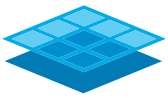
WHITE PAPER

Kubernetes Deployment Models: The Ultimate Guide



PLATFORM9

Kubernetes Overview

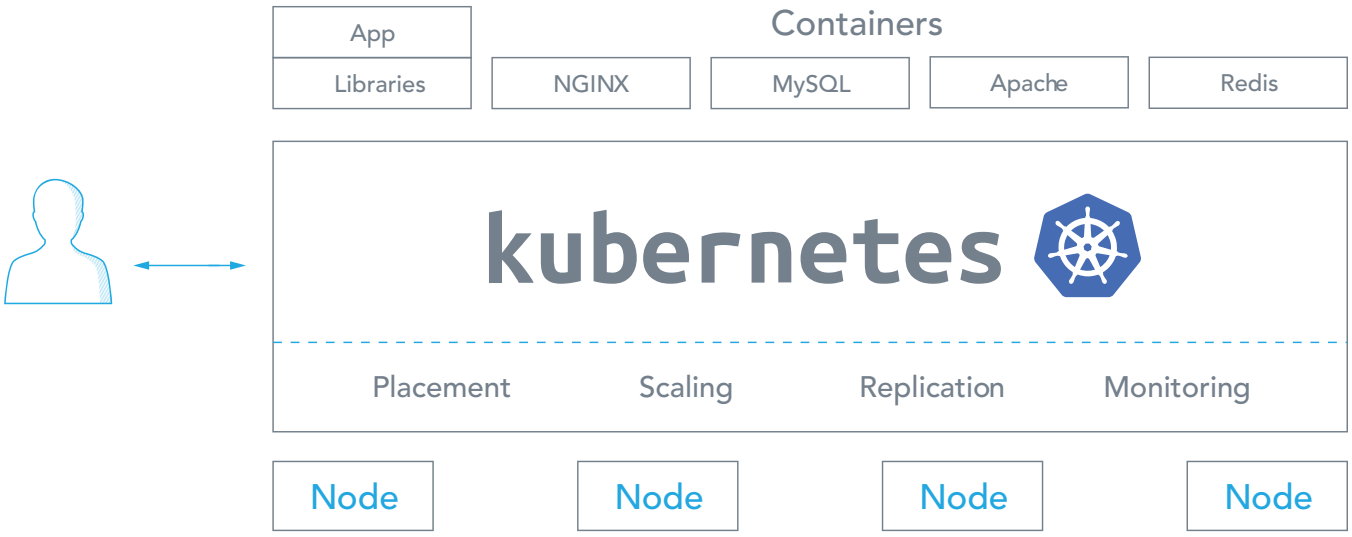
Kubernetes has become the leading platform for powering modern cloud-native micro-services in recent years. Its popularity is driven by the many benefits it provides, including:

1. Cloud-native design: Kubernetes encourages a modular, distributed architecture which increases the agility, availability, and scalability of the application.

2. Portability: Kubernetes works exactly the same way, using the same images and configuration, no matter which cloud provider or data-center environment is being used.

3. Open-source: Kubernetes is an open-source platform that developers can use without concerns of lock-in and is the most widely validated in the market today.

Once you select Kubernetes as your container orchestration platform, the next step is to select how best to deploy it. Shown below is an overview of the role Kubernetes plays in placing, scaling, replicating, and monitoring containers on nodes.



This remainder of this document will focus on deployment models for Kubernetes; for more fundamental information about why the technology matters, please see the Platform9 '**Why Kubernetes Matters**' white paper.

Kubernetes Deployment Considerations

It's easy enough to create a small test deployment of Kubernetes. You can do that by downloading upstream Kubernetes on one or more virtual machines or physical servers. However, running Kubernetes at scale with production workloads requires more thought and

effort. Here are criteria to consider while evaluating a Kubernetes solution for your enterprise workloads:

- **High Availability** - Does your Kubernetes solution deploy clusters that are highly available, with replication of the underlying metadata for recovery against failures?
- **Upgrades** - Kubernetes community delivers a major upgrade every 3-4 months. What is your Kubernetes upgrade strategy? What downtimes will upgrades require and is that acceptable for business?
- **Support for Hybrid** - Does your Kubernetes solution equally support the private data center and public cloud endpoints that your business needs to deliver Kubernetes on? Does it offer same or similar level of SLA and functionality across them?
- **Federation Support** - Does your Kubernetes solution support deployment of federated clusters that can grow across private and public clouds for robustness of infrastructure and dynamic burstability?
- **Enterprise-Ready Features** - What additional enterprise-readiness features does your Ops team need to run Kubernetes at scale and support large scale of users? Are they supported by your Kubernetes solution of choice? Some examples include, SSO support, RBAC, isolated networking, persistent storage.

Kubernetes Deployment Models

Let us now review the pros and cons of some of the best known Kubernetes deployment models including typical use cases.
A table comparing these different models is also provided at the end of this document.

All-in-one Kubernetes

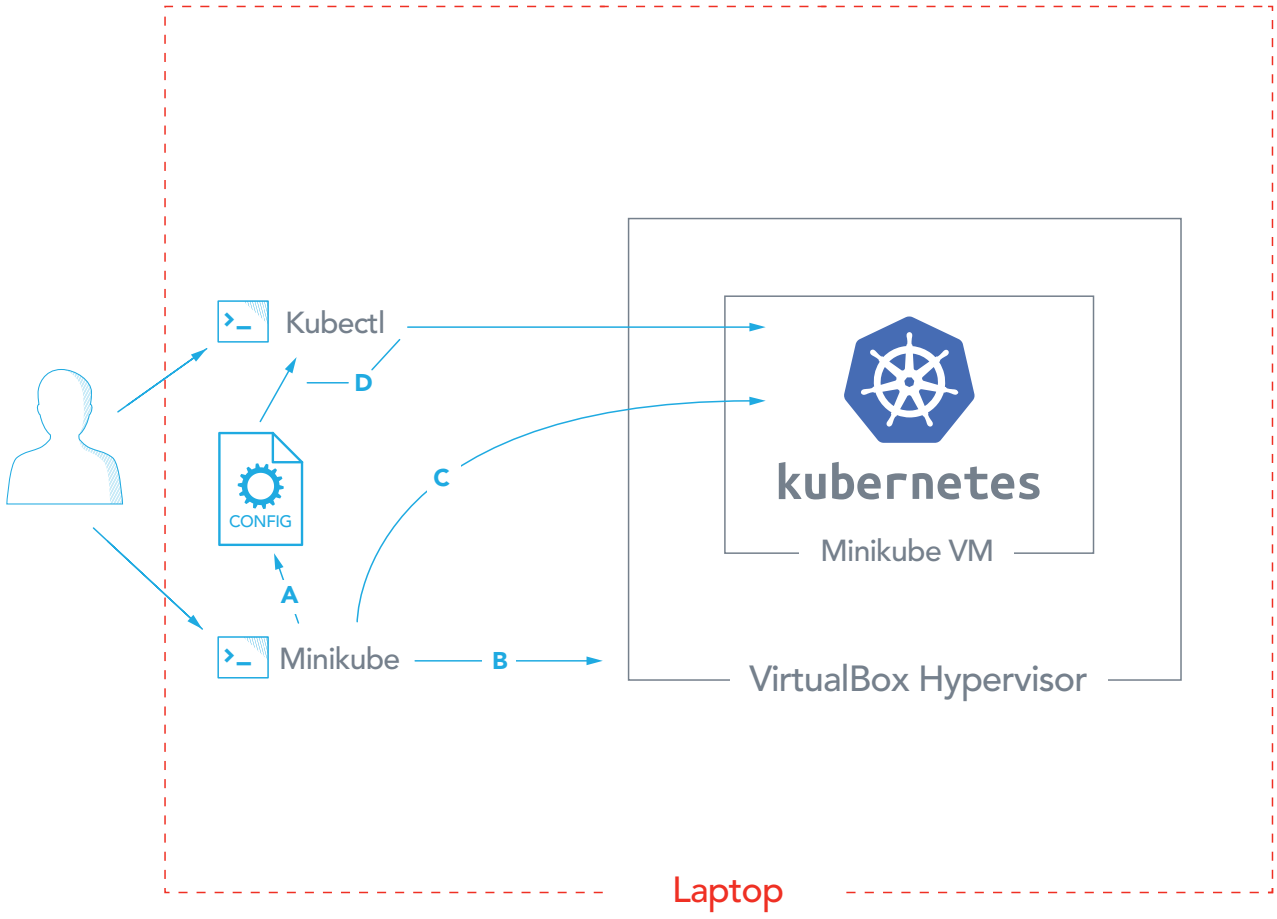
The two all-in-one deployment options described below install Kubernetes as a single host or on your laptop.

Kubernetes Deployment Using Minikube

- Use case: Developer sandbox
- Pros: Convenience, localized sandbox, testing, exploration
- Cons: Unsuitable for production, cannot scale, cannot be shared

A developer's first interaction with Kubernetes is usually on a laptop, often with Minikube. Using Minikube, a single-node Kubernetes "cluster" can be deployed on locally as a virtual machine. Minikube supports a variety of different operating systems (OSX, Linux, and Windows) and Hypervisors (Virtualbox, VMware Fusion, KVM, xhyve, and Hyper-V).

The following figure provides some details on setup with Minikube on a single host.



A: Minikube generates kubeconfig file
B: Minikube creates Minikube VM
C: Minikube sets up Kubernetes in Minikube VM
D: Kubectl uses kubeconfig to work with Kubernetes

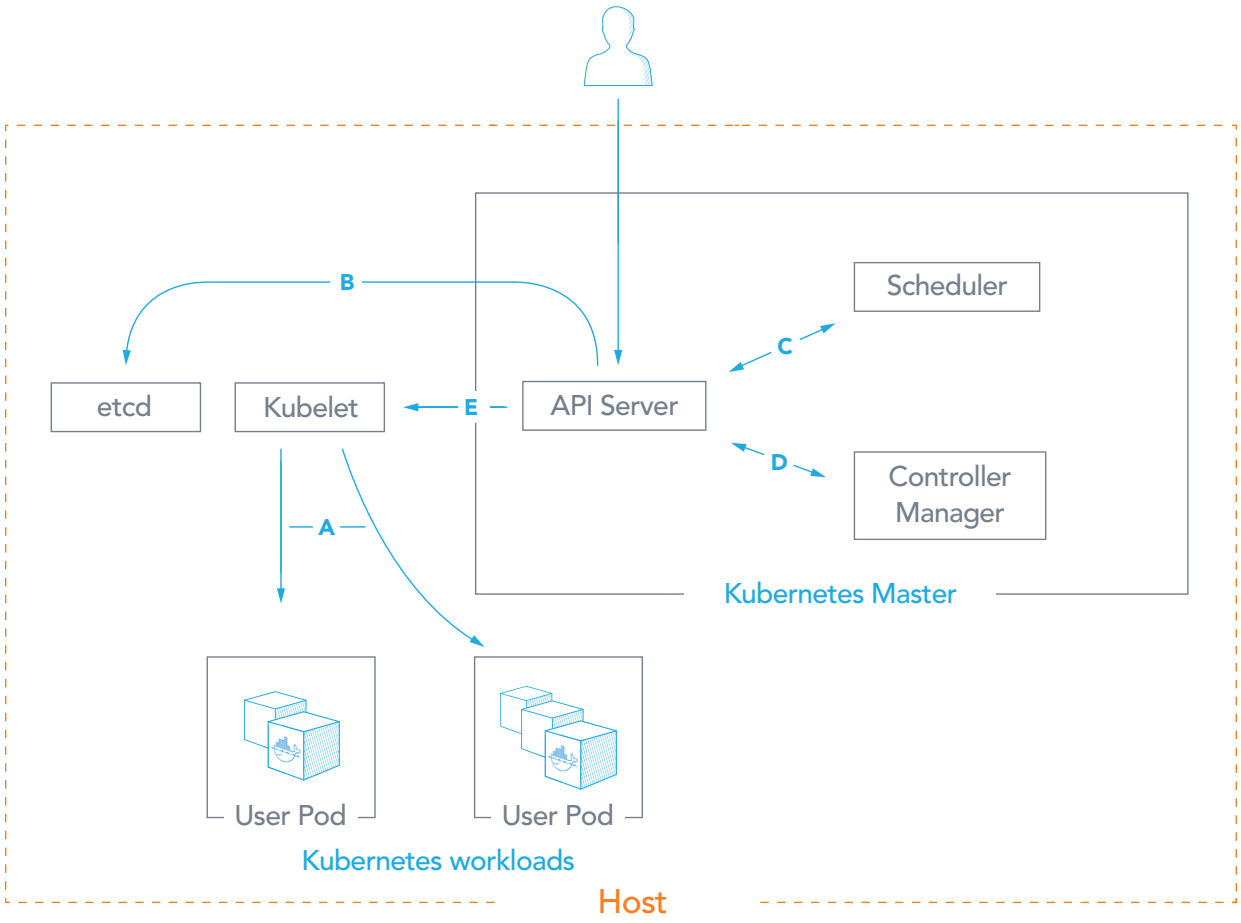
In order to deploy Kubernetes with Minikube, please refer to Appendix A.

Kubernetes Deployment Using Containers

- Use case: Developer sandbox
- Pros: Convenience, localized sandbox, testing, exploration
- Cons: Unsuitable for production, cannot scale, cannot be shared

Docker has made it very easy to try software frameworks quickly in contained environments, since the installation of a package and its dependencies in a container does not install them on the host. Kuberentes can be easily deployed as a set of Docker containers on a single host. The host can be a physical server, a virtual machine or your laptop.

The following figure shows the architecture of Kubernetes running as a set of containers on your host.



A: Kubelet manages pods running in the host
B: API Server updates state on etcd
C: API Server works with scheduler to decide where to place pods
D: Controller Manager works with API Server to determine if pods need to be replicated
E: Kubelet receives pod specifications from the API Server

As you can see from the figure above, Kubernetes components run in the form of multiple containers in this deployment model:

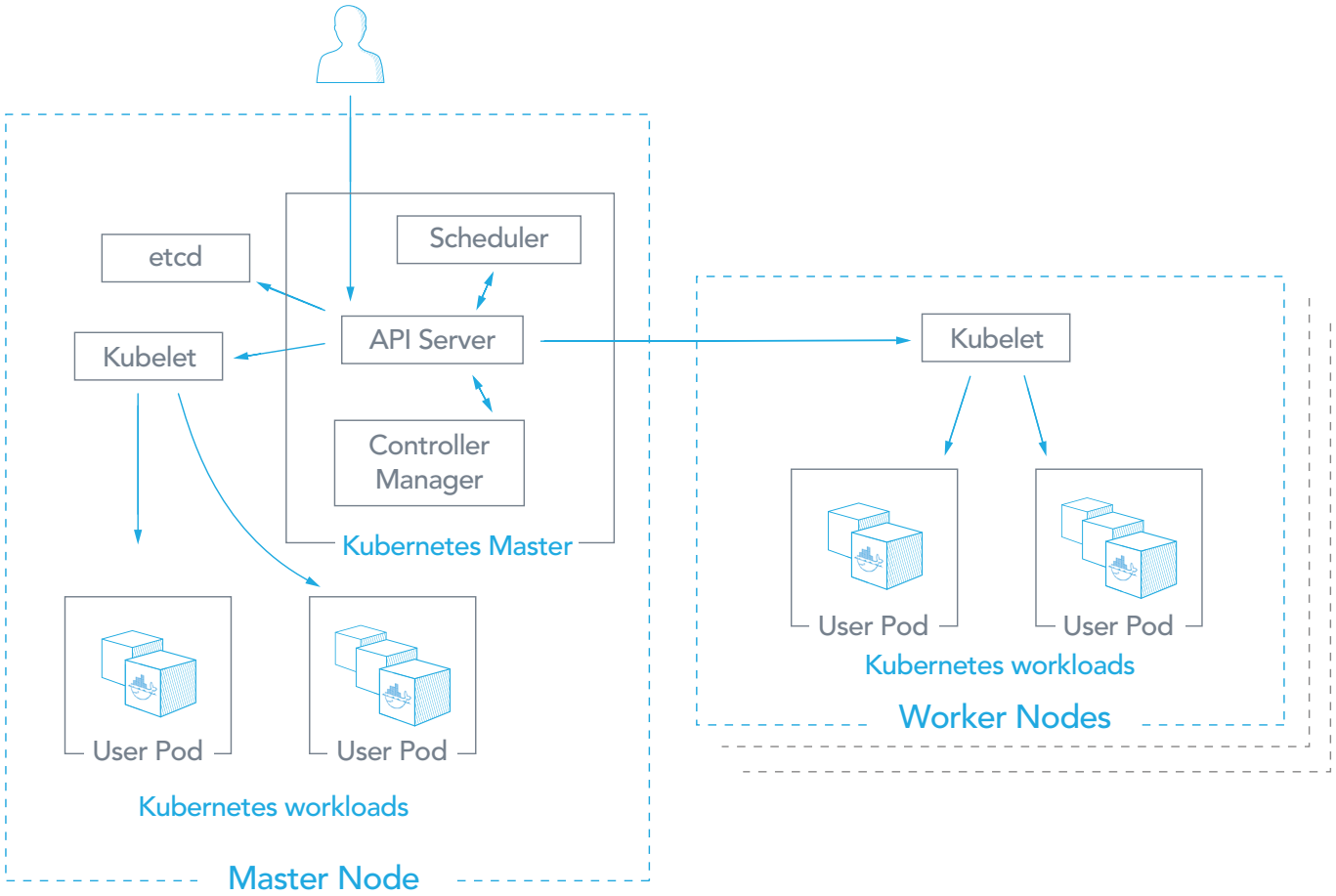
- **etcd** - This component stores configuration data which can be accessed by the Kubernetes Master's API Server by simple HTTP or JSON API.
- **API Server** - This component is the management hub for the Kubernetes Master node. It facilitates communication between the various components, thereby maintaining cluster health..
- **Controller Manager** - This component ensures that the clusters' desired state matches the current state by scaling workloads up and down.
- **Scheduler** - This component places the workload on the appropriate node - in this case all workloads will be placed locally on your host.

For step-by-step instructions on Kubernetes deployment using containers, please refer to Appendix B.

Kubernetes Deployment with an Installer

- **Use case:** Customized Kubernetes deployment
- **Pros:** Customizability, Scalability
- **Cons:** Time and resource intensive, YMMV (your mileage may vary), Requires ongoing maintenance

This method usually installs Kubernetes on one or more nodes which are either servers in your datacenter, virtual machines in your datacenter or a public cloud. Installers work best for users who are technically skilled to understand the underlying design of Kubernetes, and who are capable of addressing and resolving issues with the setup at any time. Users will also need to invest their own efforts or use external solutions to enable enterprise requirements such as scalability, high availability and monitoring. The diagram below shows an example of a Kubernetes cluster with one master node and multiple worker nodes.



Kubernetes Deployment with kubeadm

kubeadm's CLI capabilities can setup Kubernetes clusters in virtual or physical infrastructure. While, in our experience, kubeadm provided a simpler way to deploy Kubernetes when compared to kops and kargo, it does not support highly-available Kubernetes clusters. At the time of this publication, the CLI is in beta, and other parts of the installer are in alpha. Refer to Appendix C for detailed instructions on deploying Kubernetes with kubeadm.

Kubernetes Deployment with kops

kops is another CLI tool that orchestrates Kubernetes cluster deployment using a cluster specification document provided by the user. Currently the only supported deployment platform is Amazon Web Services, but kops offers other advantages, including high availability for your clusters, and addons for networking and monitoring, which include Flannel, Calico, and Dashboard. The user doesn't have to initially setup infrastructure because kops deploys the required AWS resources, such as EC2 instances, EBS storage, and VPC networking. Appendix D has more details on Kubernetes installation with kops.

Kubernetes Deployment with kargo

kargo is also a CLI driven tool used to setup Kubernetes clusters, but across multiple platforms such as Amazon Web Services, Google Compute Engine, Microsoft Azure, OpenStack and bare metal servers. It uses Ansible and requires users to customize Ansible state files (inventory and variable files). Familiarity with Ansible 2.x is expected. Appendix E has further details on deploying Kubernetes with kargo.

CoreOS Tectonic

CoreOS Tectonic, a commercial Kubernetes distribution, is best known for its on-premises deployment product. Tectonic comes with support, the Tectonic installer, and a GUI console. At this time, it enables GUI deployment on Amazon Web Services and bare metal. Products in development include Terraform-based deployment on Microsoft Azure (alpha) and OpenStack (pre-alpha).

Everything from scratch

You can also build your Kubernetes cluster from scratch without using any of these tools. The Kubernetes documentation page calls out instructions to on how one can achieve that. If your goal is to gain a deep understanding of Kubernetes, setting up Kubernetes without the help of the above tools can be useful. We recommend looking at Kelsey Hightower's "Kubernetes the Hard Way" tutorial.

- continued on next page -

Kubernetes as a Service

Kubernetes can be consumed as a service by users looking for a faster, easier solution which would allow them to focus on building software rather than managing containers. Well known examples of Managed Kubernetes service include Kube2Go.io, StackPoint.io, and Platform9 Managed Kubernetes solutions.

Platform9 Managed Kubernetes (PMK)

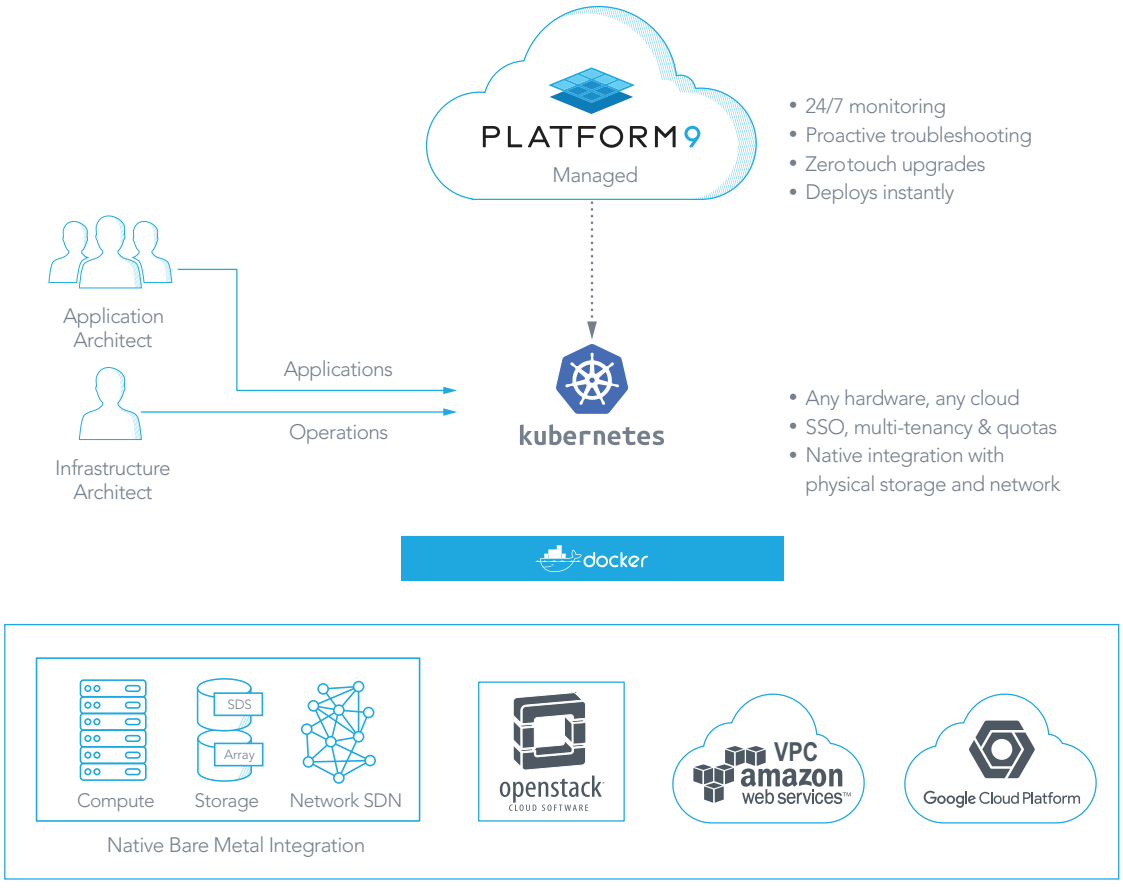
PMK offers a truly enterprise-grade managed Kubernetes service that works across any underlying infrastructure: including physical server infrastructure, virtualized environments or public clouds such as AWS, Azure and GCP.

Platform9 Managed Kubernetes is provided as a SaaS-managed solution, with deployment, monitoring, troubleshooting and upgrades being taken care of by Platform9. Therefore, the operational SLA for Kubernetes management is provided by Platform9.

Apart from the service being fully managed and working across any server or cloud infrastructure, Platform9 Managed Kubernetes features several common enterprise integrations:

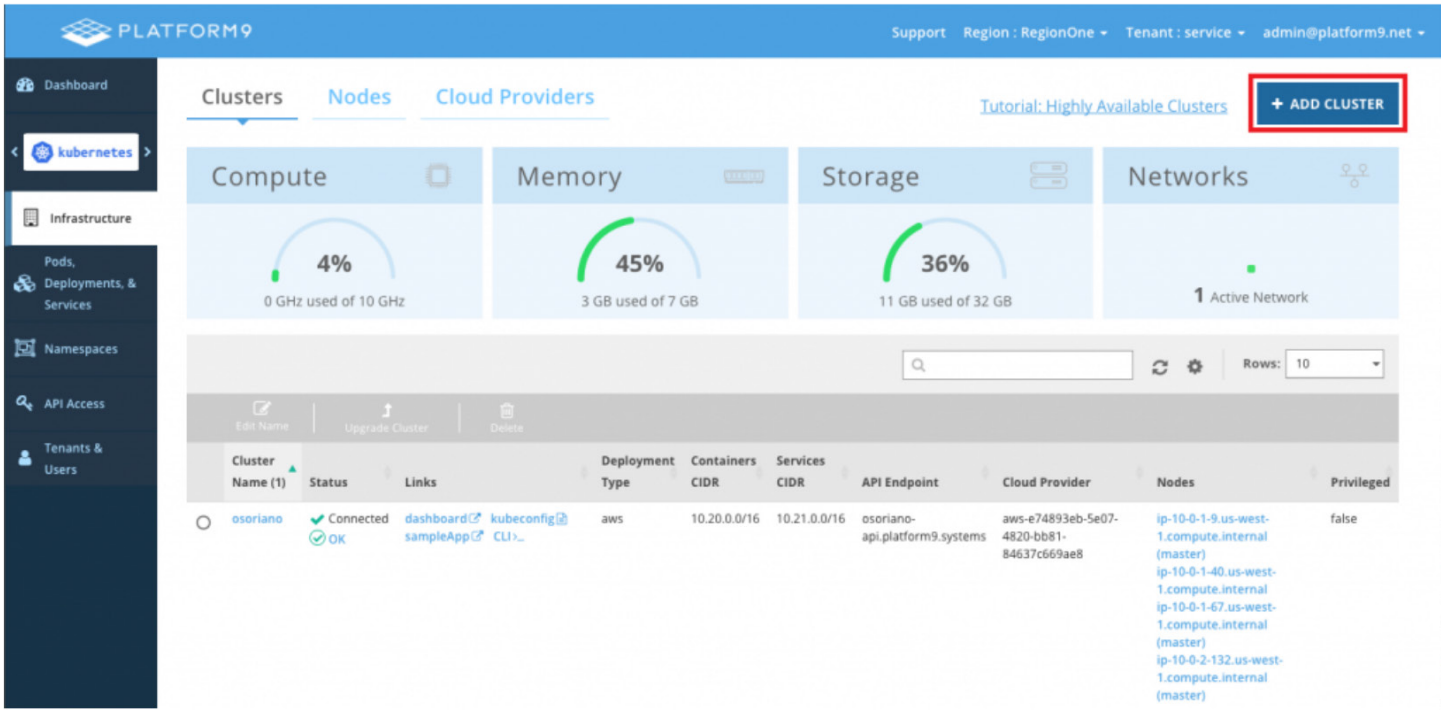
1. Single pane view of multiple clusters
2. Highly available, multi-master Kubernetes clusters that are automatically scaled-up and scaled-down based on workloads
3. Common enterprise integrations such as SSO / isolated namespaces; and the ability to deploy applications via Helm charts
4. Cluster federation that provides a truly seamless hybrid environment across multiple clouds or data-centers

The graphic outlines PMK's SaaS architecture, and its flexibility to work across clouds and hardware.



In order to get started with Platform9, you can deploy a free sandbox. The sandbox includes a guided walkthrough for SaaS-managed Kubernetes. To create an on-premises cluster, install a supported Linux operating system on hosts with Internet access, download the installer for the Platform9 host agent, and apply it on the hosts. Alternatively, you can deploy your Kubernetes cluster on a public cloud such as Amazon Web Services by supplying user credentials for your public cloud environment.

Shown below is a screenshot of the Platform9 Kubernetes UI with a cluster on Amazon Web Services. New clusters can be added by using the "Add Cluster" button on the top right of the dashboard. New nodes can be deployed using the "Nodes" tab, and cloud credentials can be specified using the "Cloud Providers" tab.



StackPoint.io

StackPoint.io offers integrations with a comprehensive set of public cloud providers on top of which to build your container clusters, including AWS, Google Compute Engine/Container Engine, Digital Ocean, Azure and Packet.

Persistent data support is available via persistent volumes, and it even includes support for TPM with selected providers. CoreOS is the widely used operating system with StackPoint, but they announced additional support for Ubuntu on Amazon Web Services in April 2017. Users can leverage a number of Kubernetes native solutions that come pre-installed, including Sysdig, Prometheus, and Deis, among others.

Like PMK, StackPoint includes the Kubernetes dashboard, cluster federation, and multi-master clusters to enable highly available applications. However, it isn't easy to use server infrastructure running on-premises or in datacenters, with support for a broader range of OSes.

Kube2Go.io

While PMK is enterprise-grade managed Kubernetes solution from Platform9, they also offer Kube2Go.io, as a free community-focused incarnation that allows users many of the same benefits as PMK for up to 5 nodes. Though Kube2Go's free version does not come backed with the enterprise support that PMK provides, users still get the benefit of having their Kubernetes clusters managed for them, along with version upgrades.

Deployment on Hosted Cloud Infrastructure

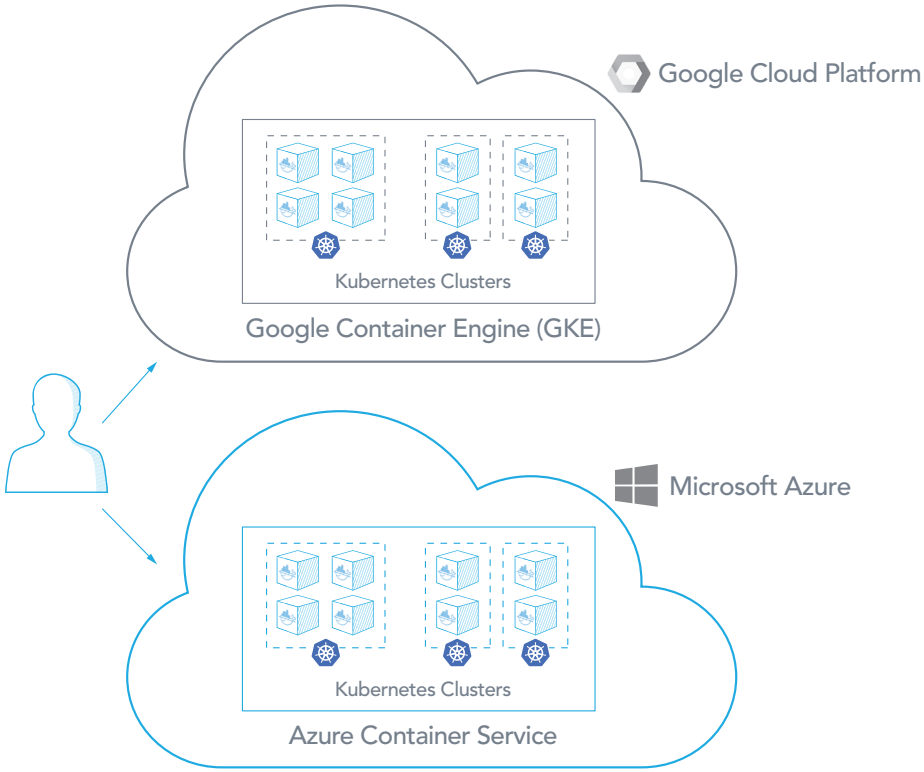
If placing all of your data and workloads in a public cloud is acceptable, the easiest way to deploy and consume Kubernetes is through a hosted service provided by a major public cloud vendor. The two prominent options today are Google Container Engine (abbreviated GKE to distinguish it from Google Compute Engine) and Azure Container Service (ACS).

Not surprisingly, GKE has been around much longer and is more mature, a result of Kubernetes being created at Google. GKE is just another service in the large and sprawling set offered as part of the Google Cloud Platform. If you are already using GCP or GCE (the IaaS portion of the platform), GKE is just a click away in the Google cloud web console and integrates with the existing Identity and Access Management. When creating a new cluster, you specify an OS (a traditional Linux or a an OS highly optimized for containers), an instance size, and the cluster size (number of worker nodes). GKE automatically creates and manages a master node for the cluster. That node is opaque and you cannot access it. You can however access (via SSH) - and are billed for - the worker nodes, which are ordinary GCE instances.

A GKE cluster can easily be expanded by adding worker nodes to its default node pool, or adding additional node pools. GKE stands out in its ability to gracefully handle upgrades: master node (API) upgrades happen automatically and transparently a few weeks after every new Kubernetes release, so your cluster's API version is kept up-to-date with new features and bug fixes. While worker node upgrades have historically required a manual user action, GKE has recently added an experimental flag allowing worker nodes to be upgraded automatically as well. Another notable recent addition is support for Federation, allowing multiple clusters to co-operate across the globe, enabling highly available and low-latency web applications.

ACS is a much younger offering from Microsoft. Unlike GKE, ACS supports multiple container orchestration engines, including Docker Swarm and Mesos DCOS. Interestingly, Kubernetes support did not land until recently, in February 2017. Compared to GKE, the quality of ACS' Kubernetes offering will probably take time to match the feature depth and maturity of Google's offering. The need to support other container engines could also be viewed as a potential downside, possibly limiting the ACS team's ability to optimize and build out the Kubernetes product. On the other hand, ACS may prove to be an interesting option for users interested in running .NET applications on Windows Server OS. Docker and Kubernetes support for Windows is young and evolving, but if there's one company that's going to champion and pour resources into it, it's Microsoft. It would therefore not be surprising for ACS to evolve into an excellent (if not the only) choice for running Windows workloads on Kubernetes.

The figure below provides a summarizes this deployment model on these two public clouds.



Synopsis: Kubernetes Deployment Models Compared

This document has reviewed the major deployment models to run Kubernetes. Since the ideal deployment model depends on your specific use case and goals, here is a quick comparison of the major alternatives and their associated pros and cons, as of May 2017.

	Developer Installs		Kubernetes Installer				Kubernetes as a service			Kubernetes hosted infrastructure	
	Minikube	Docker	kubeadm	kops	kargo	CoreOS Tectonic	PMK	StackPoint.io	Kube2Go.io	GKE	ACS
Supported Infrastructure											
Laptops	✓	✓	✓								
Data-Center/ Colocated Hardware			✓		✓	✓	✓				
Public clouds			✓	✓+	✓	✓	✓	✓	✓	✓	✓
Hybrid Deployments#							✓*				
Lifecycle Management											
Built-in Monitoring							✓	✓	✓	✓	✓
Commercial Support/SLA						✓	✓	✓		✓	✓
Kubernetes Upgrades				✓	✓	✓	✓	✓	✓	✓	✓
Enterprise-Readiness											
Cluster High Availability				✓	✓	✓	✓	✓	✓	✓	✓
RHEL, CentOS & Ubuntu Node OS Support	✓	✓	✓		✓	✓	✓	✓+	✓		
SSO Integration						✓	✓			✓	
Isolated Networking/ Network Policies	✓	✓	✓	✓	✓	✓	✓	✓			
Dynamic Persistent Volumes			✓	✓	✓		✓	✓	✓	✓	✓
Federation							✓*	✓	✓*	✓	

* Coming soon

\+ for Amazon Web Services only

\# hybrid implies the ability to deploy clusters to both on-premises and cloud, and support some integration between them - high availability, disaster recovery, or bursting

Table is based on data available as of May 2017

Conclusion

This guide has walked you through a number of different deployment models for Kubernetes. Developer installers, such as Minikube, provide a way to get started with Kubernetes on your laptop. However, if you're looking to scale Kubernetes infrastructure, it is necessary to consider installers such as kubeadm, kops, and kargo. While this approach allows the most customized Kubernetes deployments, it requires time and effort in managing Kubernetes itself.

Developers looking for a solution to get their applications running as quickly as possible using Kubernetes, with features such as high availability, single sign-on, and federation, should consider Kubernetes-as-a-Service offerings, including Platform9. These solutions allow you to focus on deploying containerized applications with lower overhead costs to upgrade or maintain the cluster.

Finally, Kubernetes can be deployed on Google or Azure clouds using GKE and ACS. These solutions work well if your cluster infrastructure lives on Google Cloud or Azure. However, the benefit of managed Kubernetes offerings like Platform9 is the ability to deploy outside these major clouds, including in your own private infrastructure within your network perimeter.

You are now well equipped to pick your Kubernetes deployment. Should you have any questions or feedback on this document, tweet us @Platform9Sys or email us at connect@platform9.com.

Appendix A: Deployment with Minikube

In order to install Minikube, follow the following steps:

1. Download the package:

   ```
   # curl -Lo minikube \
   https://storage.googleapis.com/minikube/releases/v0.18.0/minikube
   -darwin-amd64 && chmod +x minikube && mv minikube /usr/local/bin/
   ```

2. Download and install the Kubernetes command line too kubectl by running the following command:

   ```
   # curl -Lo kubectl \
   https://storage.googleapis.com/kubernetes-release/release/v1.6.0/
   bin/darwin/amd64/kubectl && chmod +x kubectl && mv kubectl
   /usr/local/bin/
   ```

3. Start Minikube by running the following command:

   ```
   # minikube start
   Starting local Kubernetes cluster...
   Starting VM...
   SSH-ing files into VM...
   Setting up certs...
   Starting cluster components...
   Connecting to cluster...
   Setting up kubeconfig...
   Kubectl is now configured to use the cluster.
   ```

   You now have a local, single-node Kubernetes "cluster" on your laptop. In order to confirm that Kubernetes is installed, you can run the following commands:

   ```
   # cat > example.yaml<<EOF
   apiVersion: apps/v1beta1
   kind: Deployment
   metadata:
           name: nginx-deployment
   spec:
           replicas: 1
           template:
           metadata:
                   labels:
                           app: nginx
           spec:
                   containers:
                   - name: nginx
                           image: nginx
                           ports:
                           - containerPort: 80
   EOF
   # kubectl create -f example.yaml
   ```

After a few seconds, you should see output similar to the following:

```
# kubectl get pods
NAME                                READY     STATUS    ...
nginx-deployment-2122188915-3s69g   1/1       Running   ...
```

Appendix B: Deployment with Containers

To get started with Kubernetes using Docker containers on a local host, run the Kubernetes components as Docker containers as outlined below:

1. The version of Kubernetes can be changed as desired. Let's use 1.5.6 for this exercise. Set K8S_VERSION environment variable to 1.5.6 using the following command:

   ```
   # export K8S_VERSION=v1.5.6
   ```

2. Run kubelet using the hypercube image from the gcr.io container registry. Kubelet will bootstrap the Kubernetes master components (api-server, scheduler, controller-manager) and other services such as etcd as individual Docker containers. Note that we run kublet also as a docker container (--containerized).

   ```
   # docker run \
           --volume=/:/rootfs:ro \
           --volume=/sys:/sys:ro \
           --volume=/dev:/dev \
           --volume=/var/lib/docker/:/var/lib/docker:rw \
           --volume=/var/lib/kubelet/:/var/lib/kubelet:rw \
           --volume=/var/run:/var/run:rw \
           --net=host \
           --pid=host \
           --privileged=true \
           -d \
           gcr.io/google_containers/hyperkube:${K8S_VERSION} \
           /hyperkube kubelet --containerized \
           --hostname-override="127.0.0.1" --address="0.0.0.0" \
           --api-servers=http://localhost:8080 \
           --config=/etc/kubernetes/manifests
   ```

3. Run service proxy, which is used to load balance ingress requests to service endpoints:

   ```
   # docker run -d --net=host --privileged \
   gcr.io/google_containers/hyperkube:${K8S_VERSION} \
   /hyperkube proxy --master=http://127.0.0.1:8080 --v=2
   ```

   You should now be able to run kubectl directly on the same node.

   ```
   # ./kubectl get nodes
   NAME        STATUS    AGE
   127.0.0.1   Ready     2m
   ```

   If you are running docker on OS/X then you will need to forward port 8080. This can be done as described in this **link**.

Appendix C: kubeadm Installer

In order to deploy, follow these steps on nodes which will be part of the cluster:

1. Ensure that all nodes have full network connectivity. Disable any firewalls.

2. On all nodes, setup upstream Kubernetes repositories for your Linux hosts. Shown below is an example of how to set it up on Ubuntu 16.04, but the process would be similar for CentOS operating systems.

   ```
   # curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
   apt-key add -
   # cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
   deb http://apt.kubernetes.io/ kubernetes-xenial main
   EOF
   # apt-get update
   ```

3. On all nodes, install kubelet, kubeadm, Docker, kubectl and kubernetes-cni from the Kubernetes upstream repositories. On Ubuntu 16.04 you would run:

   ```
   # apt-get install -y kubelet kubeadm kubectl kubernetes-cni
   docker-engine
   ```

4. You can now build your Kubernetes cluster. Start by setting up the master. On the master node, run the following commands:

   ```
   # kubeadm init *
   # cp /etc/kubernetes/admin.conf $HOME/
   # chown $(id -u):$(id -g) $HOME/admin.conf
   # export KUBECONFIG=$HOME/admin.conf
   ```

   * Note: in the case of Flannel, you need to provide a pod network CIDR parameter to the kubeadm init command:

   ```
   # kubeadm init --pod-network-cidr=10.244.0.0/16
   ```

   The kubeadm init command will do some initial setup of the cluster master. It will also give you a token for this cluster. This token should be kept safely because this is used to add other nodes to the cluster. This step also gives you an admin.conf file, which you will use as the kubeconfig file when you need to perform operations on the cluster.

   You can now setup networking for the cluster. Kubeadm cluster comes with container networking (CNI) support. You can setup any CNI compatible networking backend pod on the Kuberenetes cluster using kubectl:

   ```
   # kubectl apply -f <networking.yaml>
   ```

   Starting with Kubernetes 1.6, you may need to run RBAC policies for networking plugins, such as Flannel and Canal.

5. Add worker nodes to the cluster. On each node that you want to act like a worker, run:

   ```
   # kubeadm join --token <token> <master-ip>:<master-port>
   ```

   The token you provide in this command should be the token you obtained from step 3. You are now ready to use a Kubernetes cluster. Use the kubeconfig from step 3 with kubectl to operate the cluster.

Appendix D: kops Installer

To build the Kubernetes cluster, kops then sets up Kubernetes services on each node.

1. First, download kops from Github:

   ```
   # wget \
   https://github.com/kubernetes/kops/releases/download/1.6.0/kops-linux-amd64
   # chmod +x kops-linux-amd64
   # mv kops-linux-amd64 /usr/local/bin/kops
   ```

2. Prepare your AWS account.
   a. You need setup EC2, Route53, IAM, S3, and VPC IAM permissions for the kops user using the AWS Management Console. The kops user requires the following IAM permissions:

   ```
   AmazonEC2FullAccess
   AmazonRoute53FullAccess
   AmazonS3FullAccess
   IAMFullAccess
   AmazonVPCFullAccess
   ```

   Once you create your user, note the access key id and secret key.

   b. Download and install the AWS CLI using instructions provided in **AWS Documentation**. Enter the following commands:

   ```
   # aws configure
   <Enter access key id and secret key>
   # export AWS_ACCESS_KEY_ID=<access key id>
   # export AWS_SECRET_ACCESS_KEY=<secret key>
   ```

   c. You need configure your DNS based on whether your domain/subdomain is hosted in AWS, domain is hosted on another registrar, or subdomain is hosted on AWS and domain is hosted by an external registrar. Detailed instructions on configuring your DNS can be found in the kops **Github repository**.

3. Create a S3 bucket in AWS to store state information about your cluster. Enter the following commands:

   ```
   # aws s3api create-bucket \
   --bucket my-kops-cluster-store \
   --region us-west-1

   # aws s3api put-bucket-versioning \
   --bucket my-kops-cluster-store \
   --versioning-configuration Status=Enabled
   ```

4. Create a cluster configuration that will be used to setup the Kubernetes cluster. Note that the cluster name must be a fully-qualified DNS name within a public Route53 hosted zone whose records can be edited by the AWS account.

   ```
   # export KOPS_STATE_STORE=s3://my-kops-cluster-store

   # kops create cluster \
           --zones us-west-1a \
           --name=<cluster.example.com>
   ```

Note:

- The above command will only create a cluster configuration file. The cluster will be created in a subsequent step using the kops update command.

- If you do not want to use the SSH key in ~/.ssh/id_rsa.pub, specify your SSH key with the flag --ssh-public-key=<path/to/key>

5. Review and change the cluster configuration:

   # kops edit cluster <cluster.example.com>

6. Setup the Kubernetes cluster using kops:

   # kops update cluster <cluster.example.com> --yes

Appendix E: kargo Installer

1. Kargo requires the following packages:

   Ansible v2.3 (at the time of this publication)
   Jinja 2.9

   In addition, ensure that the nodes have Internet access, firewalls disabled, and SSH key-based logins enabled.

2. Setup the kargo CLI

   # pip install kargo

3. Edit the kargo yaml file (in /etc/kargo/kargo.yml) according to your deployment needs. For example, if you are going to use AWS to setup your cluster, you will need to provide access keys, AMIs, instance types, regions, and other parameters.

4. Setup the nodes to build the cluster.

   a. Pre-existing hosts: Ensure these hosts are up and running. SSH is a prerequisite for Ansible. Therefore all hosts should also be reachable over SSH prior to Kubernetes cluster setup. You should build an Ansible inventory file to represent these pre-existing hosts. A sample inventory file is available on Github.

   # kargo prepare --nodes node1 …

   b. Create hosts with cloud providers: Instead of using pre-existing hosts, you can also create new instances on clouds like AWS, Google Cloud Platform, and OpenStack to be used as Kubernetes nodes. For example, you can create 3 nodes (including master) and 3 etcd nodes on AWS with the command below:

   # kargo aws --nodes 3 --etcds 3

5. You are now ready to deploy the Kubernetes cluster. Doing so will setup all of the software needed by Kubernetes depending on the entries in the kargo.yaml and the inventory file that was generated above.

   # kargo deploy [-i <path to inventory file>]