

**Exhibit 3 to
Intellectual Ventures I LLC's and Intellectual Ventures II LLC's
Preliminary Infringement Contentions**

**Infringement Claim Chart of
U.S. Patent No. 7,949,785 (“the ’785 Patent”)**

The Accused Systems and Services include without limitation Southwest systems and services that utilize Kubernetes; all past, current, and future systems and services that operate in the same or substantially similar manner as the specifically identified systems and services; and all past, current, and future Southwest systems and services that have the same or substantially similar features as the specifically identified systems and services (“Example Southwest Count 1 Systems and Services” or “Southwest Systems and Services”).¹

On information and belief, Southwest’s Systems and Services directly infringe the asserted claims of the ’785 patent, either literally or under the doctrine of equivalents, through at least using Kubernetes in its various systems. In addition, Southwest directly infringes the ’785 patent by testing the Southwest Systems and Services and its various systems that use Kubernetes. On information and belief, Southwest, with knowledge at least since the time of filing the complaint, also indirectly infringes the ’785 Patent by inducing its employees, customers, and other third parties (such as vendors) to use Southwest Systems and Services, and provides documents that include instructions regarding how to use Southwest products and services that use Kubernetes in the Southwest Systems and Services in an infringing manner. On information and belief, Southwest supplies its employees, customers, and other third parties with access to the Southwest Systems and Services through various products and services that allows its employees, customers, and other third parties to use the claimed method. On information and belief, Southwest provides instructions on how to use its various products and services that use Kubernetes in the Southwest Systems and Services in an infringing manner, the use of which results in infringement of the ’785 Patent claims through performance of the claimed methods.

Consistent with this Court’s Standing Order Governing Proceedings (OGP), IV identifies Kubernetes as used in Southwest’s systems and offerings as the Accused Instrumentality. Information regarding how specifically Southwest implements Kubernetes, including for which products and offerings, is non-public. IV intends to obtain discovery regarding how the Kubernetes is used in Southwest products and services and will update its infringement positions once this discovery is completed.

IV does not intend this exemplary claim chart to be limiting, and IV reserves its rights to pursue other accused instrumentalities, patent claims, evidence, and infringement arguments in this case. Discovery has yet to begin, and this case is still in its initial stages. Accordingly,

¹ For the avoidance of doubt, Plaintiffs do not accuse public clouds of Southwest if those services are provided by a cloud provider with a license to Plaintiffs’ patents that covers Southwest’s activities. IV will provide relevant license agreements for cloud providers in discovery. To the extent any of these licenses are relevant to Southwest’s activities, Plaintiffs will meet and confer with Southwest about the impact of such license(s).

IV reserves the right to amend and/or supplement these contentions to the full extent allowed by the Court, including but not limited to, incorporating additional information, claims, theories, and / or accused products.

On information and belief, the Southwest Systems and Services use Kubernetes in Southwest's private cloud(s). For example, Southwest posts, or has posted, job opportunities that require familiarity with Kubernetes concepts.

See <https://www.linkedin.com/in/madhuker-daraboina-0038001a5/>, job profile of Senior DevOps/Cloud Engineer stating use of Kubernetes. (last accessed 9/24/24).

See <https://www.linkedin.com/in/abhijitroy18/>, job profile of Platform engineer stating use of Kubernetes. (last accessed 9/24/24).

See <https://www.linkedin.com/in/hammad--raza/>, job profile of senior security engineer stating use of Kubernetes. (last accessed 9/24/24).

See <https://www.linkedin.com/in/saikumar-kada-a8b884135/>, job profile of senior tech ops Engineer listing Kubernetes as a skill for Southwest position. (last accessed 9/24/24).

As another example, Southwest has stated that it is investing in cloud technology and has “moved about 50% of its technology” to the cloud and has indicated cloud migration is one of its areas of focus for 2024 and beyond. Source: <https://www.phocuswire.com/southwest-airlines-cio-tech-investment>.

On information and belief, other information confirms Southwest uses Kubernetes technology.



Top Airlines, Airports & Air Services Companies Using Kubernetes

29,575 companies using this technology

By [Kubernetes](#)

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.



Southwest Airlines

Technologies used by the company: 1,161

Source: <https://www.zoominfo.com/tech/23715/kubernetes-tech-from-transportation-airline-industry-by-revenue>.²

² All sources cited in this document were publicly accessible as of the filing date of the Complaint.

U.S. Patent No. 7,949,785 (Claim 30)

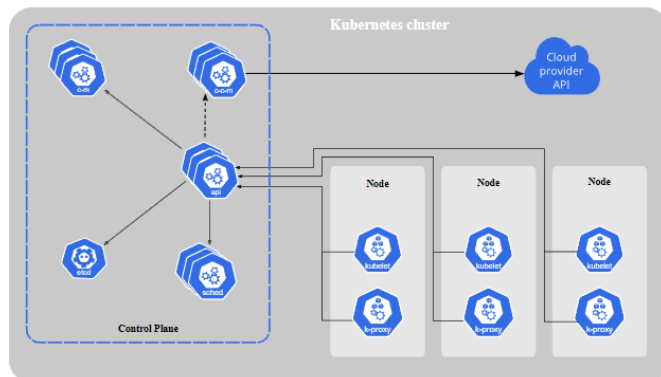
Claim(s)	Example Southwest Count 3 Systems and Services
[30.pre] A virtual network manager, comprising:	<p>To the extent this preamble is limiting, on information and belief, the Southwest Count 3 Systems and Services include a virtual network manager.</p> <p>The virtual network manager is the Kubernetes functionality related to DNS for Services and Pods.</p> <p><u>DNS for Services and Pods</u></p> <p><u>Kubernetes creates DNS records for Services and Pods.</u> You can contact Services with consistent DNS names instead of IP addresses.</p> <p>Kubernetes publishes information about Pods and Services which is used to program DNS. Kubelet configures Pods' DNS so that running containers can lookup Services by name rather than IP.</p> <p><u>Services defined in the cluster are assigned DNS names.</u> By default, a client Pod's DNS search list includes the Pod's own namespace and the cluster's default domain.</p> <p>See https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-config.</p> <p><u>A Service in Kubernetes is an abstraction which defines a logical set of Pods</u> and a policy by which to access them. Services enable a loose coupling between dependent Pods. A Service is defined using YAML or JSON, like all Kubernetes object manifests. The set of Pods targeted by a Service is usually determined by a <i>label selector</i> (see below for why you might want a Service without including a <code>selector</code> in the spec).</p> <p>See https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/.</p> <p><u>Kubernetes assigns this Service an IP address (the <i>cluster IP</i>), that is used by the virtual IP address mechanism.</u> For more details on that mechanism, read Virtual IPs and Service Proxies.</p> <p>See https://kubernetes.io/docs/concepts/services-networking/service/.</p>

U.S. Patent No. 7,949,785 (Claim 30)	
Claim(s)	Example Southwest Count 3 Systems and Services
	<p><u>Virtual IPs and Service Proxies</u></p> <p>Every node in a Kubernetes cluster runs a <code>kube-proxy</code> (unless you have deployed your own alternative component in place of <code>kube-proxy</code>).</p> <p>The <code>kube-proxy</code> component is responsible for implementing a <i>virtual IP</i> mechanism for <code>Services</code> of type other than <code>ExternalName</code>.</p> <p>See https://kubernetes.io/docs/reference/networking/virtual-ips/.</p>
[30.a] a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address;	<p>On information and belief, the Southwest Count 3 Systems and Services include a network interface configured for data communication via a virtual network that is defined by a domain name having an associated public network address.</p> <p>On information and belief, Kubernetes DNS for <code>Services</code> and pods is implemented in part through a network interface called <code>kube-proxy</code>, which maintains network rules that allow communication to pods from network sessions. <code>Kube-proxy</code> communications can relate to network sessions inside or outside of the cluster in which the services are defined. The services defined in the cluster are assigned DNS names having an associated public network address.</p>

Claim(s)

Example Southwest Count 3 Systems and Services

Kubernetes Components



kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

Source:

<https://kubernetes.io/docs/concepts/overview/components/#:~:text=kube%2Dproxy%20is%20a%20network,or%20outside%20of%20your%20cluster.>

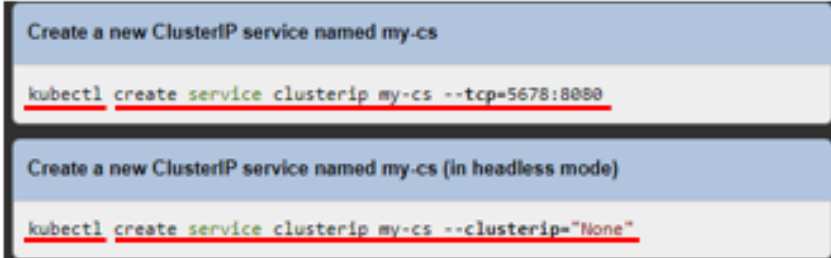
U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	<p>DNS A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.</p> <p>External IPs If there are external IPs that route to one or more cluster nodes, Kubernetes Services can be exposed on those externalIPs. When network traffic arrives into the cluster, with the external IP (as destination IP) and the port matching that Service, rules and routes that Kubernetes has configured ensure that the traffic is routed to one of the endpoints for that Service.</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/service/.</p> <p>DNS for Services and Pods Kubernetes creates DNS records for Services and Pods. You can contact Services with consistent DNS names instead of IP addresses. Services defined in the cluster are assigned DNS names. By default, a client Pod's DNS search list includes the Pod's own namespace and the cluster's default domain.</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#pod-s-dns-config.</p>
<p>[30.b] a memory and a processor to implement a register module configured to register devices in a virtual</p>	<p>On information and belief, the Southwest Count 3 Systems and Services include a memory and a processor to implement a register module configured to register devices in a virtual network.</p> <p>On information and belief, a service runs on a set of pods. The Kubernetes DNS Service module watches the Kubernetes API for incoming new services and creates a set of DNS records for each of the pods associated with the service.</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
network, the register module further configured to:	<p>Kubernetes runs your <u>workload</u> by placing containers into Pods to run on <i>Nodes</i>. A node may be a <u>virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods.</u></p> <p>Source: https://kubernetes.io/docs/concepts/architecture/nodes.</p> <p>Initializing your control-plane node</p> <p><u>The control-plane node is the machine where the control plane components run, including etcd (the cluster database) and the API Server (which the kubectl command line tool communicates with).</u></p> <p>Source: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/.</p> <p><u>DNS</u></p> <p>You can (and almost always should) set up a <u>DNS service for your Kubernetes cluster using an add-on.</u></p> <p><u>A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.</u></p> <p><u>For example, if you have a Service called my-service in a Kubernetes namespace my-ns, the control plane and the DNS Service acting together create a DNS record for my-service.my-ns. Pods in the my-ns namespace should be able to find the service by doing a name lookup for my-service. (my-service.my-ns would also work).</u></p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/service.</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	<p>Services in Kubernetes</p> <p>The Service API, part of Kubernetes, is an abstraction to help you expose groups of Pods over a network. Each Service object defines a <u>logical set of endpoints (usually these endpoints are Pods) along with a policy about how to make those pods accessible.</u></p> <p>For example, consider a stateless image-processing backend which is running with 3 replicas. Those replicas are fungible—frontends do not care which backend they use. While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that, nor should they need to keep track of the set of backends themselves.</p> <p>The Service abstraction enables this decoupling.</p> <p>The set of Pods targeted by a Service is usually determined by a <u>selector</u> that you define. To learn about other ways to define Service endpoints, see Services without selectors.</p>
[30.b.i] receive a registration request from an agent associated with a device;	<p>On information and belief, the Southwest Count 3 Systems and Services include a register module configured to receive a registration request from an agent associated with a device.</p> <p>On information and belief, a service runs on a set of pods. The Kubernetes DNS Service module watches the Kubernetes API for incoming new services being made from the kubelet.</p>  <pre>Create a new ClusterIP service named my-cs kubect1 create service clusterip my-cs --tcp=5678:8080 Create a new ClusterIP service named my-cs (in headless mode) kubect1 create service clusterip my-cs --clusterip="None"</pre> <p>Source: https://kubernetes.io/docs/reference/generated/kubect1/kubect1-commands.</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	<p data-bbox="674 315 1020 354"><u>Discovering services</u></p> <p data-bbox="674 375 1440 423">For clients running inside your cluster, Kubernetes supports two primary modes of finding a Service: environment variables and <u>DNS</u>.</p> <p data-bbox="741 431 798 456">DNS</p> <p data-bbox="741 480 1467 526">You can (and almost always should) set up a DNS service for your Kubernetes cluster using an add-on.</p> <p data-bbox="741 550 1453 646"><u>A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.</u></p> <p data-bbox="741 670 1461 766">For example, if you have a Service called <code>my-service</code> in a Kubernetes namespace <code>my-ns</code>, the control plane and the DNS Service acting together create a DNS record for <code>my-service.my-ns</code>. Pods in the <code>my-ns</code> namespace should be able to find the service by doing a name lookup for <code>my-service</code> (<code>my-service.my-ns</code> would also work).</p> <p data-bbox="674 823 1545 850">Source: https://kubernetes.io/docs/concepts/services-networking/service.</p> <p data-bbox="674 891 1083 930">Services in Kubernetes</p> <p data-bbox="674 954 1461 1034">The Service API, part of Kubernetes, is an abstraction to help you expose groups of Pods over a network. Each Service object defines a <u>logical set of endpoints (usually these endpoints are Pods) along with a policy about how to make those pods accessible.</u></p> <p data-bbox="674 1058 982 1097"><u>Defining a Service</u></p> <p data-bbox="674 1122 1444 1201">A Service is an <u>object (the same way that a Pod or a ConfigMap is an object). You can create, view or modify Service definitions using the Kubernetes API. Usually you use a tool such as <code>kubectl</code> to make those API calls for you.</u></p> <p data-bbox="674 1258 1545 1286">Source: https://kubernetes.io/docs/concepts/services-networking/service.</p>

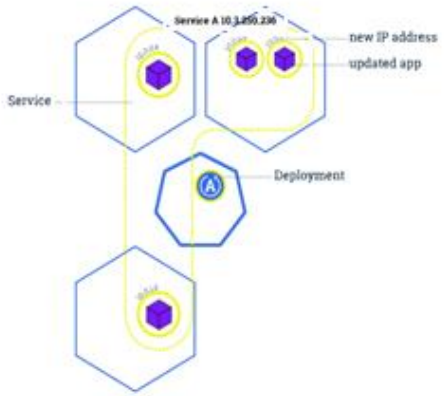
U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	<pre>kubectl logs deploy/my-deployment # dump Pod Logs for a Deployment (single-container case) kubectl logs deploy/my-deployment -c my-container # dump Pod Logs for a Deployment (multi-container case)</pre> <p>Source: https://kubernetes.io/docs/reference/kubectl/quick-reference/.</p> <p>See also https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/.</p>
[30.b.ii] distribute a virtual network address to the device when the device is registered in the virtual network, the device being identified to other devices in the virtual network by the virtual network address; and	<p>On information and belief, the Southwest Count 3 Systems and Services include a register module configured to distribute a virtual network address to the device when the device is registered in the virtual network, the device being identified to other devices in the virtual network by the virtual network address.</p> <p>On information and belief, a service runs on a set of pods. A new service is assigned to a cluster IP, and as part of configuration, the Kubernetes DNS Service module distributes the network address to at least one pod on which the service runs.</p> <p>Using a Service to Expose Your App</p> <p>Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a type in the spec of the Service:</p> <ul style="list-style-type: none">• <u>ClusterIP</u> (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster. <p>Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/.</p> <p>Service ClusterIP allocation</p> <p>In Kubernetes, <u>Services</u> are an abstract way to expose an application running on a set of Pods. <u>Services can have a cluster-scoped virtual IP address (using a Service of type: ClusterIP).</u> Clients can connect using that virtual IP address, and Kubernetes then load-balances traffic to that Service across the different backing Pods.</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	<p>Source: https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation/.</p> <p>VIP a virtual IP address, such as the one assigned to every Service in Kubernetes</p> <p>Source: https://kubernetes.io/docs/tutorials/services/source-ip/.</p> <p>DNS ⇄</p> <p>You can (and almost always should) set up a <u>DNS service for your Kubernetes cluster using an add-on</u>.</p> <p>A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.</p> <p>For example, if you have a Service called <code>my-service</code> in a Kubernetes namespace <code>my-ns</code>, the control plane and the DNS Service acting together create a DNS record for <code>my-service.my-ns</code>. Pods in the <code>my-ns</code> namespace should be able to <u>find the service</u> by doing a name lookup for <code>my-service</code> (<code>my-service.my-ns</code> would also work).</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/service.</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	 <p>Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/</p>
<p>[30.c] a DNS server for the virtual network, the DNS server configured to receive a DNS request from a first device in the virtual network and return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device.</p>	<p>On information and belief, the Southwest Count 3 Systems and Services include a DNS server for the virtual network, the DNS server configured to receive a DNS request from a first device in the virtual network and return a network address associated with a network route director, a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device.</p> <p>On information and belief, a client/ frontend pod sends a DNS query using Kubernetes DNS query functionality, such as CoreDNS or kube-dns, which includes resolver routines associated with the requesting pod. The DNS server (cluster DNS server, e.g. CoreDNS or kube-dns) is configured to return an IP address of cluster DNS server, which is referenced/returned by the process of the kubelet accessing resolv.conf.</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	<p>resolv.conf(5) File Formats Manual resolv.conf(5)</p> <p>NAME top</p> <p><u>resolv.conf - resolver configuration file</u></p> <p>SYNOPSIS top</p> <p><u>/etc/resolv.conf</u></p> <p>DESCRIPTION top</p> <p><u>The resolver is a set of routines in the C library that provide access to the Internet Domain Name System (DNS). The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file is designed to be human readable and contains a list of keywords with values that provide various types of resolver information. The configuration file is considered a trusted source of DNS information; see the trust-ad option below for details.</u></p> <p>Source: https://man7.org/linux/man-pages/man5/resolv.conf.5.html.</p> <p><u>Using CoreDNS for Service Discovery</u></p> <p>About <u>CoreDNS</u></p> <p><u>CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS.</u> Like Kubernetes, the CoreDNS project is hosted by the CNCF.</p> <p><u>You can use CoreDNS instead of kube-dns in your cluster</u> by replacing kube-dns in an existing deployment, or by using tools like kubeadm that will deploy and upgrade the cluster for you.</p> <p>Source: https://kubernetes.io/docs/tasks/administer-cluster/coredns/.</p>

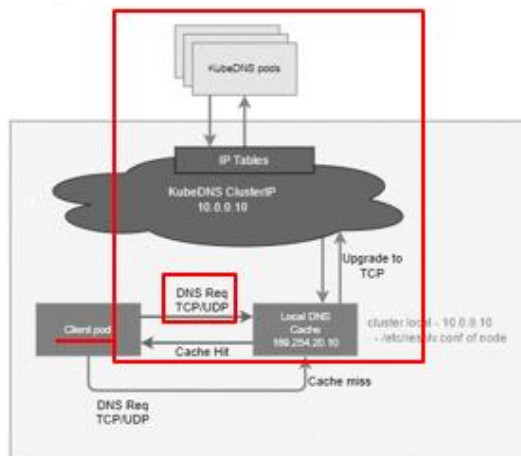
U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)

Example Southwest Count 3 Systems and Services

Introduction

NodeLocal DNSCache improves Cluster DNS performance by running a DNS caching agent on cluster nodes as a DaemonSet. In today's architecture, Pods in 'ClusterFirst' DNS mode reach out to a kube-dns serviceIP for DNS queries. This is translated to a kube-dns/CoreDNS endpoint via iptables rules added by kube-proxy. With this new architecture, Pods will reach out to the DNS caching agent running on the same node, thereby avoiding iptables DNAT rules and connection tracking. The local caching agent will query kube-dns service for cache misses of cluster hostnames ("cluster.local" suffix by default).



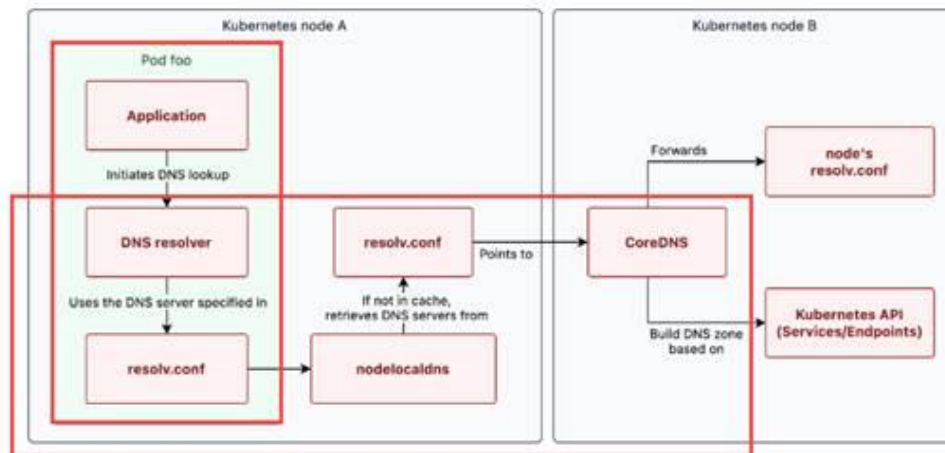
Source: <https://kubernetes.io/docs/tasks/administer-cluster/nodelocaldns/>.

U.S. Patent No. 7,949,785 (Claim 30)	
Claim(s)	Example Southwest Count 3 Systems and Services
	<p>Pod foo</p> <p>When a pod sends an API request to a service within the same Kubernetes cluster, it must first resolve the IP address of the service. To do this, the pod performs a DNS lookup using the DNS server specified in its <code>/etc/resolv.conf</code> configuration file.</p> <p>This file, which is provisioned by the Kubelet, defines the settings for DNS lookups in the pod. It contains a reference to the cluster DNS server.</p> <p>By default, this configuration file looks something like this:</p> <pre> search namespace.svc.cluster.local svc.cluster.local cluster.local nameserver 10.123.0.10 options ndots:5 </pre> <p>Source: https://www.nslookup.io/learning/the-life-of-a-dns-query-in-kubernetes/.</p>

Claim(s)

Example Southwest Count 3 Systems and Services

DNS lookups on services

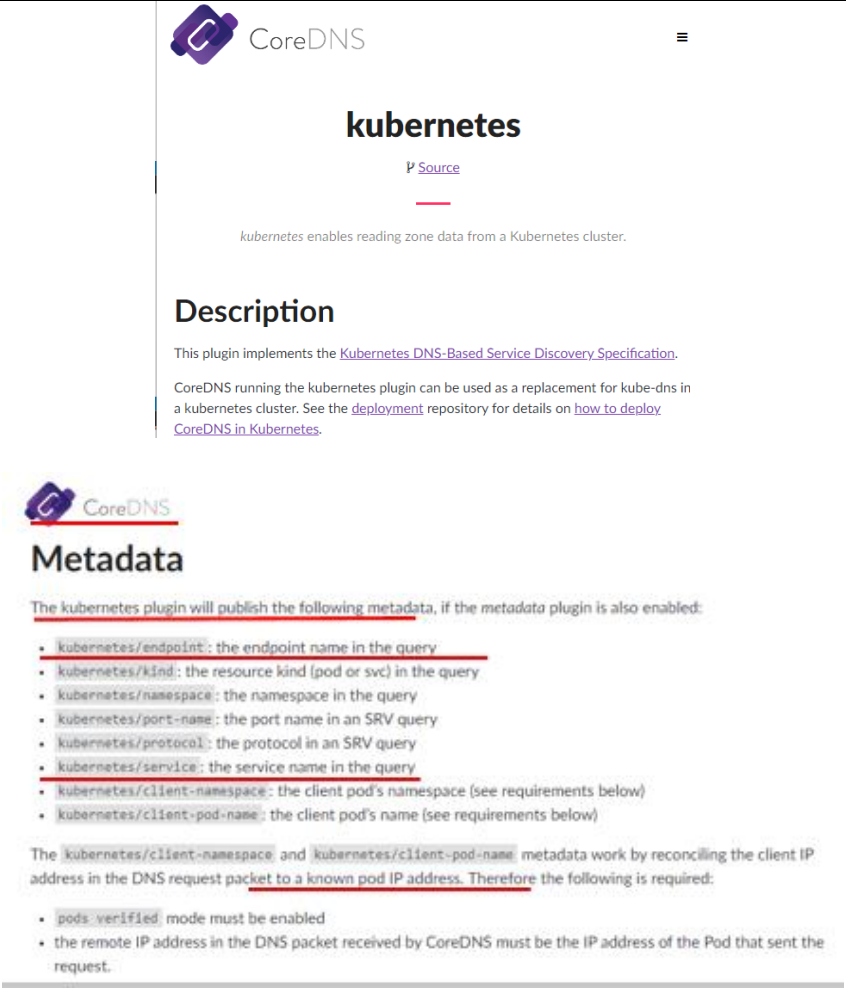


The flow of a DNS query in Kubernetes. By Nslookup.io. Licensed under CC By 4.0.

Source: <https://www.nslookup.io/learning/the-life-of-a-dns-query-in-kubernetes/>.

U.S. Patent No. 7,949,785 (Claim 30)	
Claim(s)	Example Southwest Count 3 Systems and Services
	<p>When a pod performs a DNS lookup, the query is first sent to the local DNS resolver in the pod. This resolver uses the resolv.conf configuration file. In this file, the nodelocaldns server is set up as the default recursive DNS resolver, which acts as a cache.</p> <p>If this cache does not contain the IP address for the requested hostname, the query is forwarded to the cluster DNS server (CoreDNS).</p> <p>This DNS server determines the IP address by consulting the Kubernetes service registry. This registry contains a mapping of service names to their corresponding IP addresses. This allows the cluster DNS server to return the correct IP address to the requesting pod.</p> <p>Any domains that are queried but are not in the Kubernetes service registry are forwarded to an upstream DNS server.</p> <p>We will go through each of these components in more detail step-by-step.</p> <p>Source: https://www.nslookup.io/learning/the-life-of-a-dns-query-in-kubernetes/.</p> <p>On information and belief, Southwest’s systems include “a private network address associated with a second device in the virtual network, and a virtual network address associated with the second device.”</p> <p>On information and belief, the private network address associated with a second device is a unique private IP address assigned to a backend node. The virtual network address associated with the second device is the Cluster IP or Virtual IP assigned to a pod (or pods) and/or container (or containers).</p>

U.S. Patent No. 7,949,785 (Claim 30)

Claim(s)	Example Southwest Count 3 Systems and Services
	 <p>CoreDNS</p> <h2>kubernetes</h2> <p>Source</p> <p><i>kubernetes enables reading zone data from a Kubernetes cluster.</i></p> <h3>Description</h3> <p>This plugin implements the Kubernetes DNS-Based Service Discovery Specification.</p> <p>CoreDNS running the kubernetes plugin can be used as a replacement for kube-dns in a kubernetes cluster. See the deployment repository for details on how to deploy CoreDNS in Kubernetes.</p> <h3>Metadata</h3> <p>The kubernetes plugin will publish the following metadata, if the metadata plugin is also enabled:</p> <ul style="list-style-type: none">• <code>kubernetes/endpoint</code>: the endpoint name in the query• <code>kubernetes/kind</code>: the resource kind (pod or svc) in the query• <code>kubernetes/namespace</code>: the namespace in the query• <code>kubernetes/port-name</code>: the port name in an SRV query• <code>kubernetes/protocol</code>: the protocol in an SRV query• <code>kubernetes/service</code>: the service name in the query• <code>kubernetes/client-namespace</code>: the client pod's namespace (see requirements below)• <code>kubernetes/client-pod-name</code>: the client pod's name (see requirements below) <p>The <code>kubernetes/client-namespace</code> and <code>kubernetes/client-pod-name</code> metadata work by reconciling the client IP address in the DNS request packet to a known pod IP address. Therefore the following is required:</p> <ul style="list-style-type: none">• <code>pods:verified</code> mode must be enabled• the remote IP address in the DNS packet received by CoreDNS must be the IP address of the Pod that sent the request.

Source: <https://coredns.io/plugins/kubernetes/>.

U.S. Patent No. 7,949,785 (Claim 30)	
Claim(s)	Example Southwest Count 3 Systems and Services
	<p>Connecting Applications with Services</p> <p>Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on. <u>Kubernetes gives every pod its own cluster-private IP address</u>, so you do not need to explicitly create links between pods or map container ports to host ports. This means that containers within a Pod can all reach each other's ports on localhost, and all pods in a cluster can see each other without NAT. The rest of this document elaborates on how you can run reliable services on such a networking model.</p> <p>Source: https://kubernetes.io/docs/tutorials/services/connect-applications-service/.</p> <p>Using a Service to Expose Your App</p> <p>Although <u>each Pod has a unique IP address</u>, those IPs are not exposed outside the cluster <u>without a Service</u>. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a <code>type</code> in the <code>spec</code> of the Service:</p> <ul style="list-style-type: none"> • <u>ClusterIP</u> (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster. <p>Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/.</p> <p>Service ClusterIP allocation</p> <p>In Kubernetes, <u>Services are an abstract way to expose an application running on a set of Pods</u>. <u>Services can have a cluster-scoped virtual IP address</u> (using a Service of <code>type: ClusterIP</code>). Clients can connect <u>using that virtual IP address</u>, and Kubernetes then load-balances traffic to that Service across the different backing Pods.</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation.</p>

U.S. Patent No. 7,949,785 (Claim 35)	
Claim 35	Example Southwest Count 3 Systems and Services
<p>[35] The virtual network manager of claim 30 further comprising a join module configured to receive a join request from the agent associated with the device to indicate that the device is connected for data communication within the virtual network, the join module further configured to receive a leave request from the agent associated with the device to indicate that the device will be disconnected from data communication within the virtual network.</p>	<p>On information and belief, the Southwest Count 3 Systems and Services practice the system of claim 30. <i>See</i> claim 30. On information and belief, the Southwest Count 3 Systems and Services further include a join module configured to receive a join request from the agent associated with the device to indicate that the device is connected for data communication within the virtual network, the join module further configured to receive a leave request from the agent associated with the device to indicate that the device will be disconnected from data communication within the virtual network.</p> <p>For example, on information and belief, a kubelet includes a delete pods function or process. As explained elsewhere, the kubelet (acting as the agent) on the unresponsive node starts responding and kills the pod and removes the entry from the API server.</p> <h2 style="text-align: center;">kubelet</h2> <h3 style="text-align: center;">Synopsis</h3> <p>The kubelet is the primary "node agent" that runs on each node. It can register the node with the apiserver using one of: the hostname; a flag to override the hostname; or specific logic for a cloud provider.</p> <p>The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.</p> <p>Source: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/.</p>

U.S. Patent No. 7,949,785 (Claim 35)	
Claim 35	Example Southwest Count 3 Systems and Services
	<p>A Pod is not deleted automatically when a node is unreachable. The Pods running on an unreachable Node enter the 'Terminating' or 'Unknown' state after a timeout. Pods may also enter these states when the user attempts graceful deletion of a Pod on an unreachable Node. The only ways in which a Pod in such a state can be removed from the apiserver are as follows:</p> <ul style="list-style-type: none"> • The Node object is deleted (either by you, or by the Node Controller). • The kubelet on the unresponsive Node starts responding, kills the Pod and removes the entry from the apiserver. • Force deletion of the Pod by the user. <p>The recommended best practice is to use the first or second approach. If a Node is confirmed to be dead (e.g. permanently disconnected from the network, powered down, etc), then delete the Node object. If the Node is suffering from a network partition, then try to resolve this or wait for it to resolve. When the partition heals, the kubelet will complete the deletion of the Pod and free up its name in the apiserver.</p> <p>Source: https://kubernetes.io/docs/tasks/run-application/force-delete-stateful-set-pod/.</p> <h3>DNS</h3> <p>You can (and almost always should) set up a DNS service for your Kubernetes cluster using an add-on.</p> <p>A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/service/.</p>

U.S. Patent No. 7,949,785 (Claim 37)	
Claim 37	Example Southwest Count 3 Systems and Services
<p>[37] The virtual network manager of claim 35 wherein the join module is further configured to maintain data to associate a virtual network address with a device in the virtual network.</p>	<p>On information and belief, the Southwest Count 3 Systems and Services practice the system of claim 35. <i>See</i> claim 35. On information and belief, the Southwest Count 3 Systems and Services further include the join module that is further configured to maintain data to associate a virtual network address with a device in the virtual network.</p> <p>For example, the kubernetes DNS service module includes defining a service, and within that service the join module is configured to maintain data to associate by allowing creating ,viewing, or modifying service definitions and watching kubernetes APIs.</p> <h3 style="text-align: center;">Defining a Service</h3> <p>A Service is an object (the same way that a Pod or a ConfigMap is an object). You can create, view or modify Service definitions using the Kubernetes API. Usually you use a tool such as kubectl to make those API calls for you.</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/service/.</p>

U.S. Patent No. 7,949,785 (Claim 37)

Claim 37	Example Southwest Count 3 Systems and Services
	<p>DNS</p> <p>You can (and almost always should) set up a DNS service for your Kubernetes cluster using an add-on.</p> <p>A cluster-aware DNS server, such as CoreDNS, watches the Kubernetes API for new Services and creates a set of DNS records for each one. If DNS has been enabled throughout your cluster then all Pods should automatically be able to resolve Services by their DNS name.</p> <p>For example, if you have a Service called <code>my-service</code> in a Kubernetes namespace <code>my-ns</code>, the control plane and the DNS Service acting together create a DNS record for <code>my-service.my-ns</code>. Pods in the <code>my-ns</code> namespace should be able to find the service by doing a name lookup for <code>my-service</code> (<code>my-service.my-ns</code> would also work).</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/service/.</p> <h2>Service ClusterIP allocation</h2> <p>In Kubernetes, Services are an abstract way to expose an application running on a set of Pods. Services can have a cluster-scoped virtual IP address (using a Service of <code>type: ClusterIP</code>). Clients can connect using that virtual IP address, and Kubernetes then load-balances traffic to that Service across the different backing Pods.</p> <p>Source: https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation/.</p>