Если вы видите что-то необычное, просто сообщите мне.

Architecting Kubernetes clusters — choosing a worker node size

Cluster capacity

In general, a Kubernetes cluster can be seen as abstracting a set of individual nodes as a big "super node".

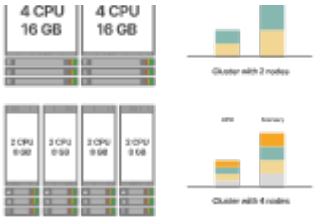
The total compute capacity (in terms of CPU and memory) of this super node is the sum of all the constituent nodes' capacities.

There are multiple ways to achieve a desired target capacity of a cluster.

For example, imagine that you need a cluster with a total capacity of 8 CPU cores and 32 GB of RAM.

```
For example, because the set of applications that you want to run on the cluster require this amount of resources.
```

Here are just two of the possible ways to design your cluster:



Both options result in a cluster with the same capacity — but the left option uses 4 smaller nodes, whereas the right one uses 2 larger nodes.

Which is better?

To approach this question, let's look at the pros and cons of the two opposing directions of "few large nodes" and "many small nodes".

```
Note that "nodes" in this article always refers to worker nodes. The choice of number and size of master nodes is an entirely different topic.
```

Few large nodes

The most extreme case in this direction would be to have a single worker node that provides the entire desired cluster capacity.

In the above example, this would be a single worker node with 16 CPU cores and 16 GB of RAM.

Let's look at the advantages such an approach could have.

1. Less management overhead

Simply said, having to manage a small number of machines is less laborious than having to manage a large number of machines.

Updates and patches can be applied more quickly, the machines can be kept in sync more easily.

Furthermore, the absolute number of expected failures is smaller with few machines than with many machines.

However, note that this applies primarily to bare metal servers and not to cloud instances.

If you use cloud instances (as part of a managed Kubernetes service or your own Kubernetes installation on cloud infrastructure) you outsource the management of the underlying machines to the cloud provider.

Thus managing, 10 nodes in the cloud is not much more work than managing a single node in the cloud.

2. Lower costs per node

While a more powerful machine is more expensive than a low-end machine, the price increase is not necessarily linear.

In other words, a single machine with 10 CPU cores and 10 GB of RAM might be cheaper than 10 machines with 1 CPU core and 1 GB of RAM.

However, note that this likely doesn't apply if you use cloud instances.

In the current pricing schemes of the major cloud providers Amazon Web Services, Google Cloud Platform, and Microsoft Azure, the instance prices increase linearly with the capacity.

For example, on Google Cloud Platform, 64 n1-standard-1 instances cost you exactly the same as a single n1-standard-64 instance — and both options provide you 64 CPU cores and 240 GB of memory.

So, in the cloud, you typically can't save any money by using larger machines.

3. Allows running resource-hungry applications

Having large nodes might be simply a requirement for the type of application that you want to run in the cluster.

For example, if you have a machine learning application that requires 8 GB of memory, you can't run it on a cluster that has only nodes with 1 GB of memory.

But you can run it on a cluster that has nodes with 10 GB of memory.

Having seen the pros, let's see what the cons are.

1. Large number of pods per node

Running the same workload on fewer nodes naturally means that more pods run on each node.

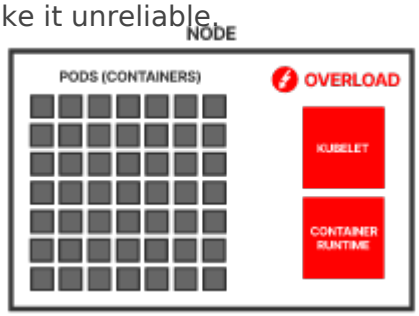
This could become an issue.

The reason is that each pod introduces some overhead on the Kubernetes agents that run on the node — such as the container runtime (e.g. Docker), the kubelet, and cAdvisor.

For example, the kubelet executes regular liveness and readiness probes against each container on the node — more containers means more work for the kubelet in each iteration.

The cAdvisor collects resource usage statistics of all containers on the node, and the kubelet regularly queries this information and exposes it on its API — again, this means more work for both the cAdvisor and the kubelet in each iteration.

If the number of pods becomes large, these things might start to slow down the system and even make it unreliable.



There are reports of nodes being reported as non-ready because the regular kubelet health checks took too long for iterating through all the containers on

the node.

For these reasons, Kubernetes recommends a maximum number of 110 pods per node.

Up to this number, Kubernetes has been tested to work reliably on common node types.

Depending on the performance of the node, you might be able to successfully run more pods per node — but it's hard to predict whether things will run smoothly or you will run into issues.

Most managed Kubernetes services even impose hard limits on the number of pods per node:

- On Amazon Elastic Kubernetes Service (EKS), the maximum number of pods per node depends on the node type and ranges from 4 to 737.
- On Google Kubernetes Engine (GKE), the limit is 100 pods per node, regardless of the type of node.
- On Azure Kubernetes Service (AKS), the default limit is 30 pods per node but it can be increased up to 250. So, if you plan to run a large number of pods per node, you should probably test beforehand if things work as expected.

2. Limited replication

A small number of nodes may limit the effective degree of replication for your applications.

For example, if you have a high-availability application consisting of 5 replicas, but you have only 2 nodes, then the effective degree of replication of the app is reduced to 2.

This is because the 5 replicas can be distributed only across 2 nodes, and if one of them fails, it may take down multiple replicas at once.

On the other hand, if you have at least 5 nodes, each replica can run on a separate node, and a failure of a single node takes down at most one replica.

Thus, if you have high-availability requirements, you might require a certain minimum number of nodes in your cluster.

3. Higher blast radius

If you have only a few nodes, then the impact of a failing node is bigger than if you have many nodes.

For example, if you have only two nodes, and one of them fails, then about half of your pods disappear.

Kubernetes can reschedule workloads of failed nodes to other nodes.

However, if you have only a few nodes, the risk is higher that there is not enough spare capacity on the remaining node to accommodate all the workloads of the failed node.

The effect is that parts of your applications will be permanently down until you bring up the failed node again.

So, if you want to reduce the impact of hardware failures, you might want to choose a larger number of nodes.

4. Large scaling increments

Kubernetes provides a Cluster Autoscaler for cloud infrastructure that allows to automatically add or remove nodes based on the current demand.

If you use large nodes, then you have a large scaling increment, which makes scaling more clunky.

For example, if you only have 2 nodes, then adding an additional node means increasing the capacity of the cluster by 50%.

This might be much more than you actually need, which means that you pay for unused resources.

So, if you plan to use cluster autoscaling, then smaller nodes allow a more fluid and cost-efficient scaling behaviour.

Having discussed the pros and cons of few large nodes, let's turn to the scenario of many small nodes.

Many small nodes

This approach consists of forming your cluster out of many small nodes instead of few large nodes.

What are the pros and cons of this approach?

The pros of using many small nodes correspond mainly to the cons of using few large nodes.

1. Reduced blast radius

If you have more nodes, you naturally have fewer pods on each node.

For example, if you have 100 pods and 10 nodes, then each node contains on average only 10 pods.

Thus, if one of the nodes fails, the impact is limited to a smaller proportion of your total workload.

Chances are that only some of your apps are affected, and potentially only a small number of replicas so that the apps as a whole stay up.

Furthermore, there are most likely enough spare resources on the remaining nodes to accommodate the workload of the failed node, so that Kubernetes can reschedule all the pods, and your apps return to a fully functional state relatively quickly.

2. Allows high replication

If you have replicated high-availability apps, and enough available nodes, the Kubernetes scheduler can assign each replica to a different node.

```
You can influence scheduler's the placement of pods with node affinites, pod affinities/anti-affinities, and taints and tolerations.
```

This means that if a node fails, there is at most one replica affected and your app stays available.

Having seen the pros of using many small nodes, what are the cons?

1. Large number of nodes

If you use smaller nodes, you naturally need more of them to achieve a given cluster capacity.

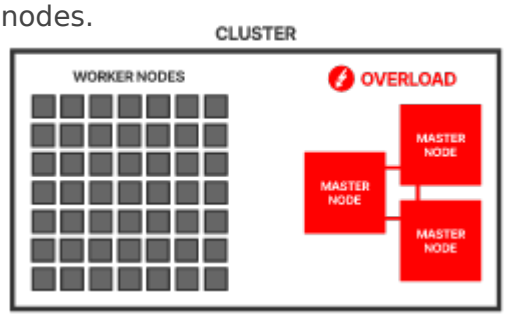
But large numbers of nodes can be a challenge for the Kubernetes control plane.

For example, every node needs to be able to communicate with every other node, which makes the number of possible communication paths grow by square of the number of nodes — all of which has to be managed by the control plane.

The node controller in the Kubernetes controller manager regularly iterates through all the nodes in the cluster to run health checks — more nodes mean thus more load for the node controller.

More nodes mean also more load on the etcd database — each kubelet and kube-proxy results in a watcher client of etcd (through the API server) that etcd must broadcast object updates to.

In general, each worker node imposes some overhead on the system components on the master nodes.



Officially, Kubernetes claims to support clusters with up to 5000 nodes.

However, in practice, 500 nodes may already pose non-trivial challenges.

The effects of large numbers of worker nodes can be alleviated by using more performant master nodes.

That's what's done in practice — here are the master node sizes used by kube-up on cloud infrastructure:

- Google Cloud Platform
  - 5 worker nodes → n1-standard-1 master nodes
  - 500 worker nodes → n1-standard-32 master nodes
- Amazon Web Services
  - 5 worker nodes → m3.medium master nodes
  - 500 worker nodes → c4.8xlarge master nodes As you can see, for 500 worker nodes, the used master nodes have 32 and 36 CPU cores and 120 GB and 60 GB of memory, respectively.

These are pretty large machines!

So, if you intend to use a large number of small nodes, there are two things you need to keep in mind:

- The more worker nodes you have, the more performant master nodes you need
- If you plan to use more than 500 nodes, you can expect to hit some performance bottlenecks that require some effort to solve
  New developments like the Virtual Kubelet allow to bypass these limitations and allow for clusters with huge numbers of worker nodes.

2. More system overhead

Kubernetes runs a set of system daemons on every worker node — these include the container runtime (e.g. Docker), kube-proxy, and the kubelet including cAdvisor.

```
cAdvisor is incorporated in the kubelet binary.
```

All of these daemons together consume a fixed amount of resources.

If you use many small nodes, then the portion of resources used by these system components is bigger.

For example, imagine that all system daemons of a single node together use 0.1 CPU cores and 0.1 GB of memory.

If you have a single node of 10 CPU cores and 10 GB of memory, then the daemons consume 1% of your cluster's capacity.

On the other hand, if you have 10 nodes of 1 CPU core and 1 GB of memory, then the daemons consume 10% of your cluster's capacity.

Thus, in the second case, 10% of your bill is for running the system, whereas in the first case, it's only 1%.

So, if you want to maximise the return on your infrastructure spendings, then you might prefer fewer nodes.

3. Lower resource utilisation

If you use smaller nodes, then you might end up with a larger number of resource fragments that are too small to be assigned to any workload and thus remain unused.

For example, assume that all your pods require 0.75 GB of memory.

If you have 10 nodes with 1 GB memory, then you can run 10 of these pods — and you end up with a chunk of 0.25 GB memory on each node that you can't use anymore.

That means, 25% of the total memory of your cluster is wasted.

On the other hand, if you use a single node with 10 GB of memory, then you can run 13 of these pods — and you end up only with a single chunk of 0.25 GB that you can't use.

In this case, you waste only 2.5% of your memory.

So, if you want to minimise resource waste, using larger nodes might provide better results.

4. Pod limits on small nodes

On some cloud infrastructure, the maximum number of pods allowed on small nodes is more restricted than you might expect.

This is the case on Amazon Elastic Kubernetes Service (EKS) where the maximum number of pods per node depends on the instance type.

For example, for a t2.medium instance, the maximum number of pods is 17, for t2.small it's 11, and for t2.micro it's 4.

These are very small numbers!

Any pods that exceed these limits, fail to be scheduled by the Kubernetes scheduler and remain in the Pending state indefinitely.

If you are not aware of these limits, this can lead to hard-to-find bugs.

Thus, if you plan to use small nodes on Amazon EKS, check the corresponding pods-per-node limits and count twice whether the nodes can accommodate all your pods.

Conclusion

So, should you use few large nodes or many small nodes in your cluster?

As always, there is no definite answer.

The type of applications that you want to deploy to the cluster may guide your decision.

For example, if your application requires 10 GB of memory, you probably shouldn't use small nodes — the nodes in your cluster should have at least 10 GB of memory.

Or if your application requires 10-fold replication for high-availability, then you probably shouldn't use just 2 nodes — your cluster should have at least 10 nodes.

For all the scenarios in-between it depends on your specific requirements.

Which of the above pros and cons are relevant for you? Which are not?

That being said, there is no rule that all your nodes must have the same size.

Nothing stops you from using a mix of different node sizes in your cluster.

The worker nodes of a Kubernetes cluster can be totally heterogeneous.

This might allow you to trade off the pros and cons of both approaches.

In the end, the proof of the pudding is in the eating — the best way to go is to experiment and find the combination that works best for you!

Revision #1
Created 2022-04-29 08:09:11 UTC by gasick
Updated 2023-04-16 19:36:18 UTC by gasick