from the metadata files found on S3. It is strictly a secondary storage.

RDS: PostgreSQL

ArtifactDB instances use PostgreSQL to provision project ID and versions in a transactional way. There's no actual data, or metadata stored in the database, only identifiers. The requirements fare pretty low, a small instance is enough, deployed in multiple AZ for production to improve uptime and reliability.

Following the same principle explained above, the whole table content can be restored from S3 itself, by listing all project IDs and their respective versions. It is also a secondary storage. be fully restored from the metadata files found on S3. It is strictly a secondary storage.

Post-deployment

Once the ArtifactDB Platform is deployed, Terraform outputs are captured and stored in a ConfigMap object in Kubernetes. This object is later used by Olympus, the self-service component of the platform, in order to discover specific parameters of the platform itself, like its version,

Implementation

The ArtifactDB Platform can be deployed using a set of Terraform modules, available at https://github.com/ArtifactDB/artifactdb-infra.

An example showing how to use configure these modules, using Terragrunt can be found at https://coming.soon.