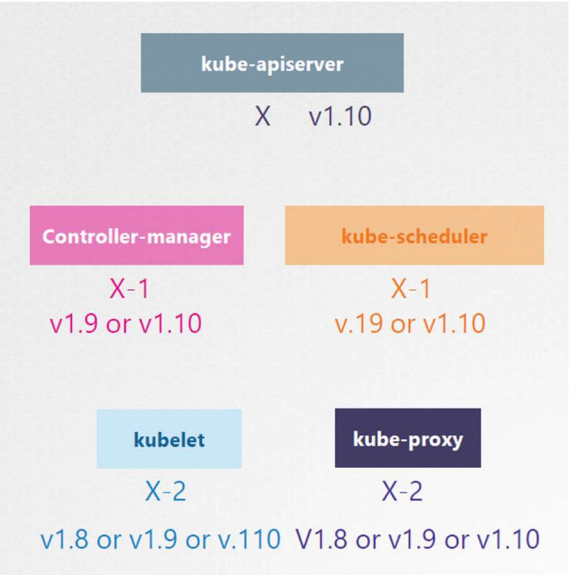
Cluster Upgrade Process

We discuss about cluster upgrade process in Kubernetes. In the previous lecture, we saw how Kubernetes manages its software releases and how different components have their versions We will keep dependency on external components like etcd and CoreDNS aside for now and focus on the core control plane components.

Is it mandatory for all of these to have the same version? No, the components can be at different release versions. Since the kube API server is the primary component in the control plane and that is the component that all other components talk to. None of the other components should ever be at a version higher than the kube API server.

The controller manager and scheduler can be at one version lower. So if kube API server was at x, controller managers and kube schedulers can be at x-1. And the kubelet and kube proxy components can be at two versions lower, x-2. So if kube API server was at 1.10, the controller manager and scheduler could be at 1.10 or 1.9, and the kubelet and kube proxy could be at 1.8. None of them could be at a version higher than the kube API server like 1.11.



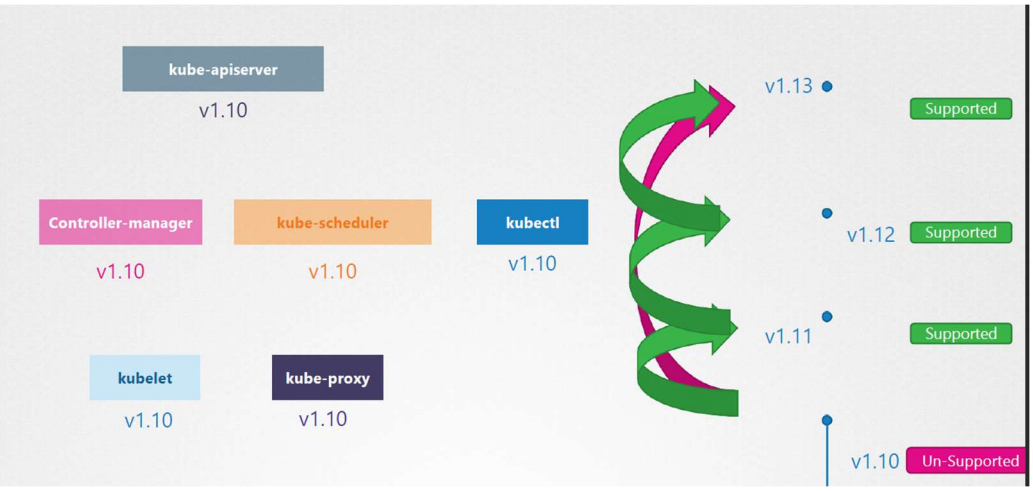
Now this is not the case with kube control, the kube control utility could be at 1.11, a version higher than the API server. 1.10, the same version as the API server. Or at 1.9, a version lower than the API server.

Now, this permissible skew in versions allows us to carry out live upgrades. We can upgrade component by component if required.

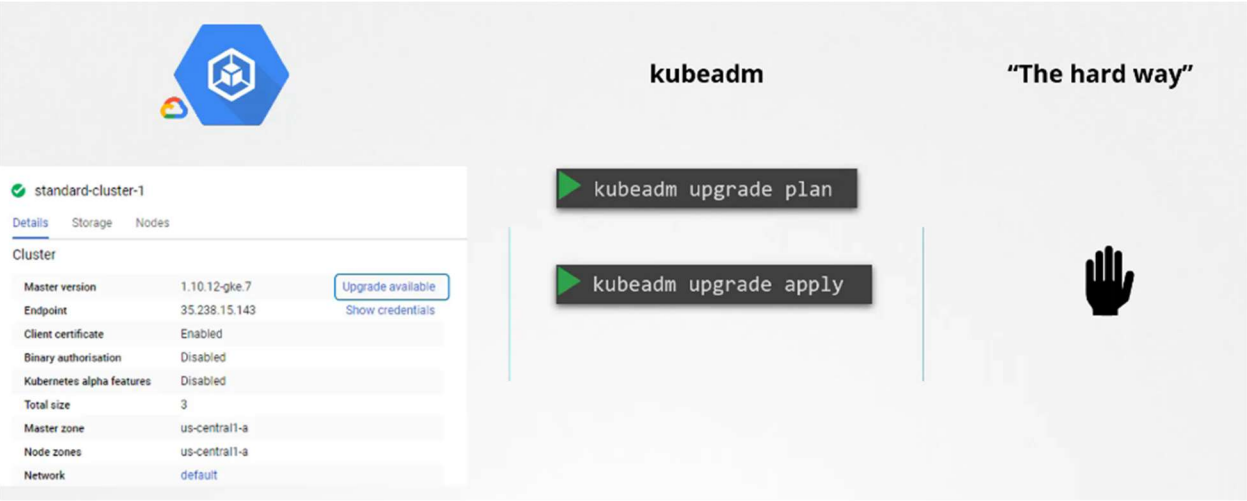
So when should you upgrade?

Say you were at 1.10 and Kubernetes releases versions 1.11 and 1.12. At any time, Kubernetes supports only up to the recent three minor versions. So with 1.12 being the latest release,

Kubernetes supports versions 1.12, 1.11, and 1.10. So when 1.13 is released, only versions 1.13, 1.12, and 1.11 are supported. Before the release of 1.13, would be a good time to upgrade your cluster to the next release.



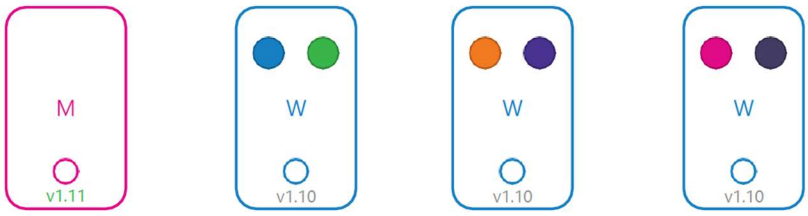
So how do we upgrade? Do we upgrade directly from 1.10 to 1.13? No, the recommended approach is to upgrade one minor version at a time, version 1.10 to 1.11, then 1.11 to 1.12, and then 1.12 to 1.13. The upgrade process depends on how your cluster is set up. For example, if your cluster is a managed Kubernetes cluster deployed on cloud service providers like Google, for instance. Google Kubernetes engine lets you upgrade your cluster easily with just a few clicks. If you deployed your cluster using tools like kubeadm, then the tool can help you plan and upgrade the cluster. If you deployed your cluster from scratch, then you manually upgrade the different components of the cluster yourself.



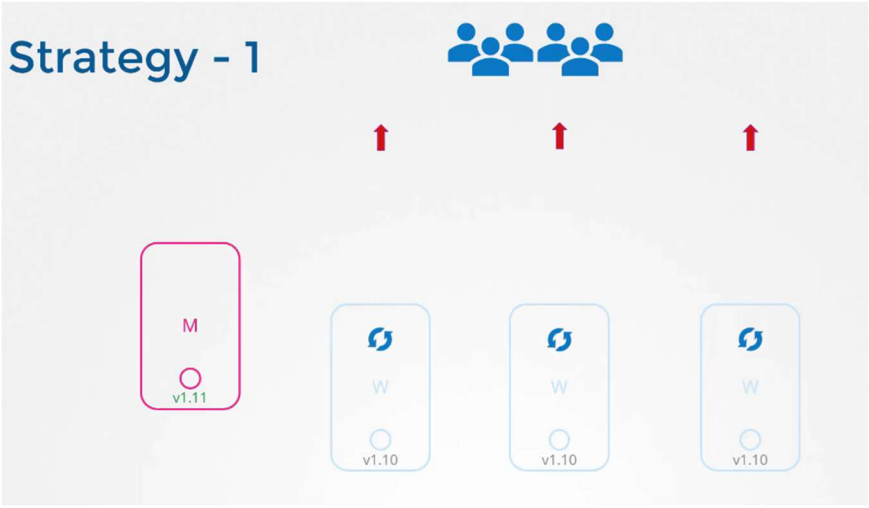
In this lecture, we will look at the options by kubeadm. So you have a cluster with master and worker nodes. running in production hosting pods, serving users. The nodes and components are at version 1.10. Upgrading a cluster involves two major steps. First, you upgrade your master nodes and then upgrade the worker nodes. While the master is being upgraded, the control plane

components such as the API server, scheduler, and controller managers go down briefly. The master going down does not mean your worker nodes and applications on the cluster are impacted. All workloads hosted on the worker nodes continue to serve users as normal, since the master is down, all management functions are down. You cannot access the cluster using kube control or other Kubernetes API. You cannot deploy new applications or delete or modify existing ones. The controller managers don't function either. If a pod was to fail, a new pod won't be automatically created. But as long as the nodes and the pods are up, your applications should be up and users will not be impacted.

Once the upgrade is complete and the cluster is back up, it should function normally. We now have the master and the master components at version 1.11 and the worker nodes at version 1.10. As we saw earlier, this is a supported configuration, it is now time to upgrade the worker nodes.

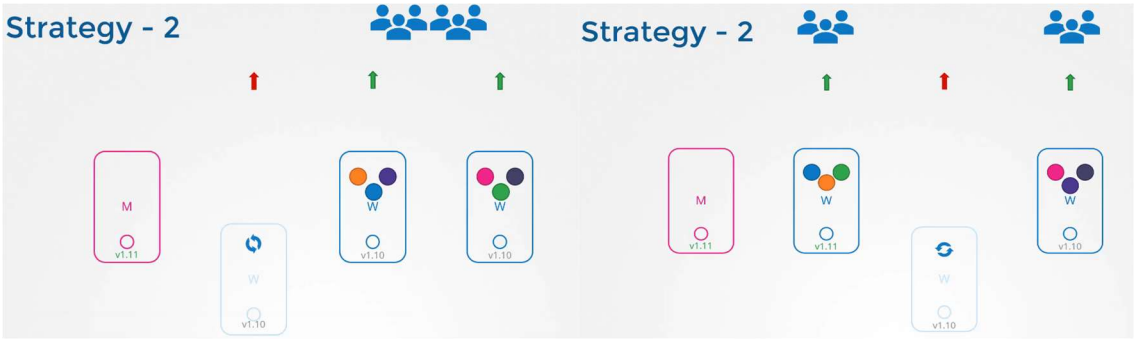


There are different strategies available to upgrade the worker nodes. One is to upgrade all of them at once, but then your pods are down and users are no longer able to access the applications. Once the upgrade is complete, the nodes are back up, new pods are scheduled, and users can resume access. That's one strategy that requires downtime.

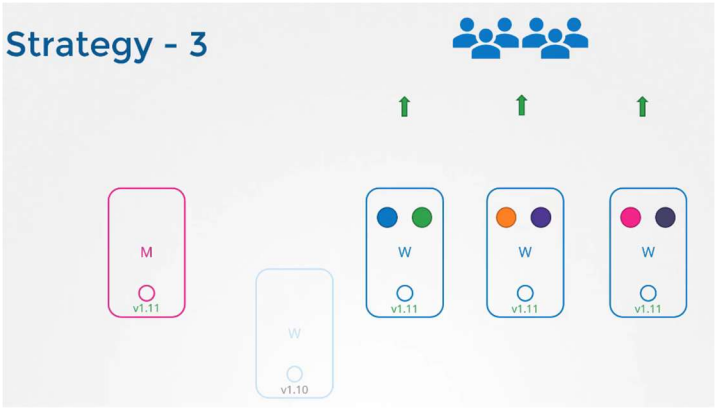


The second strategy is to upgrade one node at a time. So going back to the state where we have our master upgraded and nodes waiting to be upgraded. We first upgrade the first node where the workloads move to the second and third node and users are served from there. Once the first

node is upgraded and back up, we then update the second node where the workloads move to the first and third nodes. And finally, the third node where the workloads are shared between the first two. Until we have all nodes upgraded to a newer version, we then follow the same procedure to upgrade the nodes from 1.11 to 1.12 and then 1.13.



A third strategy would be to add new nodes to the cluster, nodes with newer software version. This is especially convenient if you're on a cloud environment where you can easily provision new nodes and decommission old ones. Nodes with the newer software version can be added to the cluster, move the workload over to the new, and remove the old node, until you finally have all new nodes with the new software version.



Let us now see how it is done. Say we were to upgrade this cluster from 1.11 to 1.13, kubeadm has an upgrade command that helps in upgrading clusters.

With kubeadm, run the kubeadm upgrade plan command and it'll give you a lot of good information. The current cluster version, the kubeadm tool version, the latest stable version of Kubernetes. Then it lists all the control plane components and their versions and what version these can be upgraded to. It also tells you that after we upgrade the control plane components, you must manually upgrade the kubelet versions on each node. Remember, kubeadm does not install or upgrade kubelets.

Finally, it gives you the command to upgrade the cluster. Also, note that you must upgrade the kubeadm tool itself before you can upgrade the cluster.

```
kubeadm upgrade plan
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.11.8
[upgrade/versions] kubeadm version: v1.11.3
[upgrade/versions] Latest stable version: v1.13.4
[upgrade/versions] Latest version in the v1.11 series: v1.11.8

Components that must be upgraded manually after you have
upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT   CURRENT       AVAILABLE
Kubelet     3 x v1.11.3   v1.13.4

Upgrade to the latest stable version:

COMPONENT            CURRENT   AVAILABLE
API Server           v1.11.8   v1.13.4
Controller Manager   v1.11.8   v1.13.4
Scheduler            v1.11.8   v1.13.4
Kube Proxy           v1.11.8   v1.13.4
CoreDNS              1.1.3     1.1.3
Etcd                 3.2.18    N/A

You can now apply the upgrade by executing the following command:

        kubeadm upgrade apply v1.13.4

Note: Before you can perform this upgrade, you have to update kubeadm to v1.13.4.
```

The kubeadm tool also follows the same software version as Kubernetes. So we are at 1.11 and we want to go to 1.13. But remember, we can only go one minor version at a time. So we first go to 1.12, first, upgrade the kubeadm tool itself to version 1.12. Then upgrade the cluster using the command from the upgrade plan output, kubeadm upgrade apply. It pulls the necessary images and upgrades the cluster components. Once complete, your control plane components are now at 1.12.

```
apt-get upgrade -y kubeadm=1.12.0-00
```

```
kubeadm upgrade apply v1.12.0
…
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.12.0". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with
upgrading your kubelets if you haven't already done so.
```

If you run the kube control get nodes command, you will still see the master node at 1.11. This is because in the output of this command, it is showing the versions of kubelets on each of these nodes registered with the API server and not the version of the API server itself.

```
kubectl get nodes
NAME      STATUS    ROLES     AGE       VERSION
master   Ready   masternode-    1d        v1.11.3
1    Ready    <none> node-2    1d        v1.11.3
Ready   <none>                  1d        v1.11.3
```

So the next step is to upgrade the kubelets. Remember, depending on your setup, you may or may not have kubelets running on your master node. In this case, the cluster deployed with kubeadm has kubelets on the master node, which are used to run the control plane components as parts on the master nodes. **When we set up a Kubernetes cluster from scratch we do not install kubelet on the master node. You will not see the master node in the output of this command in that case.**

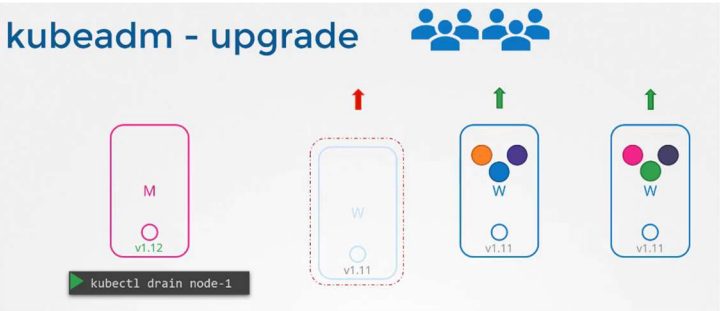
So the next step is to upgrade kubelet on the master node, if you have kubelets on them. Run the apt-get kubelet command for this. Once the package is upgraded, restart the kubelet service. Running the kube control get nodes command now shows that the master has been upgraded to 1.12. The worker nodes are still at 1.11.

```
apt-get upgrade -y kubelet=1.12.0-00
```

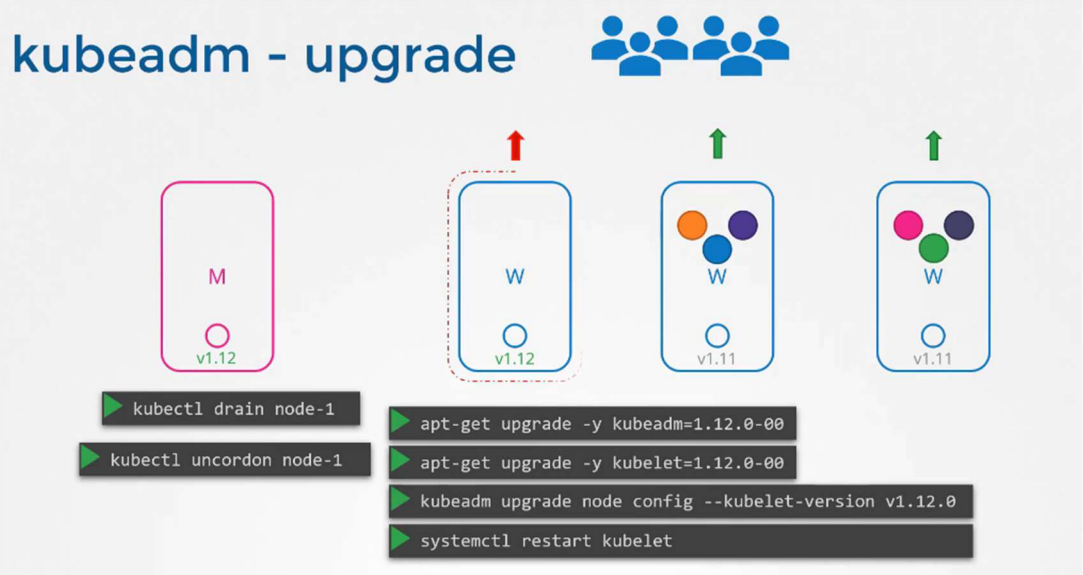
```
systemctl restart kubelet
```

```
kubectl get nodes
NAME      STATUS    ROLES     AGE       VERSION
master   Ready   masternode-    1d        v1.12.0
1    Ready    <none> node-2    1d        v1.11.3
Ready   <none>                  1d        v1.11.3
```

So next, the worker nodes. Let us start one at a time. We need to first move the workloads from the first worker node to the other nodes. The kube control drain command lets you safely terminate all the pods from a node and reschedules them on the other nodes. It also cordons the node and marks it untradeable, that way no new pods are scheduled on it.



Then upgrade the kubeadm and kubelet packages on the worker nodes as we did on the master node. Then using the kubeadm tool upgrade command, update the node configuration for the new kubelet version, then restart the kubelet service. The node should now be up with the new software version. However, when we drain the node, we actually marked it unschedulable, so we need to unmark it by running the command kube control uncordon node one.



The node is now schedulable but remember, that it is not necessary that the pods come right back to this node. It is only marked as schedulable. Only when the pods are deleted from the other nodes or when new pods are scheduled, do they really come back to this first node.

Well, it will soon come when we take down the second node, to perform the same steps to upgrade it. And finally, the third node. We now have all nodes upgraded.