elastisys

Setting up highly available Kubernetes clusters

www.elastisys.com

Introduction

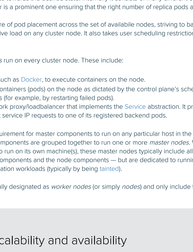
Recently, we've been working with clients on setting up highly available (HA) Kubernetes clusters. More specifically, we've been working on constructing automated installers for HA Kubernetes clusters, basically providing an automated "single-click" path to provisioning a new cluster, in a cloud or on-premise.

This whitepaper intends to share some of our experiences with Kubernetes installations and to reflect upon some of the considerations that one is faced with when planning a cluster setup. Note: this text assumes that the reader is familiar with fundamental Kubernetes concepts.

Kubernetes — master and worker (node) components

To set the stage for the following discussion, let's recapitulate the basic Kubernetes architecture and its components.

Kubernetes is really a master-slave type of architecture with certain components (*master components*) calling the shots in the cluster, and other components (*node components*) executing application workloads (containers) as decided by the master components.



The *master components* manage the state of the cluster. This includes accepting client requests (describing the desired state), scheduling containers and running control loops to drive the actual cluster state towards the desired state. These components are:

- apiserver: a REST API supporting basic CRUD operations on API objects (such as pods, deployments, and services). This is the endpoint that a cluster administrator communicates with, for example, using kubectl. The apiserver, in itself, is stateless. Instead it uses a distributed key-value storage system (etcd) as its backend for storing all cluster state.
- controller managers: runs the control/reconciliation loops that watch the desired state in the apiserver and attempt to move the actual state towards the desired state. Internally it consists of many different controllers, of which the replication controller is a prominent one ensuring that the right number of replica pods are running for each deployment.
- scheduler: takes care of *pod placement* across the set of available nodes, striving to balance resource consumption to not place excessive load on any cluster node. It also takes user scheduling restrictions into account, such as (anti-)affinity rules.

The *node components* run on every cluster node. These include:

- container runtime: such as Docker, to execute containers on the node.
- kubelet: executes containers (pods) on the node as dictated by the control plane's scheduling, and ensures the health of those pods (for example, by restarting failed pods).
- kube-proxy: a network proxy/loadbalancer that implements the Service abstraction. It programs the iptables rules on the node to redirect service IP requests to one of its registered backend pods.

There is no formal requirement for master components to run on any particular host in the cluster, however, typically these control-plane components are grouped together to run one or more *master nodes*. With the exception of etcd, which may be set up to run on its own machine(s), these master nodes typically include all components — both the control plane master components and the node components — but are dedicated to running the control plane and to not process any application workloads (typically by being tainted).

Other nodes are typically designated as *worker nodes* (or simply *nodes*) and only include the node components.

Achieving scalability and availability

A common requirement is for a Kubernetes cluster to both *scale* to accomodate increasing workloads and to be *fault-tolerant*, remaining available even in the presence of failures (datacenter outages, machine failures, network partitions).

Scalability and availability for the node plane

A cluster can be scaled by adding worker nodes, which increases the workload capacity of the cluster, giving Kubernetes more room to schedule containers.

Kubernetes is self-healing in that it keeps track of its nodes and, when a node is deemed missing (no longer passing heartbeat status messages to the master), the control plane is clever enough to re-schedule the pods from the missing node onto other (still reachable) nodes. Adding more nodes to the cluster therefore also makes the cluster more fault-tolerant, as it gives Kubernetes more freedom in rescheduling pods from failed nodes onto new nodes.

Adding nodes to a cluster is commonly carried out manually when a cluster administrator detects that the cluster is heavily loaded or cannot fit additional pods. Monitoring and managing the cluster size manually is tedious. Through the use of autoscaling, this task can be automated. The Kubernetes cluster-autoscaler is one such solution. At the time of writing, it only supports Kubernetes clusters running on GCE and AWS and only provides simplistic scaling that grows the cluster whenever a pod fails to be scheduled.

With the Elastisys autoscaling engine, more sophisticated and diverse autoscaling schemes can be implemented, utilizing predictive algorithms to stay ahead of demand and scaling not only to fit the current set of pods, but also taking into account other cluster metrics such as actual CPU/memory usage, network traffic, etc. Furthermore, the Elastisys autoscaler supports a wide range of cloud providers and can easily be extended to include more via our cloudpool abstraction. To handle node scale-downs gracefully, a kubernetes-aware cloudpool proxy is placed between the autoscaler and the chosen cloudpool.

Scalability and availability for the control plane

Adding more workers does not make the cluster resilient to all sorts of failures. For example, if the master API server goes down (for example, due to its machine failing or a network partition cutting it off from the rest of the cluster) it will no longer be possible to control the cluster (via kubectl).

For a true highly available cluster, we also need to replicate the control plane components. Such a control plane can remain reachable and functional even in the face of failures of one or a few nodes, depending on the replication factor.

A HA control plane setup requires at least three masters to withstand the loss of one master, since etcd needs to be able to form a quorum (a majority) to continue operating.

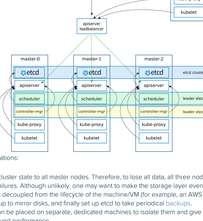
The anatomy of a HA cluster setup

Given the loosely coupled nature of Kubernetes' components, a HA cluster can be realised in many ways. But generally there are a few common guidelines:

- We need a replicated, distributed etcd storage layer.
- We need to replicate the apiserver across several machines and front them with a load-balancer.
- We need to replicate the controllers and schedulers and set them up for leader-election.
- We need to configure the (worker) nodes' kubelet and kube-proxy to access the apiserver through the load-balancer.

The replication factor to use depends on the level of availability one wishes to achieve. With three sets of master components the cluster can tolerate a failure of one master node since, in that case, etcd requires two live members to be able to form a quorum (a node majority) and continue working. This table provides the fault-tolerance of different etcd cluster sizes.

A HA cluster with a replication factor of three *could* be realized as illustrated in the following schematical image:



Some further considerations:

- etcd replicates the cluster state to all master nodes. Therefore, to lose all data, all three nodes must experience simultaneous disk failures. Although unlikely, one may want to make the storage layer even more reliable by using a separate disk that is decoupled from the lifecycle of the machine/VM (for example, an AWS EBS volume). You can also use a RAID setup to mirror disks, and finally set up etcd to take periodical backups.
- The etcd replicas can be placed on separate, dedicated machines to isolate them and give them dedicated machine resources for improved performance.
- The load-balancer must monitor the health of its apiservers and only forward traffic to live servers.
- Spread masters across data centers (availability zones in AWS lingo) to increase the overall uptime of the cluster. If the masters are placed in different zones, the cluster can tolerate the outage of an entire availability zone.
- (Worker) node kubelets and kube-proxys must access the apiserver via the load-balancer to not tie them to a particular master instance.
- The loadbalancer must not become a single point of failure. Most cloud providers can offer fault-tolerant loadbalancer services. For on-premise setups, one can make use of an active/passive nginx/HAProxy setup with a virtual/floating IP that is re-assigned by keepalived when failover is required.

Do I need a HA control plane?

So now that we know how to build a HA Kubernetes cluster, let's get started. But wait, not so fast. Before heading on to build a HA cluster you may need to ask yourself: *do I really need a HA cluster?*

A HA solution may not be strictly necessary to run a successful Kubernetes cluster. Before deciding on a HA setup, one needs to consider the requirements and needs of the cluster and the workloads one intends to run on it.

One needs to take into account that a HA solution involves more moving parts and, therefore, higher costs.

A single master can work quite well most of the time. Machine failures in most clouds are quite rare, as are availability zone outages.

Can you tolerate the master being unreachable occasionally? If you have a rather static workload that rarely changes, a few hours of downtime a year may be acceptable. Futhermore, a nice thing about the loosely coupled architecture of Kubernetes is that worker nodes remain functional, running whatever they were instructed to run, even if masters go down. Hence worker nodes, in particular if spread over different zones, can probably continue delivering the application service(s) even when the master is down.

If you do decide to run a single master, you need to ensure that the etcd data is stored reliably, preferably backed up. When we have deployed single-master Kubernetes clusters, we typically mount two disks to the master, one to hold the etcd data directory and the second disk to hold snapshots. In this way, a failed master node can be replaced (make sure you use a static IP which you can assign to a replacement node!) and the etcd disk can be attached to the replacement node. Also, on disk corruption, a backup can be restored from the backup disk.

In the end it's a balancing act where you need to trade off the need to always have a master available against the cost of keeping additional servers running.

What tools do I use?

Say that you decide that your business-level objectives requires you to run a HA setup. Then it's time to decide how to set up your cluster.

There are a bewildering number of options available, reflecting both that kubernetes can run "anywhere" (clouds, hardware architectures, OSes, etc) and that there are a lot of stakeholders that want to monetize.

One may opt for a hosted solution ("Kubernetes-as-a-Service") such as Google Container Engine or Azure Container Service. These typically reduce the administrative burden at the expense of higher costs, reduced flexibility, and vendor lock-in.

If you want to build a HA cluster yourself, there are many different options ranging from building your own from scratch (Kubernetes the hard way may lead you in the right direction), to cloud-specific installers (like kops for AWS), to installers suitable for clouds and/or bare-metal scenarios (like kubespray and kubeadm).

A lot of factors affect which solution is right for you. Using an available installer allows you to benefit from the experience of others, with the risk of ending up with a "blackboxed" solution that is difficult to adapt to your needs. At the other extreme, building from scratch will be a serious investment in time which will give you full insight and control over your solution, with the risk of repeating the same mistakes as others have already made.

Our approach

The approach we chose when we developed a cluster installer for our latest client, was to build it on the official kubeadm tool. Although kubeadm is still in beta and still does not offer HA solutions out of the box, we figured that it will soon be a stable and well-supported tool that also works well irrespective of if you are targeting a bare-metal or a cloud setup (our client wanted to support both AWS and on-premise setups).

If you would like to know more about setting up a HA cluster with kubeadm, take a look at the Creating HA clusters with kubeadm guide. Also, feel free to check out our basic installer for automating the steps of the kubeadm HA guide. It is available on github.

For anything related to Kubernetes (installation, consultancy, training, development), don't hesitate to contact us!

References

- Kubernetes Design and Architecture
- Building High-Availability Clusters

About Elastisys

Elastisys products and services extend on decades of internationally leading research in distributed systems, cloud computing, and autonomous management of virtualized resources. Elastisys was founded in 2011 and is a spin-off company from the renown distributed systems research group (www.cloudresearch.org) at Umeå University. In 2017, Elastisys was recognized by the well renowned 33-listan as one of the most promising tech startups in Sweden.

elastisys