What is Kubernetes?

For managing containerized workloads and services, Kubernetes is a portable, extensible, open-source platform that supports declarative configuration and automation. It has a huge, expanding ecosystem. Services, assistance, and tools for Kubernetes are widely accessible.

It is a platform for managing and deploying containers that is open source. Currently, it provides service discovery, load balancing, self-healing methods, container orchestration, a container runtime, and infrastructure orchestration focused on containers. Across host clusters, it is used to compose, scale, deploy, and manage application containers.

Kubernetes Architecture:

Kubernetes client server architecture. By putting containers into Pods to run on Nodes, Kubernetes manages your workload. Depending on the cluster, a node could be either a virtual or physical computer. The control plane controls each node, which has the services required to execute Pods.

In a cluster, you typically have several nodes, though you might only have one node in a learning or resource-constrained environment. A node's components are the kubelet, container runtime, and kube-proxy.

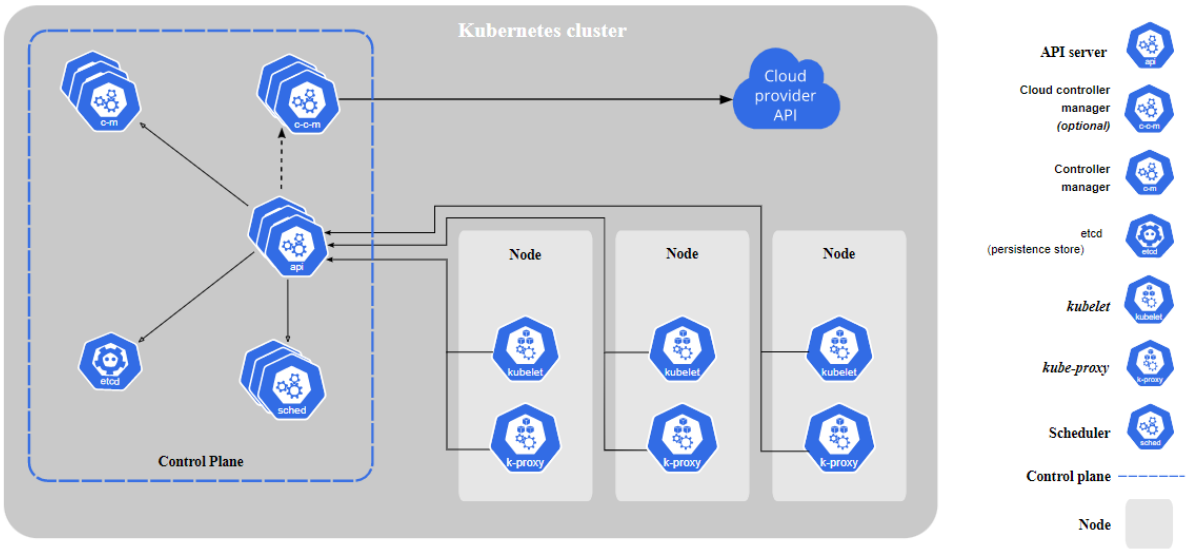
Only a modern, distributed application services platform can deliver an ingress gateway for applications created utilizing the Kubernetes **micro services** architecture. Web-scale, cloud-native applications supplied as micro services using container technology cannot be managed by conventional appliance-based ADC solutions for Kubernetes container clusters. Each one can support thousands of containers across hundreds of pods, making policy-driven deployments, total automation, and elastic container services necessary.

Kubernetes Components:

A cluster is created when Kubernetes is deployed. A group of worker computers, known as nodes, that run containerized apps make up a Kubernetes cluster. There is at least one worker node in each cluster.

The Pods that make up the application workload are hosted on the worker node(s). The cluster's worker nodes and Pods are controlled by the control plane. To provide fault tolerance and high availability, the control plane typically runs across many computers in production scenarios, and a cluster typically contains numerous nodes.

The different parts you need to have for a complete and functional Kubernetes cluster are described in this document.



There are two main types of components:

1. Control Plane Components:
   The components of the control plane take general cluster decisions (such as scheduling), as well as notice and react to cluster events (such as starting a new pod when a deployment's replicas field is not satisfied).
2. Node Components:
   Every node has components running on it that keep running pods up and running and provide the Kubernetes runtime environment.

Kubernetes Extensibility:

Kubernetes is very extensible and configurable. As a result, forking or submitting fixes to the code of the Kubernetes project is rarely necessary.

Software components known as extensions tightly integrate and augment Kubernetes. They modify it to support brand-new varieties of hardware. A hosted or distributed instance of Kubernetes is used by many cluster admins. The extensions for these clusters are already installed. As a result, few users will need to create new extensions and the majority of Kubernetes users won't need to install extensions.

Five Types of extensions:

- Client extensions
- API extensions
- API access extensions
- Infrastructure extensions
- Scheduling extensions

Kubernetes Performance Measurements:

We decided to define performance and scalability goals based on the following two metrics:

1. *"API-responsiveness"*: 99% of all our API calls return in less than 1 second
2. *"Pod start-up time"*: 99% of pods (with pre-pulled images) start within 5 seconds

How do we measure?

We established a continuous testing environment to track performance gains and spot regressions. We build a 100-node cluster from HEAD every two to three hours and perform our scalability tests on it. As the master, we use a GCE n1-standard-4 (4 cores, 15 GB RAM) system, while the nodes are GCE n1-standard-1 (1 core, 3.75 GB RAM) devices.

In scalability tests, we specifically concentrate primarily on the full-cluster case, which is the most demanding from a performance standpoint (full N-node cluster is a cluster with 30 * N pods running in it). We go through the following stages to mimic what a customer may actually do:

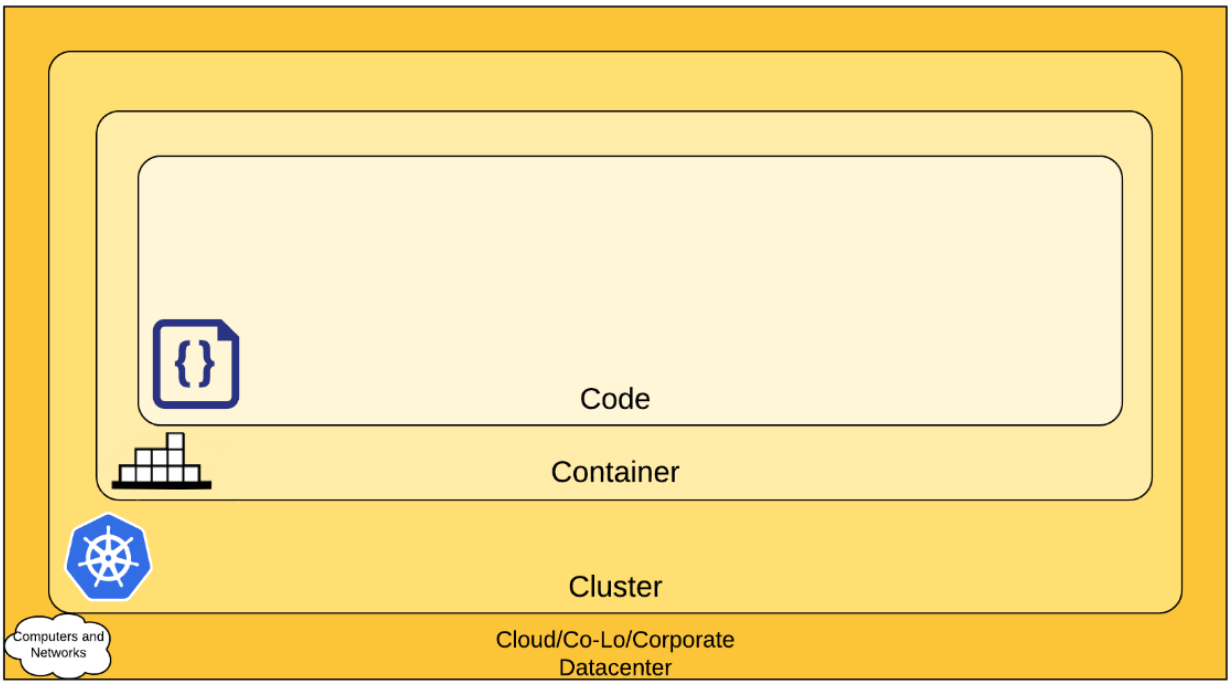
- Populate pods and replication controllers to fill the cluster
- Generate some load (create/delete additional pods and/or replication controllers, scale the existing ones, etc.) and record performance metrics
- Stop all running pods and replication controllers
- Scrape the metrics and check whether they match our expectations

Security:

Cloud Native Security:

Security can be thought of in layers. Cloud, Clusters, Containers, and Code are the four pillars of cloud native security.

Each layer of the Cloud Native security model builds upon the next outermost layer. The Code layer benefits from strong base (Cloud, Cluster, Container) security layers. You cannot safeguard against poor security standards in the base layers by addressing security at the Code level.



Pod Security Standards:

Three distinct policies are laid out in the Pod Security Standards to cover the full range of security issues. These regulations range from being very permissive to being very restrictive and are cumulative. The guidelines for each policy are described in this manual.

The Three Policies are:

- **Privileged**
  Unrestricted policy offering the broadest range of permissions is available. This policy permits known privilege increases.
- **Baseline**
  Policy with minimal restrictions that prevents known privilege escalations enables the minimally defined default Pod configuration.
- **Restricted**
  Pod hardening best practices are followed by a strict policy.

Security For Windows Nodes:

As opposed to using tmpfs or in-memory filesystems on Linux, data from Secrets is written out in clear text onto the node's local storage on Windows. You must take the additional actions listed below as a cluster operator:

1. Use file ACLs to secure the Secrets' file location.
2. Apply volume-level encryption using BitLocker.

Firewall:

Comprehensive Web Application Protection with Avi Vantage

- Avi's iWAF is 100% software WAF security solution and provides scalable app security, threat detection, and application protection using point-and-click simplicity with unparalleled visibility and intelligence, on-demand auto scaling in response to application security challenges and central policy management and analytics-driven security policies
- Application Rate Limiting and DDoS Protection, Avi Vantage includes many options for rate shaping and throttling of traffic. This may be applied at the virtual service, pool/server, or client level. Per-application rate limiting and granular control, protection against L4 and L7 denial of service (DoS) attacks and customizable via data scripts to create specific policies
- Avi natively implements HTTP Strict Transport Security (HSTS),RSA and Elliptic Curve Cryptography (ECC) certificates Perfect Forward Secrecy (PFS) with point-and-click features and URL and IP port based allow-list and deny-list through access control lists (ACLs)

Reliability requirements:

Load balancing:
The load balancing algorithm, the backend weight scheme, and other load balancing policy parameters are bootstrapped into an Ingress controller and applied to all Ingress. Persistent sessions and dynamic weights, for example, are more sophisticated load balancing techniques but are not yet provided through the Ingress. Instead, you can acquire these functionalities by using the load balancer that a Service uses.

It's also important to note that, even though health checks aren't directly accessible through Ingress, Kubernetes has equivalent ideas like readiness probes that let you accomplish the same goal. To learn more about how the controller handles health checks, please review the relevant documentation.

Deployments:
When you specify a desired state in a deployment, the deployment controller gradually shifts the current state toward the desired state. You can specify deployments to make new ReplicaSets or to delete current deployments and replace them with new deployments that use all of their resources.

Contributions:

Kubernetes documentation contributors:

- Improve existing content

- Create new content
- Translate the documentation
- Manage and publish the documentation parts of the Kubernetes release cycle

Anyone can add a change to the kubernetes/website GitHub repository by opening an issue about the documentation or submitting a pull request (PR). To function well in the Kubernetes community, you must be at ease using git and GitHub.

To get involved with documentation:

1. Sign the CNCF Contributor License Agreement.
2. Familiarize yourself with the documentation repository and the website's static site generator.
3. Make sure you understand the basic processes for opening a pull request and reviewing changes.

Suggesting content improvements:

Open an issue if you see a mistake in the Kubernetes documentation or have a suggestion for new material. A web browser and a GitHub account are all you need.

New work on the Kubernetes documentation typically starts with a GitHub issue. Then, as required, Kubernetes contributors examine, classify, and tag issues. Next, you or another Kubernetes community member open a pull request with the necessary modifications to fix the problem.

1. Opening an issue:

Open an issue if you want to suggest changes to the current material or find an error.

- On the right sidebar, click the Create an Issue option. This takes you to a GitHub issue page that has some headers already filled in.
- Describe the problem or improvement proposal. Give as much information as you can.
- Press the New Issue button.

After submitting, periodically check in on your issue or enable GitHub notifications. Before acting on your concern, reviewers and other community members could have questions.

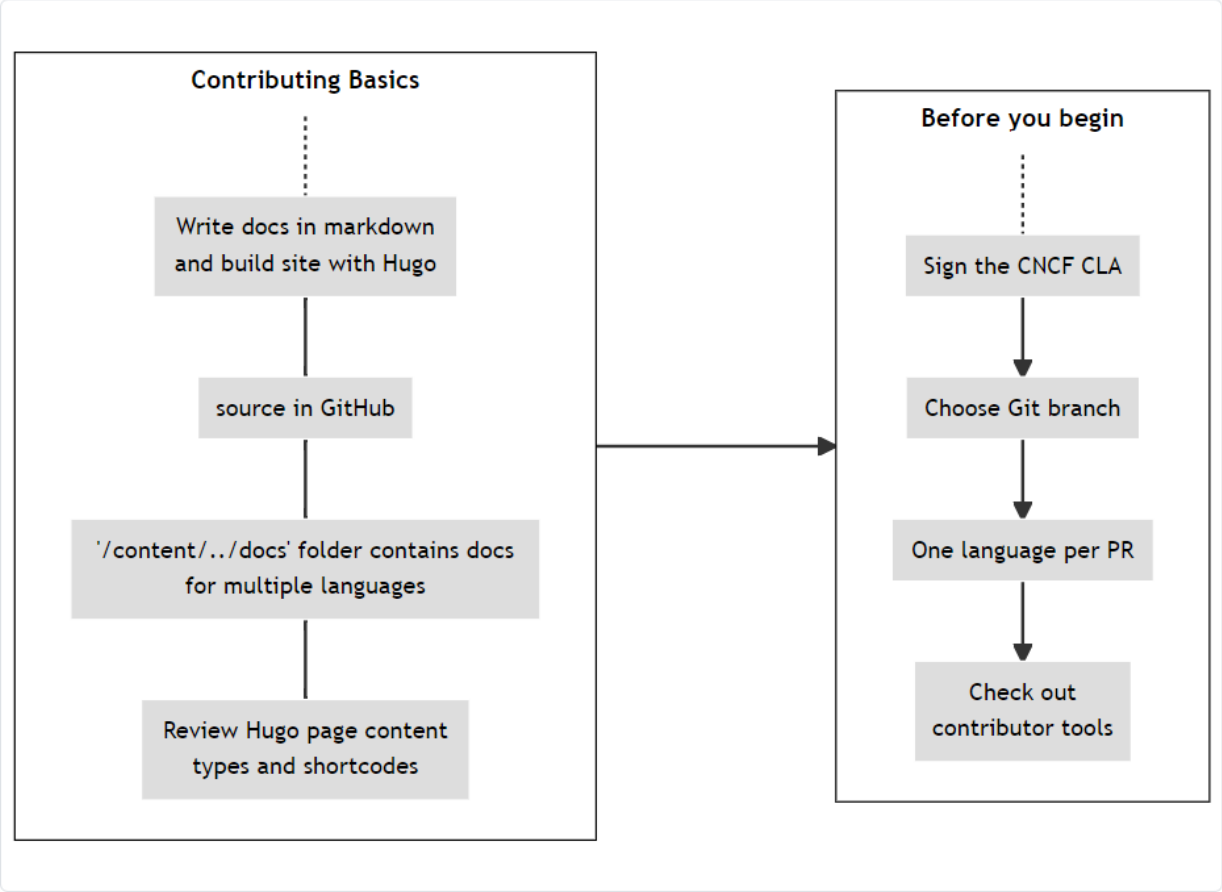
2. Suggesting new content:

You can still submit an issue if you have an idea for new material but are unsure of where it should go. Either:

- Select the section you believe the content belongs in, then click on an existing page there. Make a problem.
- Directly submit the issue on GitHub.

Contributing new content:

The details you need to be aware of prior to submitting new content are shown in the figure below.



Install and Set Up kubectl on Windows:

You must use a version of Kubectl that is identical to the cluster's version or less. A v1.25 client, for instance, can talk to v1.24, v1.25, and v1.26 control planes. Unexpected problems can be prevented by using the most recent compatible version of kubectl.

The following methods exist for installing kubectl on Windows:

Install kubectl binary with curl on Windows

Install on Windows using Chocolatey, Scoop, or Winget

Best Practices:

1. **Availability**
   Kubernetes architecture ensures high availability on the application front using replication controllers, replica sets, and pet sets. Users can set the minimum number of running pods at any time. The declarative policy can return the deployment to the desired configuration if a pod or container crashes. Configured workloads for high availability can be achieved using pet sets. Other then that Block Store (EBS), to distributed file systems such as GlusterFS and network file system (NFS), and specialized container storage plugins such as Flocker.

2. **Scalability**
   Applications deployed in Kubernetes are micro services, composed of many containers grouped into series as pods. Each container is logically designed to perform a single task. Kubernetes 1.4 supports cluster auto-scaling, and Kubernetes on Google Cloud also supports auto-scaling. During auto-scaling, Kubernetes and the underlying infrastructure coordinate to add additional nodes to the cluster when no available nodes remain to scale pods across.

3. **Portability**
   Kubernetes is made to provide options for operating systems, processor architectures, cloud platforms, container runtimes, and PaaS. For instance, CoreOS, Red Hat Linux, CentOS, Fedora, Debian, and Ubuntu all support the configuration of a Kubernetes cluster. It may be set up to run locally, on bare metal, in virtualization systems based on vSphere, KVM, and libvirt, as well as in bare metal settings. Cloud computing infrastructure like Azure, AWS, and Google Cloud can support the serverless design of Kubernetes. Combining and matching clusters from different cloud providers and on-premises locations is another way to develop hybrid cloud capabilities.

4. **Security**
   The latest three versions of Kubernetes are supported with security patches for newly identified vulnerabilities. Other than that it enables users to deactivate anonymous/unauthenticated access and use TLS encryption for connections between the API server and kubelets. Embed security early in the container lifecycle. Ensure shared goals between DevOps and security teams. Operational risks reduced using Kubernetes-native security controls. When possible, leverage native Kubernetes controls to enforce security policies so your own security controls and the orchestrator don’t collide.

How to deploy a web application using Kubernetes:

- Sign in to your Google Cloud account. If you're new to Google Cloud, create an account to evaluate how our products perform in real-world scenarios
- In the Google Cloud console, on the project selector page, select or create a Google Cloud project
- Make sure that billing is enabled for your Cloud project
- Package a sample web application into a Docker image
- Upload the Docker image to Artifact Registry
- Create a GKE cluster
- Deploy the sample app to the cluster
- Manage autoscaling for the deployment
- Expose the sample app to the internet
- Deploy a new version of the sample app

Costs to keep in mind:

- New customers get $300 in free credits to run, test, and deploy workloads.
- GKE
- Artifact Registry