:::Summary Week 8:::

Weeks goals:

* More precise delay measurements
* Timeline migration delays
* K8s Bare metal vs VM implementation
* Look deeper into application specific migration

Finally being able to simply parse relevant information from the huge S3 logs files I've retrieved more precise measurements regarding each part of the node/container relevant migration procedures.

They are the following:

:::Node:::

  Switch Reaction delay:

          Avg: 0.683s

Min: 0.515s

Max: 0.836s

  Send config delay:

Avg: 0.258s

Min: 0.214s

Max: 0.473s

  Image Download & Ready delay:

Avg: 2.994s

Min: 2.491s

Max: 3.363s

Complete Pod setup Delay:

Avg: 7.782s

Min: 7.114s

Max: 8.672s

:::Container:::

  Container creation delay:

Avg: 0.73s

Min: 0.56s

Max: 1.1s

Container start delay:

Avg: 3.2s

Min: 2.9s

Max: 3.8s

What do these titles mean you ask? well:

**Switch reaction delay:** The time delay of sending "kubectl apply..." (i.e the migration signal) to when k8s controller acts on that signal and starts migration procedure.

**Send config delay:** The time delay of moving the configuration of the first node/pod/container to the new node

**Image download & Ready delay:** The time delay of the pod retrieving the image from local repository(that is Harbor) and setting it up for usage. Done by Kubelet process.

**Complete Pod setup delay:** The time delay of all setup processes of the pod until the Kubelet starts the container build procedure.

**Container creation delay:** The time delay of dockers procedure for building/creating the container base for an application.

**Container start delay:** The time delay of booting up the application until first application signal is sent.

The visual timeline will be presented on the next discussion meeting we have, or if you want it sooner just write to me.

But to be precise this will show what processes actually run parallel to each other and thus what the expected average delay restraint actually is.

As for Bare metal vs VM implementation of a K8s system.

There are a lot of ups and down with either implementation, currently IceKube runs on Bare-metal, meaning that there is no extra virtualized infrastructure layers on top of each server and its container environments.

Management:

* Bare metal provides more control and simplifies administration in many ways such as:  
  + Network config. By removing layers of virtualized infrastructure bare metal simplifies network setup
  + Troubleshooting. Complexity of bare metal is smaller, thus troubleshooting is simpler than with VM's
  + Automation/deployment. Can be simpler to automate services and software on bare metal clusters because of the lack of virtualized infrastructure layers.
* Although VM management is generally easier.   
  + Scripts or VM orchestration tools make it easy to spin up large number VM's with pre-built images.
  + With VM architecture you can also images to create backups of your VMs and restore failed VMs.
  + Snapshot restore is also something that is often enabled for virtualization platforms.
  + Automated failover is usually provided as well.
  + Most of these features above can be done with bare metal servers but is usually much more complex to implement.

Performance:

benchmark study: <https://www.stratoscale.com/blog/data-center/running-containers-on-bare-metal/>

good material link: <https://platform9.com/blog/kubernetes-on-bare-metal-why-and-how/>

* Latency is generally 3 times lower on bare metal k8s.
* Containers generally perform 25-30% better on bare metal.
* less hypervisor overhead than with guest os environments as is done with VM's
* Edge architecture take advantage of both performance and low latency of bare metal servers
* VM's applications that depend on bare metal resources can be very limiting as opposed to running on bare metal.

Drawbacks (bare metal)

* Resiliency against node failure
* Ease of Management implementations

Lastly, as for application specific migration, I'll have more for next friday but most of the research and test implementations hint towards a protocol for synchronization. Either this has to be customized for each application or possibly a communication format must be generalized as to synch many application types with the same procedure and goal outcome.