

Appendix C: Claim Chart for U.S. Patent 7,640,332

Based on information presently available, Valtrus Innovations Ltd. and Key Patent Innovations Ltd. (collectively, “Valtrus” or “Plaintiffs”) contend that FedEx Corporation and FedEx Office and Print Services, Inc. (collectively, “FedEx”) infringe at least Claims 1-8 (the “Asserted Claims”) of U.S. Patent 7,640,332 (the “’332 Patent”) through the ’332 Accused Products which are manufactured, sold, offered for sale, and/or used by FedEx.

Terms not defined in this Appendix are defined in the document entitled “Plaintiffs’ Disclosure of Asserted Claims and Infringement Contentions.” The ’332 Accused Products set forth in this Appendix include, but are not limited to, Kubernetes, alone or in combination with other software, applications, among other products, and Kubernetes sold, imported, or used in the United States since 2019 by FedEx customers.

FedEx directly infringes each of the Asserted Claims by importing, using, selling, and/or offering to sell the ’332 Accused Products in violation of 35 U.S.C. § 271(a). By way of example, and as set forth in Valtrus’ Complaint, FedEx’s employees utilize Kubernetes alone or in combination with one or more FedEx software applications as set forth in this chart. Valtrus incorporates by reference all of the evidence set forth in the allegations in its Complaint.

FedEx indirectly infringes the Asserted Claims in violation of 35 U.S.C. § 271(b) by inducing third parties, including FedEx customers and end-users, to directly infringe through their operation and use of the ’332 Accused Products. FedEx has knowingly and intentionally induced this direct infringement by, *inter alia*, (i) selling, importing, or otherwise providing the ’332 Accused Products to third parties with the intent that the ’332 Accused Products will operate and be used in a manner that practices the Asserted Claims, and (ii) marketing and advertising the ’332 Accused Products. FedEx’s marketing and promotional materials for the ’332 Accused Products are found, for example, on FedEx’s website. For example, FedEx’s website offers customers with instruction and support for the ’332 Accused Products that direct customers to, among other things, set up and use the ’332 Accused Products.¹ On information and belief, FedEx knows that its actions will result in infringement of the Asserted Claims, or subjectively believes that there is a high probability that its actions will result in infringement of the Asserted Claims but has taken deliberate actions to avoid learning these facts.

The contentions below list certain features of Kubernetes that Valtrus contends are representative of features of the ’332 Accused Products. These contentions are representative of the ’332 Accused Products which include, but are not limited to, Kubernetes, alone or in combination with related software, applications, and programs.

¹ See <https://www.fedex.com/en-us/dataworks.html#:~:text=Smarter%20supply%20chains%20for%20everyone,and%20visibility%20into%20their%20business;https://www.fedex.com/en-id/business-insights/tech-innovation/how-big-data-is-delivering-the-future.html;andhttps://developer.fedex.com/api/en-us/home.htmlhttps://developer.fedex.com/api/en-us/home.html>.

Appendix C: Claim Chart for U.S. Patent 7,640,332

Moreover, the citations to certain documents and other information below are intended to be exemplary only and in no way foreclose Valtrus from citing or relying on additional documents, source code, information, and/or testimony at a later time. These contentions are preliminary in nature, and an analysis of the '332 Accused Products, internal documentation, source code, and/or testimony from relevant witnesses may be necessary to more fully and accurately describe the infringing features of the '332 Accused Products. Accordingly, Valtrus reserves the right to supplement, correct, modify, and/or amend these contentions once such additional information is made available to Valtrus. Furthermore, Valtrus reserves the right to supplement, correct, modify, and/or amend these contentions as discovery in this case progresses in view of the Court's claim construction order(s), in view of any positions taken by FedEx including, but not limited to, positions on claim construction, invalidity, and/or non-infringement, and/or in connection with the preparation and exchange of expert reports.

The contents of every below claim cell on which another claim cell depends are expressly incorporated by reference in that dependent cell, as if set forth in their entirety therein. To the extent FedEx produces source code over the course of this litigation, Valtrus expressly reserves the right to further amend these contentions to include the later produced source code, pursuant to P.R. 3-1.

To the extent these contentions include references to entities or persons other than FedEx, or to the extent FedEx contends that any element of the '332 Accused Products is attributable to a third party, Valtrus contends that they are attributable to FedEx such that they constitute direct infringement by FedEx under 271(a). The acts may be attributable to FedEx because FedEx either directs or controls the others' performance, or FedEx and the other entity form a joint enterprise. *Akamai Techs., Inc. v. Limelight Networks, Inc.*, 797 F.3d 1020, 1022 (Fed. Cir. 2015). Additionally, FedEx may be vicariously liable for the activities of these other entities. *Centillion Data Sys., LLC v. Qwest Commc'ns Int'l, Inc.*, 631 F.3d 1279, 1286 (Fed. Cir. 2011). Further, the activities of these other entities (including consumers or end-users), may be attributable to FedEx because FedEx (1) conditions participation in an activity or receipt of a benefit upon others' performance of one or more steps of a patented method, and (2) establishes the manner or timing of that performance. *Eli Lilly & Co. v. Teva Parenteral Meds., Inc.*, 845 F.3d 1357, 1365 (Fed. Cir. 2017). These acts also may be attributed to FedEx because, for example, it initiated the activities of its end-users. *SiRF Tech., Inc. v. Int'l Trade Comm'n*, 601 F.3d 1319, 1330 (Fed. Cir. 2010).

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
1[Pre]. A method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes, comprising:	The '332 Accused Products infringe directly and/or indirectly by performing and/or inducing others to perform: a method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes. This element is met literally, or in the alternative, under the doctrine of equivalents. For example, Kubernetes uses a method of rolling updates to allow for hot deployment and redeployment (rollouts and rollbacks). Using rolling updates, application containers (e.g., pods) are updated and scheduled on worker nodes of the cluster. Kubernetes progressively rolls out changes to your application or its configuration, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions. Source: https://kubernetes.io/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Core Components A Kubernetes cluster consists of a control plane and one or more worker nodes. Here's a brief overview of the main components: Control Plane Components Manage the overall state of the cluster: kube-apiserver The core component server that exposes the Kubernetes HTTP API etcd Consistent and highly-available key value store for all API server data kube-scheduler Looks for Pods not yet bound to a node, and assigns each Pod to a suitable node. kube-controller-manager Runs <u>controllers</u> to implement Kubernetes API behavior. cloud-controller-manager (optional) Integrates with underlying cloud provider(s). Source: https://kubernetes.io/docs/concepts/overview/components/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Node Components Run on every node, maintaining running pods and providing the Kubernetes runtime environment: kubelet Ensures that Pods are running, including their containers. kube-proxy (optional) Maintains network rules on nodes to implement <u>Services</u>. Container runtime Software responsible for running containers. Read Container Runtimes to learn more. Source: https://kubernetes.io/docs/concepts/overview/components/ Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A rolling update allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Valtrus reserves its right to identify source code under Local Rule 3-1once access to relevant source code is provided.
1[a]. adding a new version of an application release bundle in a repository server;	Each '332 Accused Product performs the step of adding a new version of an application release bundle in a repository server. This element is met literally, or in the alternative, under the doctrine of equivalents. For example, container images are software bundles shared in a registry. To roll out a new version of software, a container image with the new version is first formed. Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run. You can change that information later by updating your Deployment; Modules 5 and 6 of the bootcamp discuss how you can scale and update your Deployments. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>. Source: https://kubernetes.io/docs/concepts/containers/images/ If you don't specify a registry hostname, Kubernetes assumes that you mean the Docker public registry. You can change this behaviour by setting default image registry in container runtime configuration. Source: https://kubernetes.io/docs/concepts/containers/images/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
1[b]. determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;	Each '332 Accused Product performs the step of determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server. This element is met literally, or in the alternative, under the doctrine of equivalents. For example, a control plane has a deployment controller that ensures that the new container image is deployed all across the ReplicaSet of the cluster.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Updating an application Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/ Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A rolling update allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/ Initially, the pods run the first version of your application—let's suppose its image is tagged as <code>v1</code>. You then develop a newer version of the app and push it to an image repository as a new image, tagged as <code>v2</code>. You'd next like to replace all the pods with this new version. Because you can't change an existing pod's image after the pod is created, you need to remove the old pods and replace them with new ones running the new image. Source: https://livebook.manning.com/book/kubernetes-in-action/chapter-9/13

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Deployments A Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state. A <i>Deployment</i> provides declarative updates for <u>Pods</u> and <u>ReplicaSets</u>. You describe a <i>desired state</i> in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ Each time a new Deployment is observed by the Deployment controller, a ReplicaSet is created to bring up the desired Pods. If the Deployment is updated, the existing ReplicaSet that controls Pods whose labels match <code>.spec.selector</code> but whose template does not match <code>.spec.template</code> are scaled down. Eventually, the new ReplicaSet is scaled to <code>.spec.replicas</code> and all old ReplicaSets is scaled to 0. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

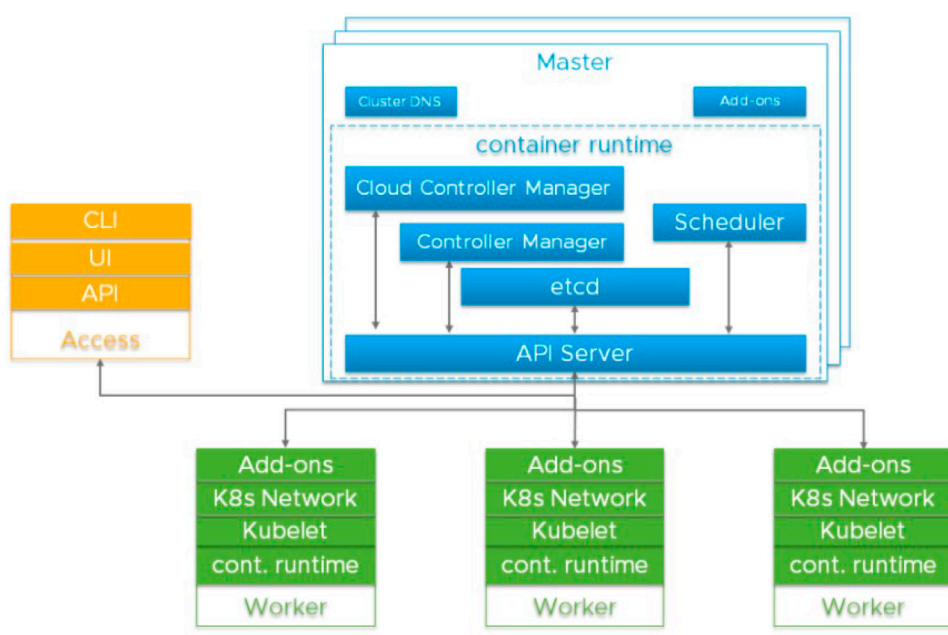
Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster. Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. This provides a self-healing mechanism to address machine failure or maintenance. In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to application management. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
1[c]. notifying a client application manager associated with one or more of the determined grid	Each '332 Accused Product performs the step of notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and	This element is met literally, or in the alternative, under the doctrine of equivalents. For example, the Deployment controller notifies Kubelet processes in worker nodes. Each Kubelet process is therefore notified of the desired configuration including the new application and associated data transfer protocol information. Each time a new Deployment is observed by the Deployment controller, a ReplicaSet is created to bring up the desired Pods. If the Deployment is updated, the existing ReplicaSet that controls Pods whose labels match <code>.spec.selector</code> but whose template does not match <code>.spec.template</code> are scaled down. Eventually, the new ReplicaSet is scaled to <code>.spec.replicas</code> and all old ReplicaSets is scaled to 0. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ 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. 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. Source: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	 The diagram illustrates a Kubernetes architecture. On the left, a stack of orange boxes represents the user interface and access: CLI, UI, API, and Access. In the center, a stack of blue boxes represents the Master node, which includes Cluster DNS, Add-ons, a container runtime (dashed box), Cloud Controller Manager, Controller Manager, Scheduler, etcd, and the API Server. On the right, three stacks of green boxes represent Worker nodes, each containing Add-ons, K8s Network, Kubelet, cont. runtime, and the Worker label. Arrows indicate the flow of communication: from the CLI/UI/API/Access stack to the Master node's API Server, and from the Master node's API Server to the Worker nodes' Kubelets. Source: https://kubernetes.io/blog/2018/08/03/out-of-the-clouds-onto-the-ground-how-to-make-kubernetes-production-grade-anywhere/

Appendix C: Claim Chart for U.S. Patent 7,640,332

	<pre> Name: nginx-deployment Namespace: default CreationTimestamp: Thu, 30 Nov 2017 10:56:25 +0000 Labels: app=nginx Annotations: deployment.kubernetes.io/revision=2 Selector: app=nginx Replicas: 3 desired 3 updated 3 total 3 available 0 unavailable StrategyType: RollingUpdate MinReadySeconds: 0 RollingUpdateStrategy: 25% max unavailable, 25% max surge Pod Template: Labels: app=nginx Containers: nginx: Image: nginx:1.16.1 Port: 80/TCP Environment: <none> Mounts: <none> Volumes: <none> Conditions: Type Status Reason ---- - Available True MinimumReplicasAvailable Progressing True NewReplicaSetAvailable OldReplicaSets: <none> NewReplicaSet: nginx-deployment-1564180365 (3/3 replicas created) Events: Type Reason Age From Message ---- - Normal ScalingReplicaSet 2m deployment-controller Scaled up replica set nginx-deployment-2 Normal ScalingReplicaSet 24s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 22s deployment-controller Scaled down replica set nginx-deployment-1 Normal ScalingReplicaSet 22s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 19s deployment-controller Scaled down replica set nginx-deployment-1 Normal ScalingReplicaSet 19s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 14s deployment-controller Scaled down replica set nginx-deployment-1 </pre>
--	--

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
1[d]. hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.	Each '332 Accused Product performs the step of hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers. This element is met literally, or in the alternative, under the doctrine of equivalents. For example, rolling updates of the new software are made with the rollout command of the kubectl interface. The image is deployed in the worker nodes according to the desired configuration by the Kubelet process.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Updating an application <i>Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones.</i> Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A rolling update allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Deployments A Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state. A <i>Deployment</i> provides declarative updates for <u>Pods</u> and <u>ReplicaSets</u>. You describe a <i>desired state</i> in a Deployment, and the Deployment <u>Controller</u> changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ • Declare the new state of the Pods by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new ReplicaSet updates the revision of the Deployment. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	• The <code>.spec.template</code> field contains the following sub-fields: ◦ The Pods are labeled <code>app: nginx</code> using the <code>.metadata.labels</code> field. ◦ The Pod template's specification, or <code>.spec</code> field, indicates that the Pods run one container, <code>nginx</code>, which runs the <code>nginx</code> Docker Hub image at version 1.14.2. ◦ Create one container and name it <code>nginx</code> using the <code>.spec.containers[0].name</code> field. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT										
	2. To see the rollout status, run: <pre>kubectl rollout status deployment/nginx-deployment</pre> The output is similar to this: <pre>Waiting for rollout to finish: 2 out of 3 new replicas have been updated...</pre> or <pre>deployment "nginx-deployment" successfully rolled out</pre> Get more details on your updated Deployment: After the rollout succeeds, you can view the Deployment by running <code>kubectl get deployments</code> . The output is similar to this: <table><tr><th>NAME</th><th>READY</th><th>UP-TO-DATE</th><th>AVAILABLE</th><th>AGE</th></tr><tr><td>nginx-deployment</td><td>3/3</td><td>3</td><td>3</td><td>36s</td></tr></table>	NAME	READY	UP-TO-DATE	AVAILABLE	AGE	nginx-deployment	3/3	3	3	36s
NAME	READY	UP-TO-DATE	AVAILABLE	AGE							
nginx-deployment	3/3	3	3	36s							

Source: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT															
	The output is similar to this: <table><tr><th>NAME</th><th>DESIRED</th><th>CURRENT</th><th>READY</th><th>AGE</th></tr><tr><td>nginx-deployment-1564180365</td><td>3</td><td>3</td><td>3</td><td>6s</td></tr><tr><td>nginx-deployment-2035384211</td><td>0</td><td>0</td><td>0</td><td>36s</td></tr></table> Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/	NAME	DESIRED	CURRENT	READY	AGE	nginx-deployment-1564180365	3	3	3	6s	nginx-deployment-2035384211	0	0	0	36s
NAME	DESIRED	CURRENT	READY	AGE												
nginx-deployment-1564180365	3	3	3	6s												
nginx-deployment-2035384211	0	0	0	36s												

Appendix C: Claim Chart for U.S. Patent 7,640,332

	<pre> Name: nginx-deployment Namespace: default CreationTimestamp: Thu, 30 Nov 2017 10:56:25 +0000 Labels: app=nginx Annotations: deployment.kubernetes.io/revision=2 Selector: app=nginx Replicas: 3 desired 3 updated 3 total 3 available 0 unavailable StrategyType: RollingUpdate MinReadySeconds: 0 RollingUpdateStrategy: 25% max unavailable, 25% max surge Pod Template: Labels: app=nginx Containers: nginx: Image: nginx:1.16.1 Port: 80/TCP Environment: <none> Mounts: <none> Volumes: <none> Conditions: Type Status Reason ---- - Available True MinimumReplicasAvailable Progressing True NewReplicaSetAvailable OldReplicaSets: <none> NewReplicaSet: nginx-deployment-1564180365 (3/3 replicas created) Events: Type Reason Age From Message ---- - Normal ScalingReplicaSet 2m deployment-controller Scaled up replica set nginx-deployment-2 Normal ScalingReplicaSet 24s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 22s deployment-controller Scaled down replica set nginx-deployment-1 Normal ScalingReplicaSet 22s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 19s deployment-controller Scaled down replica set nginx-deployment-1 Normal ScalingReplicaSet 19s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 14s deployment-controller Scaled down replica set nginx-deployment-1 </pre>	
--	---	--

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
2. The method of claim 1, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises: selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and	Each '332 Accused Product performs the method of Claim 1, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises: selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1[d].</i> For example, selecting and invoking appropriate hot deployment plug-ins is performed by ReplicaSets and is done manually by the user or automatically by the deployment process.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	How a ReplicaSet works A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria. A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number. When a ReplicaSet needs to create new Pods, it uses its Pod template. A ReplicaSet is linked to its Pods via the Pods' metadata.ownerReferences field, which specifies what resource the current object is owned by. All Pods acquired by a ReplicaSet have their owning ReplicaSet's identifying information within their ownerReferences field. It's through this link that the ReplicaSet knows of the state of the Pods it is maintaining and plans accordingly. A ReplicaSet identifies new Pods to acquire by using its selector. If there is a Pod that has no OwnerReference or the OwnerReference is not a Controller and it matches a ReplicaSet's selector, it will be immediately acquired by said ReplicaSet. Source: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/ For another example, selecting and invoking appropriate hot deployment plug-ins is performed by StatefulSets where applications require maintenance of certain network and storage states.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	StatefulSets A StatefulSet runs a group of Pods, and maintains a sticky identity for each of those Pods. This is useful for managing applications that need persistent storage or a stable, unique network identity. StatefulSet is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of <u>Pods</u>, <i>and provides guarantees about the ordering and uniqueness</i> of these Pods. Like a <u>Deployment</u>, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling. If you want to use storage volumes to provide persistence for your workload, you can use a StatefulSet as part of the solution. Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed. Source: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	For yet another example, selecting and invoking appropriate hot deployment plug-ins is performed by CronJobs which starts a one-time Job on a repeating schedule. CronJob A CronJob starts one-time Jobs on a repeating schedule. Source: https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Jobs Jobs represent one-off tasks that run to completion and then stop. A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created. Suspending a Job will delete its active Pods until the Job is resumed again. A simple case is to create one Job object in order to reliably run one Pod to completion. The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot). You can also use a Job to run multiple Pods in parallel. If you want to run a Job (either a single task, or several in parallel) on a schedule, see CronJob. Source: https://kubernetes.io/docs/concepts/workloads/controllers/job/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
2[a]. hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.	Each '332 Accused Product performs the method of Claim 1, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1[d].</i> For example, through the Deployment Controller, the user selects the appropriate node on which each respective pod and Kubelet process are located. Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
3. The method of claim 1, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot	Each '332 Accused Product performs the method of Claim 1, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises: hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1[d].</i> For example, the deployment controller transfers data to the Kubelet processes in worker nodes. Each Kubelet process, therefore, receives data for the desired configuration, including the new application and associated protocol information through its interface services.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
deployment plug-in by a data transfer client associated with a respective one of the client application managers.	Deployments A Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state. A <i>Deployment</i> provides declarative updates for <u>Pods</u> and <u>ReplicaSets</u>. You describe a <i>desired state</i> in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ Each time a new Deployment is observed by the Deployment controller, a ReplicaSet is created to bring up the desired Pods. If the Deployment is updated, the existing ReplicaSet that controls Pods whose labels match <code>.spec.selector</code> but whose template does not match <code>.spec.template</code> are scaled down. Eventually, the new ReplicaSet is scaled to <code>.spec.replicas</code> and all old ReplicaSets is scaled to 0. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	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. 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. Source: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
4. The method of claim 1, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.	Each '332 Accused Product performs the method of Claim 1, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1.</i> For example, a deployment contains container images and replicas with a set of components that are distributed and stored on nodes in the Kubernetes cluster.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run. You can change that information later by updating your Deployment; Modules 5 and 6 of the bootcamp discuss how you can scale and update your Deployments. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ Service ClusterIP allocation 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.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/concepts/services-networking/cluster-ip-allocation/ Topology Aware Routing <i>Topology Aware Routing</i> provides a mechanism to help keep network traffic within the zone where it originated. Preferring same-zone traffic between Pods in your cluster can help with reliability, performance (network latency and throughput), or cost. Source: https://kubernetes.io/docs/concepts/services-networking/topology-aware-routing/

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	The Kubernetes model for connecting containers Now that you have a continuously running, replicated application you can expose it on a network. Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on. Kubernetes gives every pod its own cluster-private IP address, 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. This tutorial uses a simple nginx web server to demonstrate the concept. Source: https://kubernetes.io/docs/tutorials/services/connect-applications-service/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
5. The method of claim 1, wherein the repository [sic]server stores one or more new versions of the application release bundle associated with the application.	Each '332 Accused Product performs the method of Claim 1, wherein the repository [sic]server stores one or more new versions of the application release bundle associated with the application. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1.</i>

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	For example, container images are stored on registries before being deployed to a pod. To roll out an update, a new container image with a new version of the application is formed. Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. Source: https://kubernetes.io/docs/concepts/containers/images/ If you don't specify a registry hostname, Kubernetes assumes that you mean the Docker public registry. You can change this behaviour by setting default image registry in container runtime configuration. Source: https://kubernetes.io/docs/concepts/containers/images/ Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes Deployment. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run. You can change that information later by updating your Deployment; Modules 5 and 6 of the bootcamp discuss how you can scale and update your Deployments. Source: https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/ By way of further example, deployments are controlled through a revision control with the capability of rolling back versions of deployment.

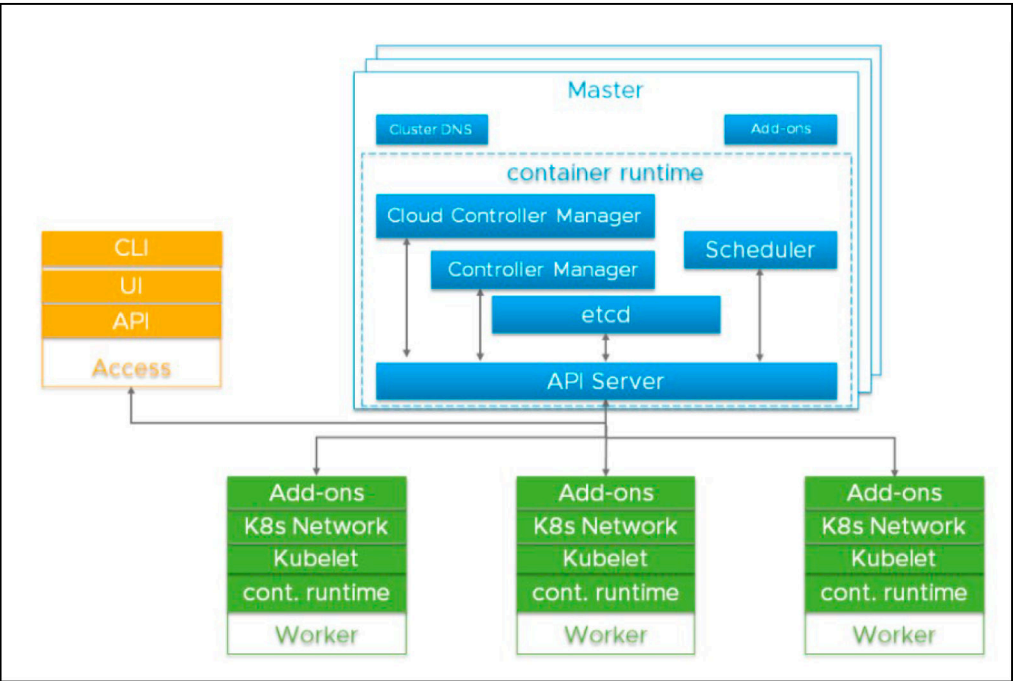
Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Rolling Back a Deployment Sometimes, you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping. By default, all of the Deployment's rollout history is kept in the system so that you can rollback anytime you want (you can change that by modifying revision history limit). Note: A Deployment's revision is created when a Deployment's rollout is triggered. This means that the new revision is created if and only if the Deployment's Pod template (<code>.spec.template</code>) is changed, for example if you update the labels or container images of the template. Other updates, such as scaling the Deployment, do not create a Deployment revision, so that you can facilitate simultaneous manual- or auto-scaling. This means that when you roll back to an earlier revision, only the Deployment's Pod template part is rolled back. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#rolling-back-a-deployment Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
6. The method of claim 1, further comprising: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.	Each '332 Accused Product performs the method of Claim 1, further comprising: registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1.</i> For example, the deployment controller is notified by the Kubelet processes in worker nodes. Each Kubelet process, therefore, notifies the deployment controller of the receipt of the new application release bundle. Each time a new Deployment is observed by the Deployment controller, a ReplicaSet is created to bring up the desired Pods. If the Deployment is updated, the existing ReplicaSet that controls Pods whose labels match <code>.spec.selector</code> but whose template does not match <code>.spec.template</code> are scaled down. Eventually, the new ReplicaSet is scaled to <code>.spec.replicas</code> and all old ReplicaSets is scaled to 0. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ 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. 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.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/  Source: https://kubernetes.io/blog/2018/08/03/out-of-the-clouds-onto-the-ground-how-to-make-kubernetes-production-grade-anywhere/

Appendix C: Claim Chart for U.S. Patent 7,640,332

	<pre> Name: nginx-deployment Namespace: default CreationTimestamp: Thu, 30 Nov 2017 10:56:25 +0000 Labels: app=nginx Annotations: deployment.kubernetes.io/revision=2 Selector: app=nginx Replicas: 3 desired 3 updated 3 total 3 available 0 unavailable StrategyType: RollingUpdate MinReadySeconds: 0 RollingUpdateStrategy: 25% max unavailable, 25% max surge Pod Template: Labels: app=nginx Containers: nginx: Image: nginx:1.16.1 Port: 80/TCP Environment: <none> Mounts: <none> Volumes: <none> Conditions: Type Status Reason ---- - Available True MinimumReplicasAvailable Progressing True NewReplicaSetAvailable OldReplicaSets: <none> NewReplicaSet: nginx-deployment-1564180365 (3/3 replicas created) Events: Type Reason Age From Message ---- - Normal ScalingReplicaSet 2m deployment-controller Scaled up replica set nginx-deployment-2 Normal ScalingReplicaSet 24s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 22s deployment-controller Scaled down replica set nginx-deployment-1 Normal ScalingReplicaSet 22s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 19s deployment-controller Scaled down replica set nginx-deployment-1 Normal ScalingReplicaSet 19s deployment-controller Scaled up replica set nginx-deployment-1 Normal ScalingReplicaSet 14s deployment-controller Scaled down replica set nginx-deployment-1 </pre>	
--	---	--

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ By way of further example, during deployment a process of proportional scaling maintains support for multiple versions of the deployed applications. Proportional scaling RollingUpdate Deployments support running multiple versions of an application at the same time. When you or an autoscaler scales a RollingUpdate Deployment that is in the middle of a rollout (either in progress or paused), the Deployment controller balances the additional replicas in the existing active ReplicaSets (ReplicaSets with Pods) in order to mitigate risk. This is called <i>proportional scaling</i>. For example, you are running a Deployment with 10 replicas, maxSurge=3, and maxUnavailable=2. Source: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
7. The method of claim 1, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.	Each '332 Accused Product performs the method of Claim 1, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp. This element is met literally, or in the alternative, under the doctrine of equivalents. <i>See supporting evidence for Claim 1.</i>

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	For example, Kubernetes supports tcp and udp protocols, and supports ftp, http, and gridftp protocols through tcp transport and proxy protocol. Kubernetes supports the following protocols with Services: • SCTP • TCP (the default) • UDP When you define a Service, you can also specify the application protocol that it uses. This document details some special cases, all of them typically using TCP as a transport protocol: • HTTP and HTTPS • PROXY protocol • TLS termination at the load balancer Source: https://kubernetes.io/docs/reference/networking/service-protocols/#protocol-sctp Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.
8. The method of claim 1, wherein each grid node comprises one or more application servers that include components selected from the group consisting of Web containers, Web service	Each '332 Accused Product performs the method of Claim 1, wherein each grid node comprises one or more application servers that include components selected from the group consisting of Web containers, Web service containers, and EJB containers. This element is met literally, or in the alternative, under the doctrine of equivalents. For example, on information and belief, Kubernetes supports provision of Services, which uses exposed network applications, such as Web containers, Web service containers, and EJB containers.

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
containers, and EJB containers.	Service Expose an application running in your cluster behind a single outward-facing endpoint, even when the workload is split across multiple backends. In Kubernetes, a Service is a method for exposing a network application that is running as one or more <u>Pods</u> in your cluster. Source: https://kubernetes.io/docs/concepts/services-networking/service/

Appendix C: Claim Chart for U.S. Patent 7,640,332

Port definitions

Port definitions in Pods have names, and you can reference these names in the `targetPort` attribute of a Service. For example, we can bind the `targetPort` of the Service to the Pod port in the following way:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app.kubernetes.io/name: proxy
spec:
  containers:
    - name: nginx
      image: nginx:stable
      ports:
        - containerPort: 80
          name: http-web-svc

---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app.kubernetes.io/name: proxy
  ports:
    - name: name-of-service-port
      protocol: TCP
      port: 80
      targetPort: http-web-svc
```

Appendix C: Claim Chart for U.S. Patent 7,640,332

CLAIM	SUPPORT
	Source: https://kubernetes.io/docs/concepts/services-networking/service/ This works even if there is a mixture of Pods in the Service using a single configured name, with the same network protocol available via different port numbers. This offers a lot of flexibility for deploying and evolving your Services. For example, you can change the port numbers that Pods expose in the next version of your backend software, without breaking clients. Source: https://kubernetes.io/docs/concepts/services-networking/service/ Valtrus reserves its right to identify source code under Local Rule 3-1 once access to relevant source code is provided.