CHAPTER 3

Getting Kubernetes

> Perplexity is the beginning of knowledge.
>
> —Kahlil Gibran

Kubernetes is the operating system of the cloud native world, providing a reliable and scalable platform for running containerized workloads. But how should you run Kubernetes? Should you host it yourself? On cloud instances? On bare-metal servers? Or should you use a managed Kubernetes service? Or a managed platform that's based on Kubernetes, but extends it with workflow tools, dashboards, and web interfaces?

That's a lot of questions for one chapter to answer, but we'll try.

It's worth noting that we won't be particularly concerned here with the technical details of operating Kubernetes itself, such as building, tuning, and troubleshooting clusters. There are many excellent resources to help you with that, of which we particularly recommend Brendan Burns' and Craig Tracey's book *Managing Kubernetes: Operating Kubernetes Clusters in the Real World* (O'Reilly).

Instead, we'll focus on helping you understand the basic architecture of a cluster, and give you the information you need to decide how to run Kubernetes. We'll outline the pros and cons of managed services, and look at some of the popular vendors.

If you want to run your own Kubernetes cluster, we list some of the best installation tools available to help you set up and manage clusters.

Cluster Architecture

You know that Kubernetes connects multiple servers into a *cluster*, but what is a cluster, and how does it work? The technical details don't matter for the purposes of this book, but you should understand the basic components of Kubernetes and

how they fit together, in order to understand what your options are when it comes to building or buying Kubernetes clusters.

The Control Plane

The cluster's brain is called the *control plane*, and it runs all the tasks required for Kubernetes to do its job: scheduling containers, managing Services, serving API requests, and so on (see Figure 3-1).

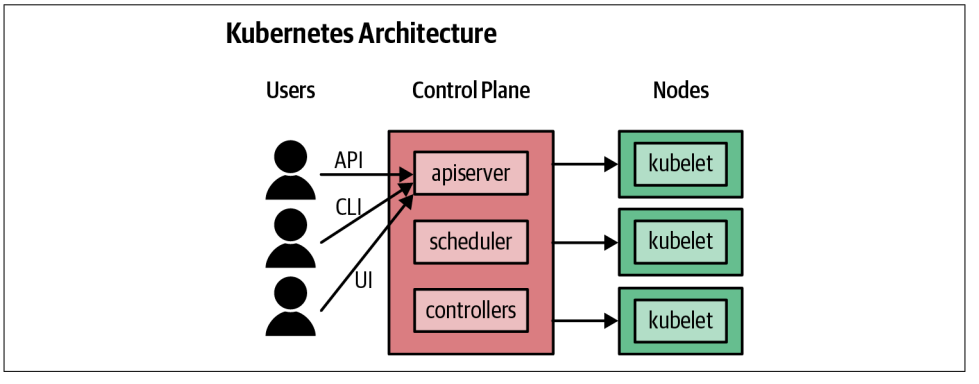

Figure 3-1. How a Kubernetes cluster works

The control plane is actually made up of several components:

`kube-apiserver`
: This is the frontend server for the control plane, handling API requests.

`etcd`
: This is the database where Kubernetes stores all its information: what nodes exist, what resources exist on the cluster, and so on.

`kube-scheduler`
: This decides where to run newly created Pods.

`kube-controller-manager`
: This is responsible for running resource controllers, such as Deployments.

`cloud-controller-manager`
: This interacts with the cloud provider (in cloud-based clusters), managing resources such as load balancers and disk volumes.

The control-plane components in a production cluster typically run on multiple servers to ensure high availability.

Node Components

Cluster members that run user workloads are called *worker nodes*.

Each worker node in a Kubernetes cluster runs these components:

`kubelet`
: This is responsible for driving the container runtime to start workloads that are scheduled on the node, and monitoring their status.

`kube-proxy`
: This does the networking magic that routes requests between Pods on different nodes, and between Pods and the internet.

Container runtime
: This actually starts and stops containers and handles their communications. Historically the most popular option has been Docker, but Kubernetes supports other container runtimes as well, such as containerd and CRI-O.

Other than running different containerized components, there's no intrinsic difference between a node running control plane components and worker nodes that run application workloads. Typically, nodes running the control plane components do not also run user-created workloads, except in very small clusters (like Docker Desktop or Minikube).

High Availability

A correctly configured Kubernetes cluster has multiple control plane nodes, making it *highly available*; that is, if any individual node fails or is shut down, or one of the control plane components on it stops running, the cluster will still work properly. A highly available control plane will also handle the situation where the control plane nodes are working properly, but some of them cannot communicate with the others, due to a network failure (known as a *network partition*).

The `etcd` database is replicated across multiple nodes, and can survive the failure of individual nodes, so long as a quorum of over half the original number of `etcd` replicas is still available.

If all of this is configured correctly, the control plane can survive a reboot or temporary failure of individual nodes.

Control plane failure

A damaged control plane doesn't necessarily mean that your applications will go down, although it might well cause strange and erratic behavior.

For example, if you were to stop all the control plane nodes in your cluster, the Pods on the worker nodes would keep on running—at least for a while. But you would be unable to deploy any new containers or change any Kubernetes resources, and controllers such as Deployments would stop working.

Therefore, high availability of the control plane is critical to a properly functioning cluster. You need to have enough control plane nodes available that the cluster can maintain a *quorum* even if one fails; for production clusters, the workable minimum is three (see "The smallest cluster" on page 98).

Worker node failure

By contrast, the failure of any single worker node shouldn't really matter as long as applications are configured to run with more than one replica. Kubernetes will detect the failure and reschedule the node's Pods somewhere else, so long as the control plane is still working.

If a large number of nodes fail at once, this might mean that the cluster no longer has enough resources to run all the workloads you need. Fortunately, this doesn't happen often, and even if it does, Kubernetes will keep as many of your Pods running as it can while you replace the missing nodes.

It's worth bearing in mind, though, that the fewer worker nodes you have, the greater the proportion of the cluster's capacity that each one represents. You should assume that a single-node failure could happen at any time, especially in the cloud, and two simultaneous failures are not unheard of.

A rare, but entirely possible, kind of failure is losing a whole cloud *availability zone*. Cloud vendors like AWS and Google Cloud provide multiple availability zones in each region, each corresponding roughly to a single datacenter. For this reason, rather than having all your worker nodes in the same zone, it's a good idea to distribute them across two or even three zones.

Trust, but verify

Although high availability should enable your cluster to survive losing some nodes, it's always wise to *actually test* this. During a scheduled maintenance window, or outside of peak hours, try rebooting a worker and see what happens. (Hopefully, nothing, or nothing that's visible to users of your applications.) Then, if you can, try rebooting a control plane node. See if you are able to continue running `kubectl` commands while the node is down.

For a more demanding test, reboot one of the control plane nodes. (Managed services such as Amazon EKS, Azure AKS, or Google Kubernetes Engine (GKE), which we'll discuss later in the chapter, don't allow you to do this). Still, a production-grade cluster should survive this with no problems whatsoever.

The Costs of Self-Hosting Kubernetes

The most important decision facing anyone who's considering running production workloads in Kubernetes is *buy or build?* Should you run your own clusters, or pay someone else to run them? Let's look at some of the options.

The most basic choice of all is self-hosted Kubernetes. By *self-hosted* we mean that you, personally, or a team in your organization, install and configure Kubernetes, on machines that you own or control, just as you might do with any other software that you use, such as Redis, PostgreSQL, or NGINX.

This is the option that gives you the maximum flexibility and control. You can decide what versions of Kubernetes to run, what options and features are enabled, when and whether to upgrade clusters, and so on. But there are some significant downsides, as we'll see in the next section.

It's More Work Than You Think

The self-hosted option also requires the maximum resources, in terms of people, skills, engineering time, maintenance, and troubleshooting. Just setting up a working Kubernetes cluster is pretty simple, but that's a long way from a cluster that's ready for production. You need to consider at least the following questions:

- Is the control plane highly available? That is, if any node goes down or becomes unresponsive, does your cluster still work? Can you still deploy or update apps? Will your running applications still be fault-tolerant without the control plane? (See "High Availability" on page 35.)
- Is your pool of worker nodes highly available? That is, if an outage should take down several worker nodes, or even a whole cloud availability zone, will your workloads stop running? Will your cluster keep working? Will it be able to automatically provision new nodes to heal itself, or will it require manual intervention?
- Is your cluster set up *securely*? Do its internal components communicate using TLS encryption and trusted certificates? Do users and applications have minimal rights and permissions for cluster operations? Are container security defaults set properly? Do nodes have unnecessary access to control plane components? Is access to the underlying `etcd` database properly controlled and authenticated?

- Are all services in your cluster secure? If they're accessible from the internet, are they properly authenticated and authorized? Is access to the cluster API strictly limited?
- Is your cluster *conformant*? Does it meet the standards for Kubernetes clusters defined by the Cloud Native Computing Foundation? (See "Conformance Checking" on page 104 for details.)
- Are your cluster nodes fully *config-managed*, rather than being set up by imperative shell scripts and then left alone? The operating system and kernel on each node needs to be updated, have security patches applied, and so on.
- Is the data in your cluster properly backed up, including any persistent storage? What is your restore process? How often do you test restores?
- Once you have a working cluster, how do you maintain it over time? How do you provision new nodes? Roll out config changes to existing nodes? Roll out Kubernetes updates? Scale in response to demand? Enforce policies?

It's Not Just About the Initial Setup

Now bear in mind that you need to pay attention to these factors not just when setting up the first cluster for the first time, but for all your clusters for all time. When you make changes or upgrades to your Kubernetes infrastructure, you need to consider the impact on high availability, security, and so on.

You'll need to have monitoring in place to make sure the cluster nodes and all the Kubernetes components are working properly. You'll also need an alerting system so that staff can be paged to deal with any problems, day or night.

Kubernetes is still in rapid development, and new features and updates are being released all the time. You'll need to keep your cluster up-to-date with those, and understand how the changes affect your existing setup. You may need to reprovision your cluster to get the full benefit of the latest Kubernetes functionality.

It's also not enough to read a few books or articles, configure the cluster the right way, and leave it at that. You need to test and verify the configuration on a regular basis—by taking down any random control plane node and making sure everything still works, for example.

Automated resilience testing tools such as Netflix's Chaos Monkey can help with this, by randomly killing nodes, Pods, or network connections every so often. Depending on the reliability of your cloud provider, you may find that Chaos Monkey is unnecessary, as regular real-world failures will also test the resilience of your cluster and the services running on it (see "Chaos Testing" on page 108).

Tools Don't Do All the Work for You

There are tools—lots and lots of tools—to help you set up and configure Kubernetes clusters, and many of them advertise themselves as being more or less point-and-click, zero-effort, instant solutions. The sad fact is that in our opinion, the large majority of these tools solve only the easy problems, and ignore the hard ones.

On the other hand, powerful, flexible, enterprise-grade commercial tools tend to be very expensive, or not even available to the public, since there's more money to be made selling a managed service than there is selling a general-purpose cluster management tool.

Kubernetes the Hard Way

Kelsey Hightower's *Kubernetes the Hard Way* tutorial is perhaps one of the best ways to get familiar with all of the underlying components of a Kubernetes cluster. It illustrates the complexity of the moving parts involved and it's an exercise worth doing for anyone considering running Kubernetes, even as a managed service, just to get a sense of how it all works under the hood.

Kubernetes Is Hard

Despite the widespread notion that it's simple to set up and manage, the truth is that *Kubernetes is hard*. Considering what it does, it's remarkably simple and well designed, but it has to deal with very complex situations, and that leads to complex software.

Make no mistake, there is a significant investment of time and energy involved in both learning how to manage your own clusters properly, and actually doing it from day to day, month to month. We don't want to discourage you from using Kubernetes, but we want you to have a clear understanding of what's involved in running Kubernetes yourself. This will help you to make an informed decision about the costs and benefits of self-hosting, as opposed to using managed services.

Administration Overhead

If your organization is large, with resources to spare for a dedicated Kubernetes cluster operations team, this may not be such a big problem. But for small to medium enterprises, or even startups with a handful of engineers, the administration overhead of running your own Kubernetes clusters may be prohibitive.

Given a limited budget and number of staff available for IT operations, what proportion of your resources do you want to spend on administering Kubernetes itself? Would those resources be better used to support your business's workloads instead? Can you operate Kubernetes more cost-effectively with your own staff, or by using a managed service?

Start with Managed Services

You might be a little surprised that, in a Kubernetes book, we recommend that you don't run Kubernetes! At least, don't run the control plane yourself. For the reasons we've outlined in the previous sections, we think that using managed services is likely to be far more cost-effective than self-hosting Kubernetes clusters. Unless you want to do something strange and experimental with Kubernetes that isn't supported by any managed provider, there are basically no good reasons to go the self-hosted route.

In our experience, and that of many of the people we interviewed for this book, a managed service is the best way to run Kubernetes, period.

If you're considering whether Kubernetes is even an option for you, using a managed service is a great way to try it out. You can get a fully working, secure, highly available, production-grade cluster in a few minutes, for a few dollars a day. (Most cloud providers even offer a free tier that lets you run a Kubernetes cluster for weeks or months without incurring any charges.) Even if you decide, after a trial period, that you'd prefer to run your own Kubernetes cluster, the managed services will show you how it should be done.

On the other hand, if you've already experimented with setting up Kubernetes yourself, you'll be delighted with how much easier managed services make the process. You probably didn't build your own house; why build your own cluster, when it's cheaper and quicker to have someone else do it, and the results are better?

In the next section, we'll outline some of the most popular managed Kubernetes services, tell you what we think of them, and recommend our favorite. If you're still not convinced, the second half of the chapter will explore Kubernetes installers you can use to build your own clusters (see "Kubernetes Installers" on page 43).

We should say at this point that neither of the authors is affiliated with any cloud provider or commercial Kubernetes vendor. Nobody's paying us to recommend their product or service. The opinions here are our own, based on personal experience and the views of hundreds of Kubernetes users we spoke to while writing this book.

Naturally, things move quickly in the Kubernetes world, and the managed services marketplace is especially competitive. Expect the features and services described here to change rapidly. The list presented here is not complete, but we've tried to include the services we feel are the best, the most widely used, or otherwise important.

Managed Kubernetes Services

Managed Kubernetes services relieve you of almost all the administration overhead of setting up and running Kubernetes clusters, particularly the control plane. Effectively, a managed service means you pay for someone else (such as Microsoft, Amazon, or Google) to run the cluster for you.

Google Kubernetes Engine (GKE)

As you'd expect from the originators of Kubernetes, Google offers a fully managed Kubernetes service that is completely integrated with the Google Cloud Platform (GCP). You can deploy clusters using the GCP web console `gcloud CLI`, or their Terraform module. Within a few minutes, your cluster will be ready to use.

Google takes care of monitoring and replacing failed nodes, and auto-applying security patches. You can set your clusters to automatically upgrade to the latest version of Kubernetes that's available, during a maintenance window of your choice.

For extended high availability, you can create *multizone* clusters, which spread worker nodes across multiple failure zones (roughly equivalent to individual datacenters). Your workloads will keep on running, even if a whole failure zone is affected by an outage.

Cluster Autoscaling

GKE also offers a cluster autoscaling option (see "Autoscaling" on page 104). With autoscaling enabled, if there are pending workloads that are waiting for a node to become available, the system will add new nodes automatically to accommodate the demand.

Conversely, if there is spare capacity, the autoscaler will consolidate Pods onto a smaller number of nodes and remove the unused nodes. Since billing for GKE is based on the number of worker nodes, this helps you control costs.

Autopilot

Google also offers a tier for GKE called *Autopilot* that takes the managed service one step further. While most hosted offerings take care of managing the control plane node, Autopilot also fully manages the worker nodes. You are billed for the CPU and memory that your Pods request, and the actual worker node VMs are abstracted away

from you. For teams that want the flexibility of Kubernetes, but do not care much about the underlying servers where the containers end up running, this would be an option worth considering.

Amazon Elastic Kubernetes Service (EKS)

Amazon has also been providing managed container cluster services for a long time, but until very recently the only option was Elastic Container Service (ECS), Amazon's proprietary technology for running containers on EC2 virtual machines. While perfectly usable, ECS is not as powerful or flexible as Kubernetes, and evidently even Amazon has decided that the future is Kubernetes, with the launch of Elastic Kubernetes Service (EKS).

Amazon has the most popular public cloud offering today, and most cloud deployments of Kubernetes are running in AWS. If you already have your infrastructure in AWS and are looking to migrate your applications to Kubernetes, then EKS is a sensible choice. Amazon takes care of managing the control plane nodes, and you deploy your containers to EC2 instance worker nodes.

If you wish to set up centralized logging with CloudWatch or cluster auto-scaling, you'll need to configure those once the cluster is up and running. This makes for less of a "batteries-included" experience than some of the other hosted offerings on the market, but depending on your environment and use-case, you may wish to customize or leave out these features anyway.

There are a few options for deploying EKS clusters including using the AWS Management Console, the `aws` CLI tool, an open source CLI tool called `eksctl`, and there is a popular EKS Terraform module. Each of these tools can automate creating the various IAM, VPC, and EC2 resources needed for a functioning Kubernetes cluster. `eksctl` can additionally handle setting up additional components, like CloudWatch logging, or installing various add-ons as part of provisioning the cluster, making it a more full-featured out-of-box Kubernetes experience.

Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) is the Microsoft option for hosted Kubernetes clusters on Azure. AKS has traditionally rolled out support for newer versions of Kubernetes before their GKE or EKS competitors. You can create clusters from the Azure web interface or using the Azure `az` command-line tool, or their Terraform AKS module.

As with GKE and EKS, you have the option to hand off managing the control plane nodes, and your billing is based on the number of worker nodes in your cluster. AKS also supports cluster autoscaling to adjust the number of worker nodes based on usage.

IBM Cloud Kubernetes Service

Naturally, the venerable IBM is not to be left out in the field of managed Kubernetes services. IBM Cloud Kubernetes Service is pretty simple and straightforward, allowing you to set up a vanilla Kubernetes cluster in IBM Cloud.

You can access and manage your IBM Cloud cluster through the default Kubernetes CLI and the provided command-line tool, or a basic GUI. There are no real killer features that differentiate IBM's offering from the other major cloud providers, but it's a logical option if you're already using IBM Cloud.

DigitalOcean Kubernetes

DigitalOcean is known for providing a simple cloud offering with excellent documentation and tutorials for developers. Recently they started offering a managed Kubernetes offering called, not surprisingly, DigitalOcean Kubernetes. Like AKS, they do not charge for running the managed control plane nodes, and you are billed for the worker nodes where your applications run.

Kubernetes Installers

If managed or turnkey clusters won't work for you, then you'll need to consider some level of Kubernetes self-hosting: that is, setting up and running Kubernetes yourself on your own machines.

It's very unlikely that you'll deploy and run Kubernetes completely from scratch, except for learning and demo purposes. The vast majority of people use one or more of the available Kubernetes installer tools or services to set up and manage their clusters.

kops

kops is a command-line tool for automated provisioning of Kubernetes clusters. It's part of the Kubernetes project, and has been around a long time as an AWS-specific tool, but is now adding alpha and beta support for other providers as well including Google Cloud, DigitalOcean, Azure, and OpenStack.

kops supports building high-availability clusters, which makes it suitable for production Kubernetes deployments. It uses declarative configuration, just like Kubernetes resources themselves, and it can not only provision the necessary cloud resources and set up a cluster, but also scale it up and down, resize nodes, perform upgrades, and do other useful admin tasks.

Like everything in the Kubernetes world, kops is under rapid development, but it's a relatively mature and sophisticated tool that is widely used. If you're planning to run self-hosted Kubernetes in AWS, kops is a good choice.

Kubespray

Kubespray (formerly known as Kargo), a project under the Kubernetes umbrella, is a tool for easily deploying production-ready clusters. It offers lots of options, including high availability, and support for multiple platforms.

Kubespray is focused on installing Kubernetes on existing machines, especially on-premise and bare-metal servers. However, it's also suitable for any cloud environment, including private cloud (virtual machines that run on your own servers). It uses Ansible Playbooks, so if you have experience using Ansible for configuration management of your servers, then this would be an option worth exploring.

kubeadm

kubeadm is part of the Kubernetes distribution, and it aims to help you install and maintain a Kubernetes cluster according to best practices. kubeadm does not provision the infrastructure for the cluster itself, so it's suitable for installing Kubernetes on bare-metal servers or cloud instances of any flavor.

Many of the other tools and services we'll mention in this chapter use kubeadm internally to handle cluster-admin operations, but there's nothing to stop you using it directly, if you want to.

Rancher Kubernetes Engine (RKE)

RKE aims to be a simple, fast Kubernetes installer. It doesn't provision the nodes for you, and you have to install Docker on the nodes yourself before you can use RKE to install the cluster. RKE supports high availability of the Kubernetes control plane.

Puppet Kubernetes Module

Puppet is a powerful, mature, and sophisticated general configuration management tool that is very widely used and has a large open source module ecosystem. The officially supported Kubernetes module installs and configures Kubernetes on existing nodes, including high availability support for both the control plane and `etcd`.

Buy or Build: Our Recommendations

This has necessarily been a quick tour of some of the options available for managing Kubernetes clusters, because the range of offerings is large and varied, and growing all the time. However, we can make a few recommendations based on commonsense principles. One of these is the philosophy of *run less software*.

Run Less Software

> There are three pillars of the Run Less Software philosophy, all of which will help you manipulate time and defeat your enemies.
>
> 1. Choose standard technology
> 2. Outsource undifferentiated heavy lifting
> 3. Create enduring competitive advantage
>
> —Rich Archbold

While using innovative new technologies is fun and exciting, it doesn't always make sense from a business point of view. Using *boring* software that everybody else is using is generally a good bet. It probably works, it's probably well supported, and you're not going to be the one taking the risks and dealing with the inevitable bugs.

If you're running containerized workloads and cloud native applications, Kubernetes is the boring choice, in the best possible way. Given that, you should opt for the most mature, stable, and widely used Kubernetes tools and services.

Undifferentiated heavy lifting is a term coined at Amazon to denote all the hard work and effort that goes into things like installing and managing software, maintaining infrastructure, and so on. There's nothing special about this work; it's the same for you as it is for every other company out there. It costs you money, instead of making you money.

The run less software philosophy says that you should outsource undifferentiated heavy lifting because it'll be cheaper in the long run, and it frees up resources you can use to work on your core business.

Use Managed Kubernetes if You Can

With the run less software principles in mind, we recommend that you outsource your Kubernetes cluster operations to a managed service. Installing, configuring, maintaining, securing, upgrading, and making your Kubernetes cluster reliable is undifferentiated heavy lifting, so it makes sense for almost all businesses not to do it themselves.

> Cloud native is the practice of accelerating your business by not running stuff that doesn't differentiate you. It's not a cloud provider, it's not Kubernetes, it's not containers, it's not a technology.
>
> —Justin Garrison

But What About Vendor Lock-in?

If you commit to a managed Kubernetes service from a particular vendor, such as Google Cloud, will that lock you in to the vendor and reduce your options in the future? Not necessarily. Kubernetes is a standard platform, so any applications and services you build to run on GKE will also work on any other certified Kubernetes provider's system with possibly some minor tweaks.

Does managed Kubernetes make you more prone to lock-in than running your own Kubernetes cluster? We think it's the other way around. Self-hosting Kubernetes involves a lot of machinery and configuration to maintain, all of which is intimately tied in to a specific cloud provider's API. Provisioning AWS virtual machines to run Kubernetes, for example, requires completely different code than the same operation on Google Cloud. Some Kubernetes setup assistants, like the ones we've mentioned in this chapter, support multiple cloud providers, but many don't.

Part of the point of Kubernetes is to abstract away the technical details of the underlying infrastructure, and present developers with a standard, familiar interface that works the same way whether it happens to be running on Azure or your own bare-metal servers.

As long as you design your applications and automation to target Kubernetes itself, rather than the cloud infrastructure directly, you're as free from vendor lock-in as you can reasonably be. Each infrastructure provider will have some differences as to how they define compute, networking, and storage, but to Kubernetes those present as minor tweaks to the Kubernetes manifests. The majority of Kubernetes deployments will work the same way, regardless of where they are running.

Bare-Metal and On-Prem

It may come as a surprise to you that being cloud native doesn't actually require being *in the cloud*, in the sense of outsourcing your infrastructure to a public cloud provider such as Azure or AWS.

Many organizations run part or all of their infrastructure on bare-metal hardware, whether colocated in datacenters or on-premises. Everything we've said in this book about Kubernetes and containers applies just as well to in-house infrastructure as it does to the cloud.

You can run Kubernetes on your own hardware machines; if your budget is limited, you can even run it on a stack of Raspberry Pis (Figure 3-2). Some businesses run a *private cloud*, consisting of virtual machines hosted by on-prem hardware.

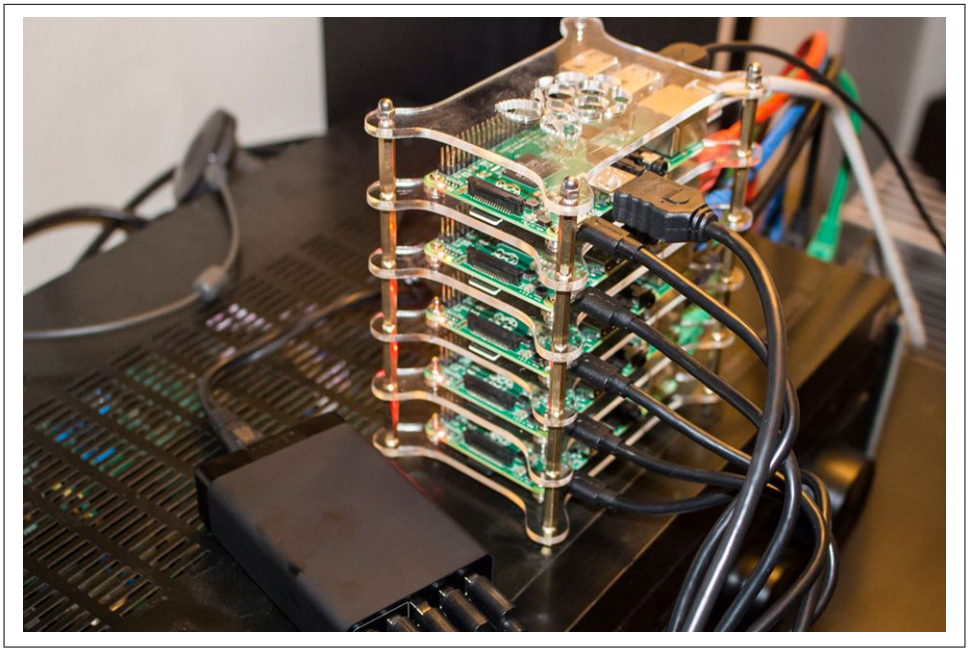

Figure 3-2. Kubernetes on a budget: a Raspberry Pi cluster (photo by David Merrick)

Multicloud Kubernetes Clusters

In some organizations, developers deploy applications to multiple types of infrastructure, including public cloud and private on-premise environments. Because Kubernetes can run anywhere, it has become a useful tool to standardize the experience between these multicloud or hybrid-cloud situations. There are tools available to make it easier to manage workloads across different Kubernetes clusters running in such environments.

VMware Tanzu

VMware has traditionally been associated with making tools for managing virtual machines and the related VM infrastructure. They now also have tooling for Kubernetes environments with a suite of tools called Tanzu. Specifically, Tanzu Mission Control allows you to centrally manage multiple Kubernetes clusters, regardless of where they are running. There are also tools for building, deploying, and monitoring the containerized workloads.

OpenShift

OpenShift is more than just a managed Kubernetes service: it's a full *Platform-as-a-Service* (PaaS) product, which aims to manage the whole software development life cycle, including continuous integration and build tools, test runner, application deployment, monitoring, and orchestration.

OpenShift can be deployed to bare-metal servers, virtual machines, private clouds, and public clouds, so you can create a single Kubernetes cluster that spans all these environments. This makes it a good choice for very large organizations, or those with very heterogeneous infrastructure.

Anthos

Similar to VMware Tanzu, Google's Anthos tooling makes it possible to centrally manage Kubernetes workloads in clusters that are running in multiple clouds, such as GKE, AWS, and on-premise Kubernetes environments. They also allow you to hook your on-premise infrastructure into the other services that Google Cloud offers, like their hosted container registry, build pipeline tooling, and networking layer.

Most small- and mid-sized teams likely do not need to start out focused on multi-cloud infrastructures. These types of environments come with additional complexity and cost. We recommend starting out with the basics and building up from there. Putting your applications into containers and deploying to a managed Kubernetes cloud offering is already going a long way toward setting yourself up for future flexibility, should you ever end up needing to run your applications in multiple clouds simultaneously.

Use Standard Kubernetes Self-Hosting Tools if You Must

If you already know that you have special requirements that mean managed Kubernetes offerings won't work for you, only then should you consider running Kubernetes yourself.

If that's the case, you should go with the most mature, powerful, and widely used tools available. We recommend starting with kubeadm, and then compare that to something like kops or Kubespray to see if they meet your requirements.

On the other hand, if you need your cluster to span multiple clouds or platforms, including bare-metal servers, and you want to keep your options open, consider looking into VMware Tanzu or Google Anthos. Since most of the administration overhead of Kubernetes is in setting up and maintaining the control plane, this is a good compromise.

Clusterless Container Services

If you really want to minimize the overhead of running container workloads, there's yet another level above fully managed Kubernetes services. These are so-called *clusterless* services, such as Azure Container Instances (ACI), AWS Fargate, and Google Cloud Run. Although there really is a cluster under the hood, you don't have access to it via tools like `kubectl`. Instead, you specify a container image to run, and a few parameters like the CPU and memory requirements of your application, and the service does the rest.

These types of managed services can be great for getting containers up and running quickly, but you may run into a limitation around something like how the networking or storage works on a particular cloud provider, and you may need to turn to something a little more powerful and flexible for your applications, like the full Kubernetes platform.

AWS Fargate

According to Amazon, "Fargate is like EC2, but instead of a virtual machine, you get a container." Unlike ECS, there's no need to provision cluster nodes yourself and then connect them to a control plane. You just define a task, which is essentially a set of instructions for how to run your container image, and launch it. Pricing is per second based on the amount of CPU and memory resources that the task consumes.

It's probably fair to say that Fargate makes sense for simple, self-contained, long-running compute tasks or batch jobs (such as data crunching) that don't require much customization or integration with other services. It's also ideal for build containers, which tend to be short-lived, and for any situation where the overhead of managing worker nodes isn't justified.

If you're already using ECS with EC2 worker nodes, switching to Fargate will relieve you of the need to provision and manage those nodes.

Azure Container Instances (ACI)

Microsoft's Azure Container Instances (ACI) service is similar to Fargate, but also offers integration with the Azure Kubernetes Service (AKS). For example, you can configure your AKS cluster to provision temporary extra Pods inside ACI to handle spikes or bursts in demand.

Similarly, you can run batch jobs in ACI in an ad hoc way, without having to keep idle nodes around when there's no work for them to do. Microsoft calls this idea *serverless containers*, but we find that terminology both confusing (*serverless* usually refers to cloud functions, or functions as a service [FaaS]) and inaccurate (there are servers; you just can't access them).

ACI is also integrated with Azure Event Grid, Microsoft's managed event routing service. Using Event Grid, ACI containers can communicate with cloud services, cloud functions, or Kubernetes applications running in AKS.

You can create, run, or pass data to ACI containers using Azure Functions. The advantage of this is that you can run any workload from a cloud function, not just those using the officially supported (*blessed*) languages, such as Python or JavaScript.

If you can containerize your workload, you can run it as a cloud function, with all the associated tooling. For example, Microsoft Power Automate allows even nonprogrammers to build up workflows graphically, connecting containers, functions, and events.

Google Cloud Run

Similar to ACI and Fargate, Google Cloud Run is another "container-as-a-service" offering that hides all of the server infrastructure from you. You simply publish a container image and tell Cloud Run to run that container for every incoming web request or received Pub/Sub message, their hosted message-bus service. By default, the launched containers timeout after 5 minutes, although you can extend this up to 60 minutes.

Summary

Kubernetes is everywhere! Our journey through the extensive landscape of Kubernetes tools, services, and products has been necessarily brief, but we hope you found it useful.

While our coverage of specific products and features is as up-to-date as we can make it, the world moves pretty fast, and we expect a lot will have changed even by the time you read this.

However, we think the basic point stands: it's not worth managing Kubernetes clusters yourself if a service provider can do it better and cheaper.

In our experience of consulting for companies migrating to Kubernetes, this is often a surprising idea, or at least not one that occurs to a lot of people. We often find that organizations have taken their first steps with self-hosted clusters, using tools like kops, and hadn't really thought about using a managed service such as GKE. It's well worth thinking about.

More things to bear in mind:

- Kubernetes clusters are made up of the *control plane*, and *worker nodes*, which run your workloads.
- Production clusters must be *highly available*, meaning that the failure of a control plane node won't lose data or affect the operation of the cluster.
- It's a long way from a simple demo cluster to one that's ready for critical production workloads. High availability, security, and node management are just some of the issues involved.
- Managing your own clusters requires a significant investment of time, effort, and expertise. Even then, you can still get it wrong.
- Managed services like GKE, EKS, AKS, and other similar offerings do all the heavy lifting for you, at much lower cost than self-hosting.
- If you have to host your own cluster, kops and Kubespray are mature and widely used tools that can provision and manage production-grade clusters on AWS and Google Cloud.
- Tools like VMware Tanzu and Google Anthos make it possible to centrally manage Kubernetes clusters running in multiple clouds and on-premise infrastructure.
- Don't self-host your cluster without sound business reasons. If you do self-host, don't underestimate the engineering time involved for the initial setup and ongoing maintenance overhead.