Optimizations and design considerations for Kubernetes cluster for Large Scale

RaviShankar KS
Aug, 2019

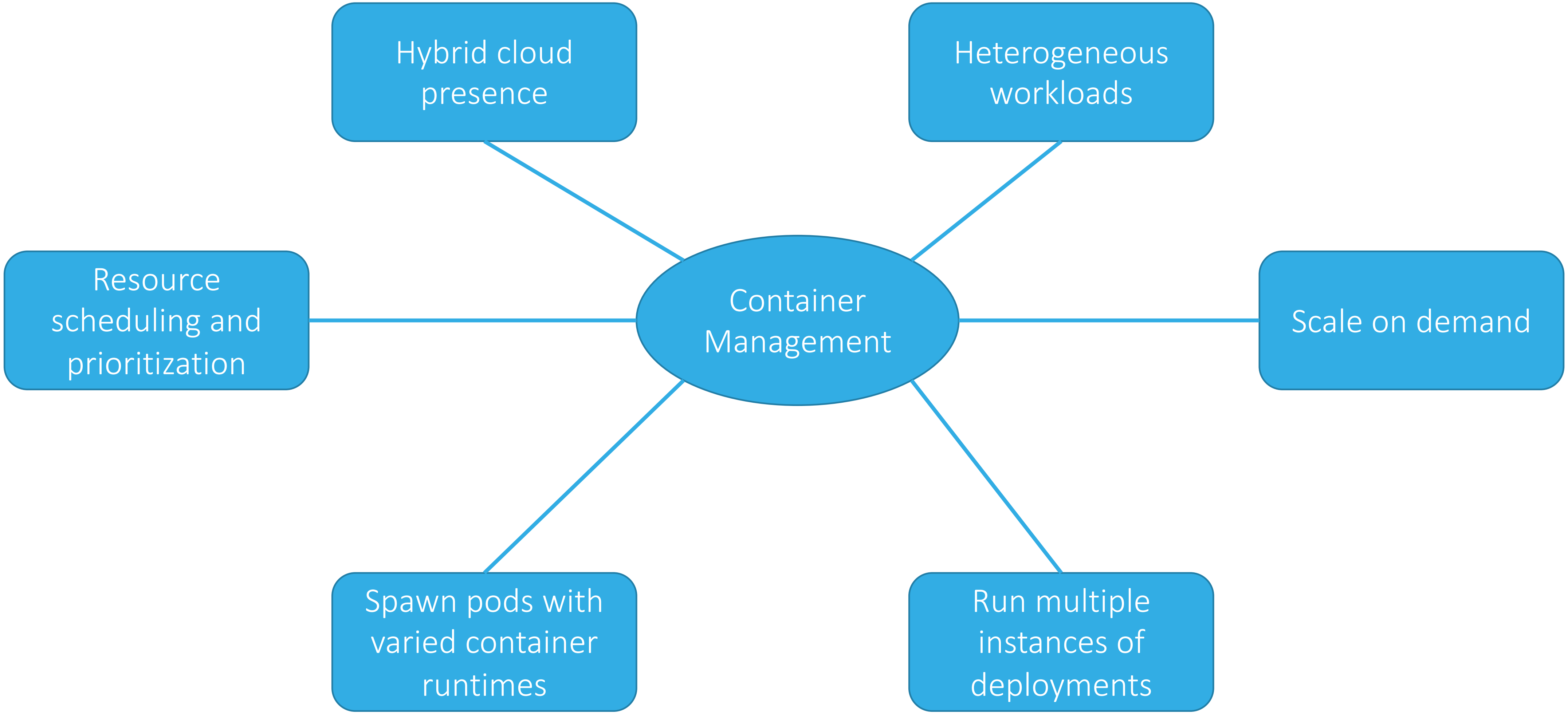
Outline

- Expectations

- Deep Dive
  - K8s components overview
  - SLAs and tuning
  - Dynamic Autoscaling
  - Monitoring
- Few more Lessons Learnt trivia of interesting use-cases and solution

- Q&A

Expectations



Container Management

Hybrid cloud presence

Heterogeneous workloads

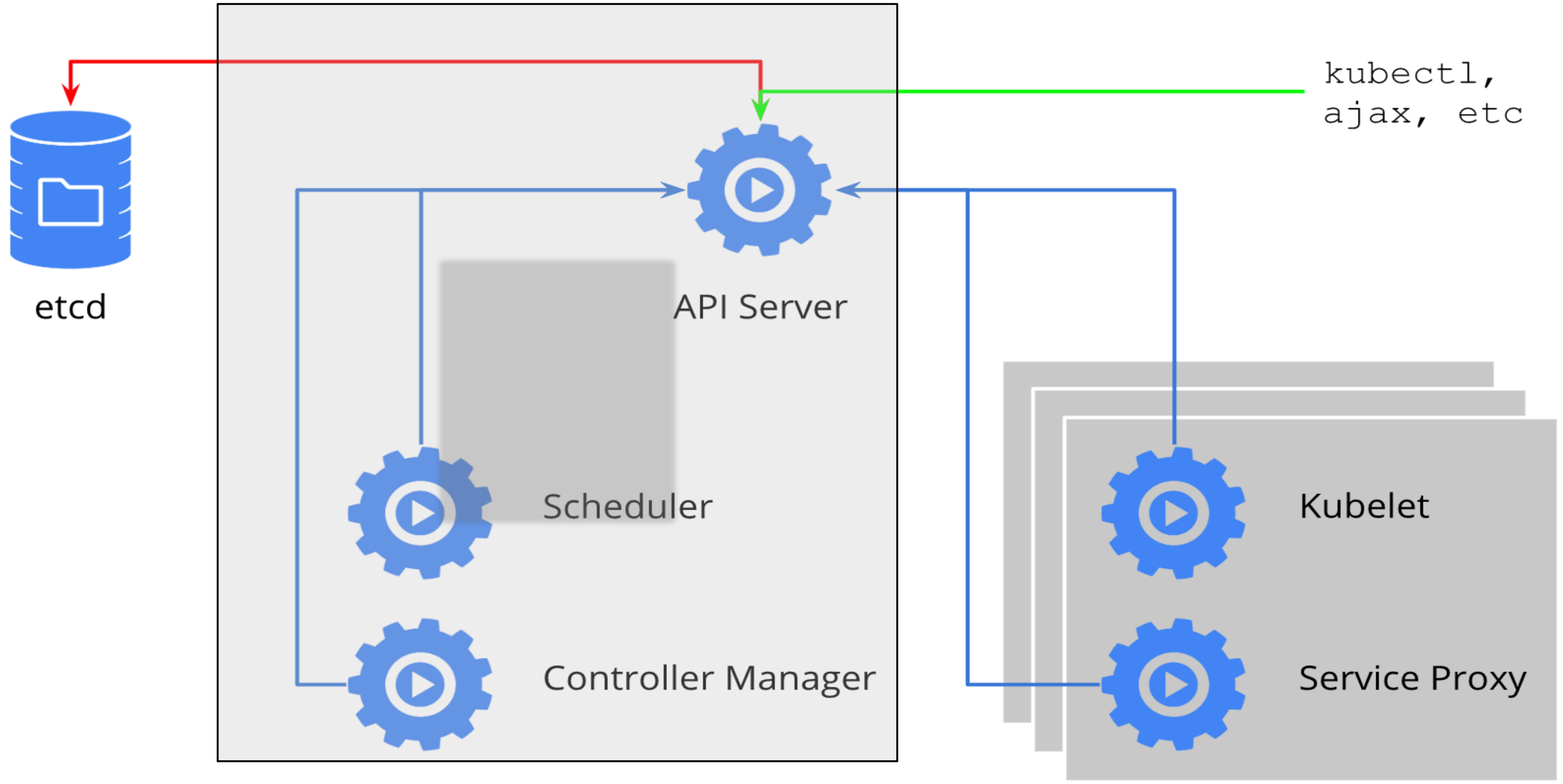
Resource scheduling and prioritization

Scale on demand

Spawn pods with varied container runtimes

Run multiple instances of deployments

Kubernetes Architecture



https://docs.google.com/presentation/d/1H4ywDb4vAJeg8KEjpYfhNqFSig0Q8e_X5I36kM9S6q0/pub?start=false&loop=false&delayms=3000&slide=id.gb2c433030_2_62

Kubernetes system components

etcd

- Persistent Key value store
- Needs Quorum/Leader to operate

apiserver

Security

- Admission Control – RBAC
- Authentication

Objects

- Caching
- Validation
- CRUD operations

Docker containers

- Logging
- Exec

Controller manager

- Watch/List k8s resources
- Uses Apiserver
- Moves resources from current state to desired state

Scheduler

- Watch Pending pods
- Eliminate nodes which are not fit for request to be fulfilled
- Rank Nodes

Kubelet + kube-proxy

- Node Level admission control
- Communicate with container through CRI
- Node metrics
- Report readiness / liveness

kube-proxy

- Update iptables with service endpoints

opportunities for optimization

- Keep K8s production grade and always running
  - Parameter tuning and trade-offs
  - Process panic
  - Node issues

Parameter tuning

apiserver	
# of k8s master nodes	3-4 Multi-zonal
Rate limit	--max-requests-inflight [400]
Choosing the correct Storage backend	[etcd3]
Feature gates	cpumanager debugcontainer
Caching	--target-ram-mb --event-ttl --audit-log-maxage
Auditing	--auditing*
# requests to etcd	--etcd-count-metric-poll-period duration
etcd	--etcd-compaction-interval
Expose Kubernetes service	--kubernetes-service-node-port

Parameter tuning

Node Level	
• Docker daemon hangs • Kernel crashes • Kubelet crash	Cron jobs
Monitoring	read-only-port [10250]
Docker fs issues	use overlay2

Port conflicts	net.ipv4.ip_local_port_range = [32768 61000]
	net.ipv4.ip_local_reserved_ports = [30000 – 32000]

etcd	
Quorum size	3,5,7 Multi-zonal
etcd sync	ETCD_ELECTION_TIMEOUT
	ETCD_HEARTBEAT_INTERVAL

general	
Networking choices	Calico / Flannel

controller manager	
Node monitoring	--node-monitor-period --node-monitor-grace-period
Parallelism – number of workers	--concurrent*
Caching	*cache-size*
Rate limit	--kube-api-qps --kube-api—burst
GC	--terminated-pod-gc-threshold

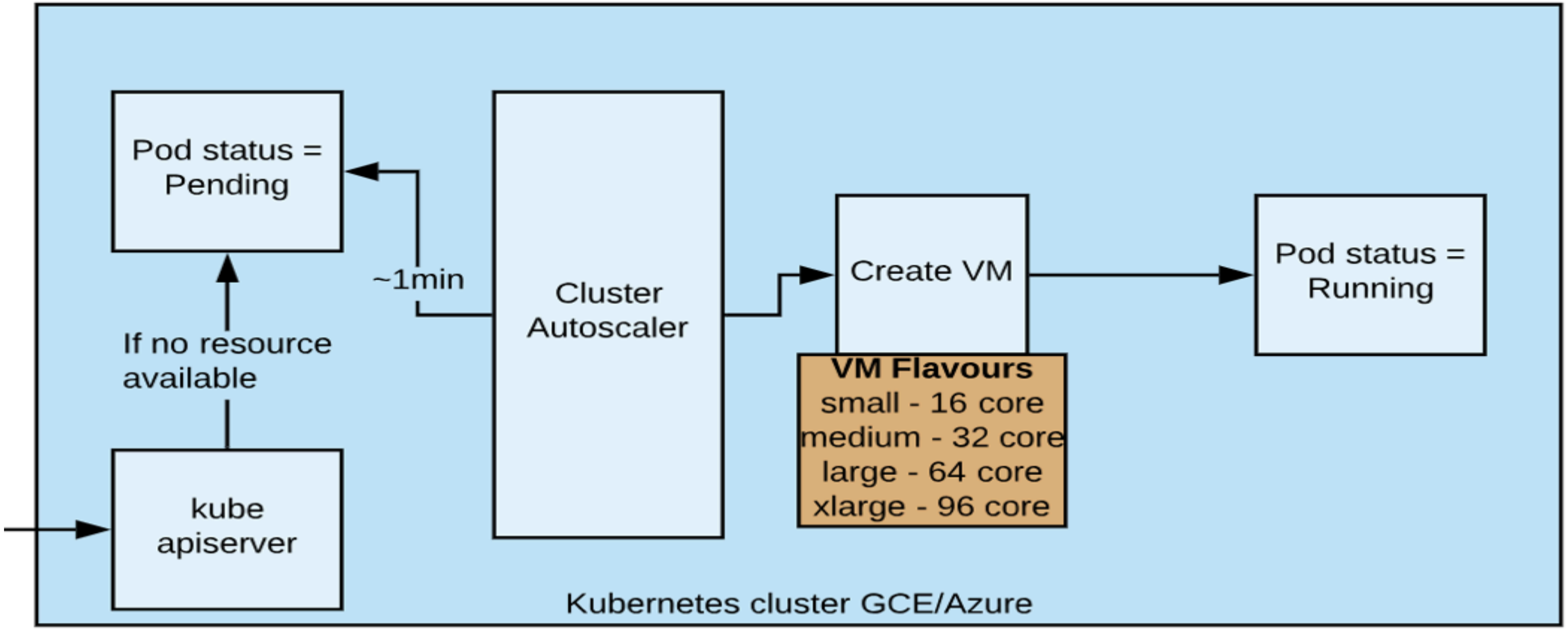
opportunities for optimization

- Capacity Allocation
  - granularize minions into flavors to avoid node wastage, reduce cost

- Auto-scaling for public cloud (Azure / GCP)
  - Modify kubernetes cluster-autoscaler for node autoscaling
  - Observe Pitfalls of API usage and rate limiting

- Logging / monitoring
  - Every metric matters

Capacity Allocation and autoscaling



kube apiserver

If no resource available

Pod status = Pending

~1min

Cluster Autoscaler

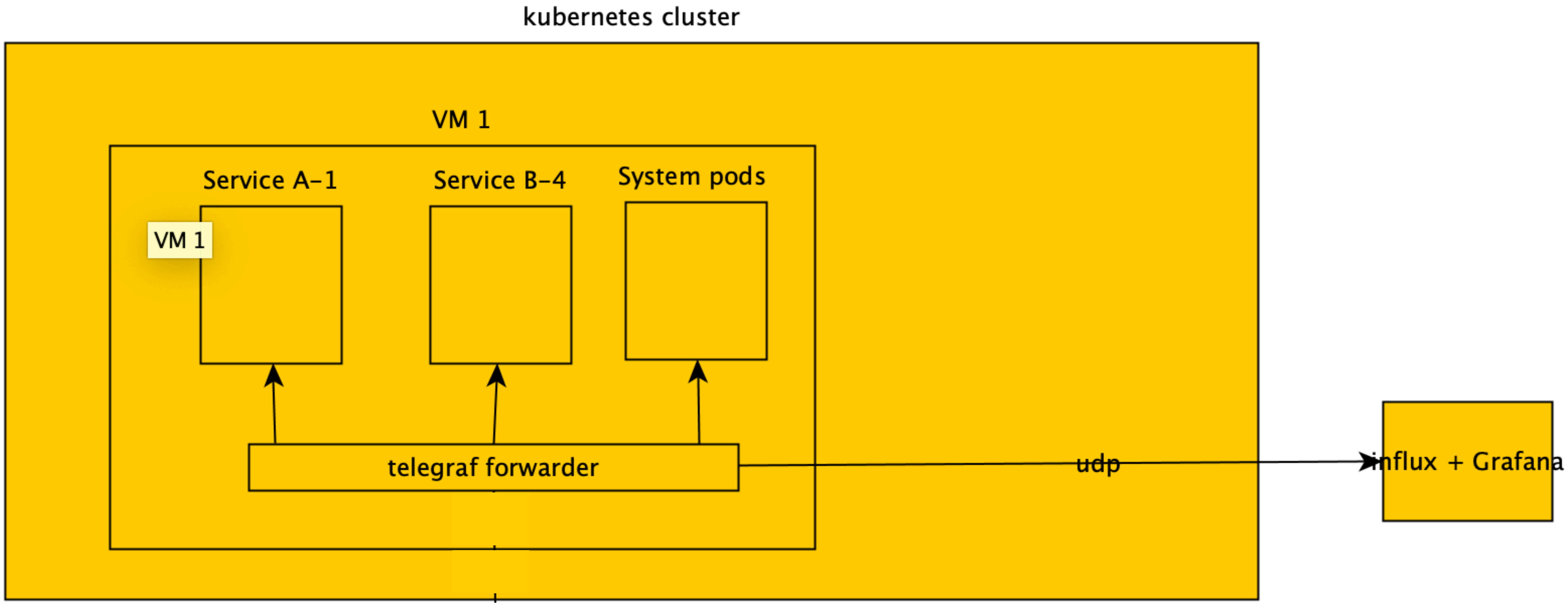
Create VM

VM Flavours
small - 16 core
medium - 32 core
large - 64 core
xlarge - 96 core

Pod status = Running

Kubernetes cluster GCE/Azure

Monitoring



Application Monitoring

1. API calls
2. DB conn
3. Health

Node/VM Monitoring

1. Disk Space
2. Disk Mounts

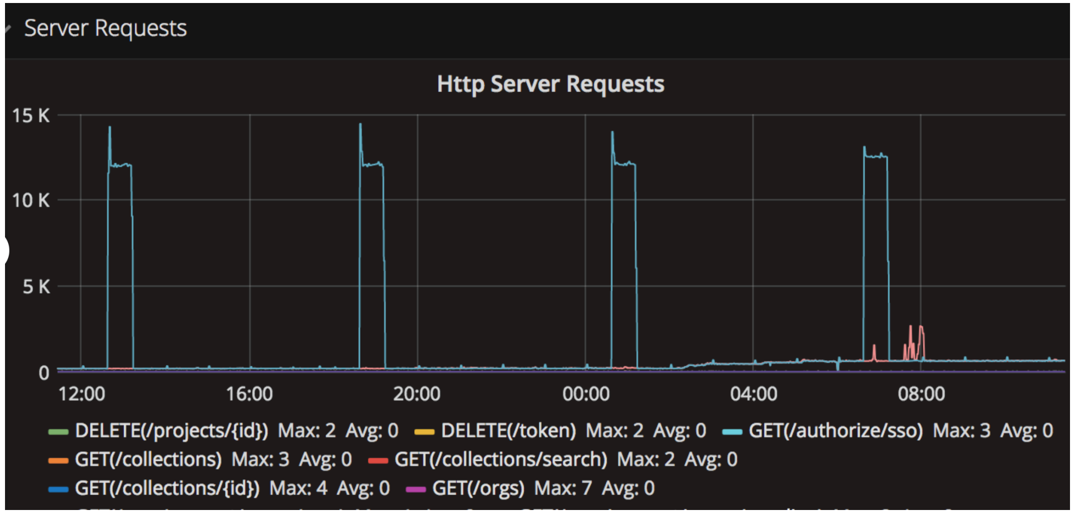
Device Monitoring

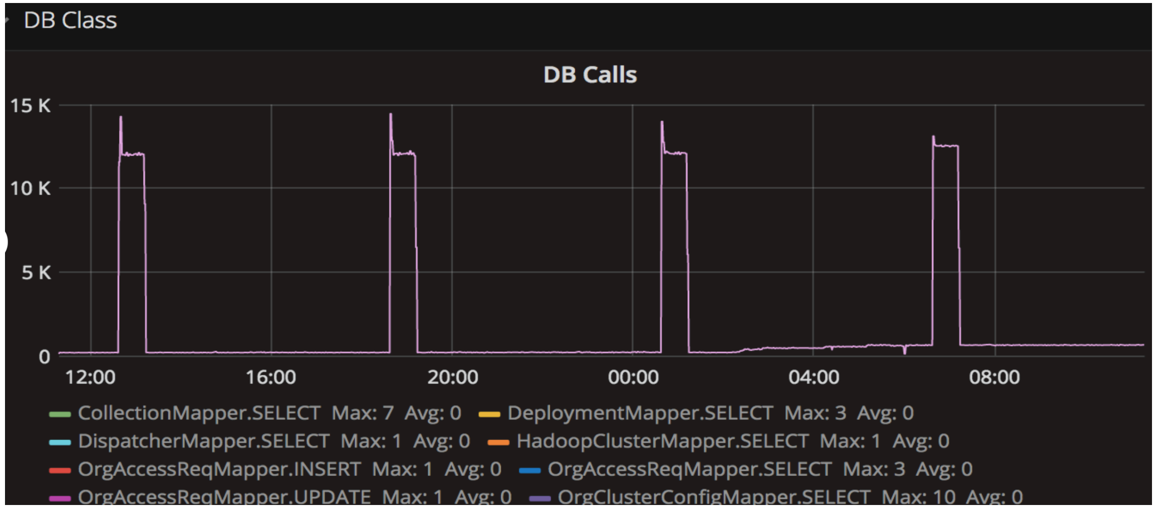
1. GPU
2. CPU/Memory
3. Disk IO

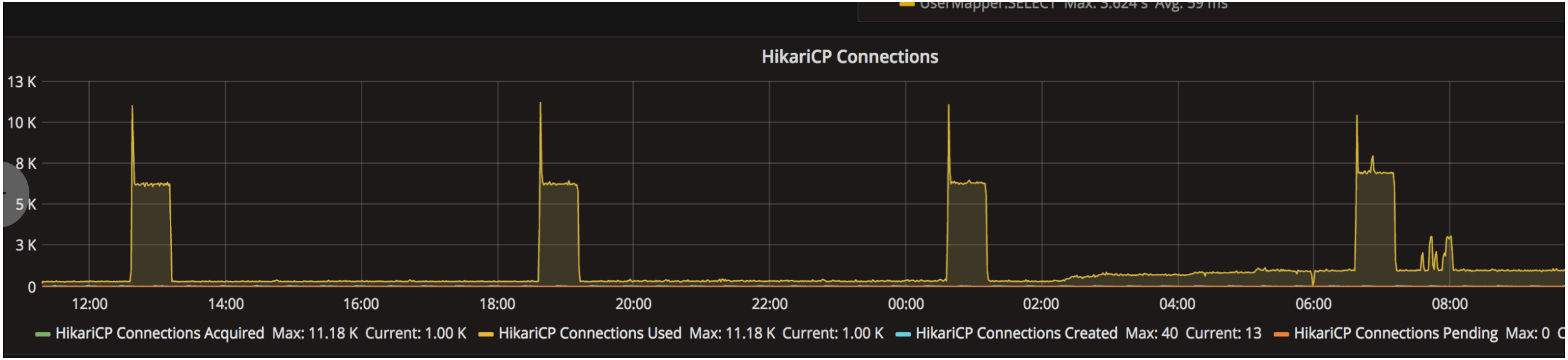
Cluster Monitoring

1. ApiServer
2. Controller
3. Scheduler
4. Kubelet
5. Etcd
6. DB Health

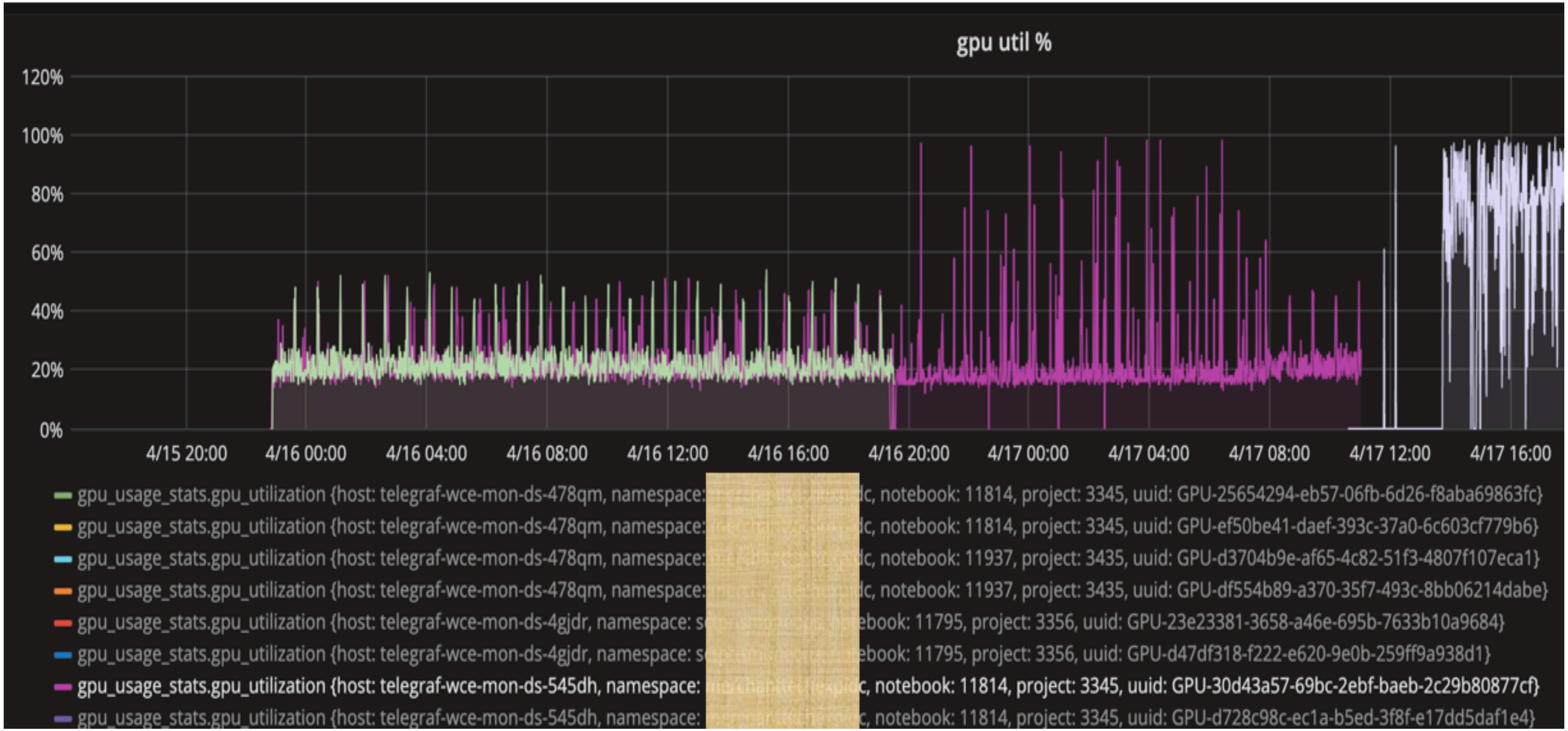
Search for patterns ....







Search for patterns ....



More lessons learnt….

- Using managed platforms – know limits
  - IP space restrictions over vpc
  - Image prepulls – not supported with coreos (GCE)
  - Limits for # pods, # node-groups not documented

- For persistent external storage
  - Using mount propagation technique
  - Kubernetes Volume Types

- Multi-zonal or multi-regional cluster setup for HA

- IP CIDR Allocation
  - Avoid conflicts
  - Keep room for long term increase in pods/services

- Using ingress endpoints – avoid more number of NodePorts/ClusterIP service types

Q&A

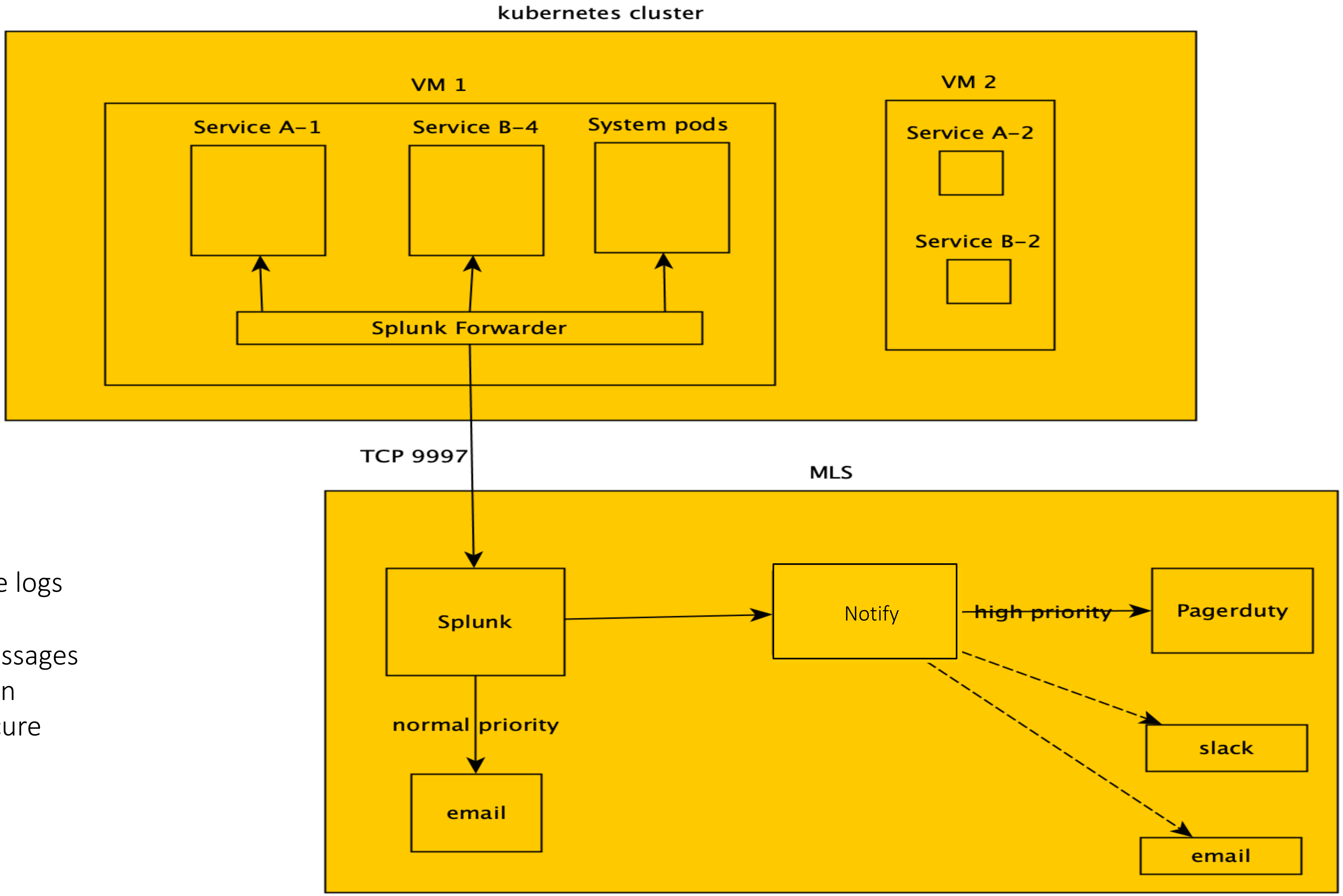
References

- https://www.slideshare.net/harryzhang735/kubernetes-beyond-a-black-box-part-1
- https://www.slideshare.net/harryzhang735/kubernetes-beyond-a-black-box-part-2
- https://www.slideshare.net/applatix/webcast-making-kubernetes-production-ready-78130374
- https://www.slideshare.net/LCChina/scale-kubernetes-to-support-50000-services
- https://unix.stackexchange.com/questions/111858/randomizing-the-source-port-for-new-connections/111869#111869

Addendum slides

Design considerations / Features	Kubernetes	Docker Swarm
Open Source	Highly modularized with tons of plugins	Less modularized
Stable setup in hybrid clouds	Highly stable, if configured with correct settings for individual components	**Limited Fault Tolerance.** X
Scale On-demand	Cluster-autoscaler module	**Not natively supported** X
Resource scheduling and prioritization	Native support	Native support
Heterogeneous workloads	NVIDIA plugins available	**Lot of scripting Effort** X
Run multiple replicas with HA	Available as deployments	Native support
Access Control	Native RBAC support	**Only available in EE** X
Other features – privileged mode, mount propagation, NFS support, cpu pinning	Natively supported or plugins available	**Complex to setup** X
Support base	Large active community	**Not much traction , though still pursued.** X
Learning curve	Steep learning curve during setup/maintenance phase	Fully Integrated to docker Engine/CLI
Upgrade to higher versions	Not Easy	Easy to upgrade
Post setup maintenance phase	Various choices for logging and monitoring	3rd party only

Logging



- Log from:
  - Application logs
    - microservice logs
  - Nodes
    - /var/log/messages
    - /var/log/cron
    - /var/log/secure
  - Kube system logs
    - Apiserver
    - Controller
    - Kubelet etc

More Lessons Learnt

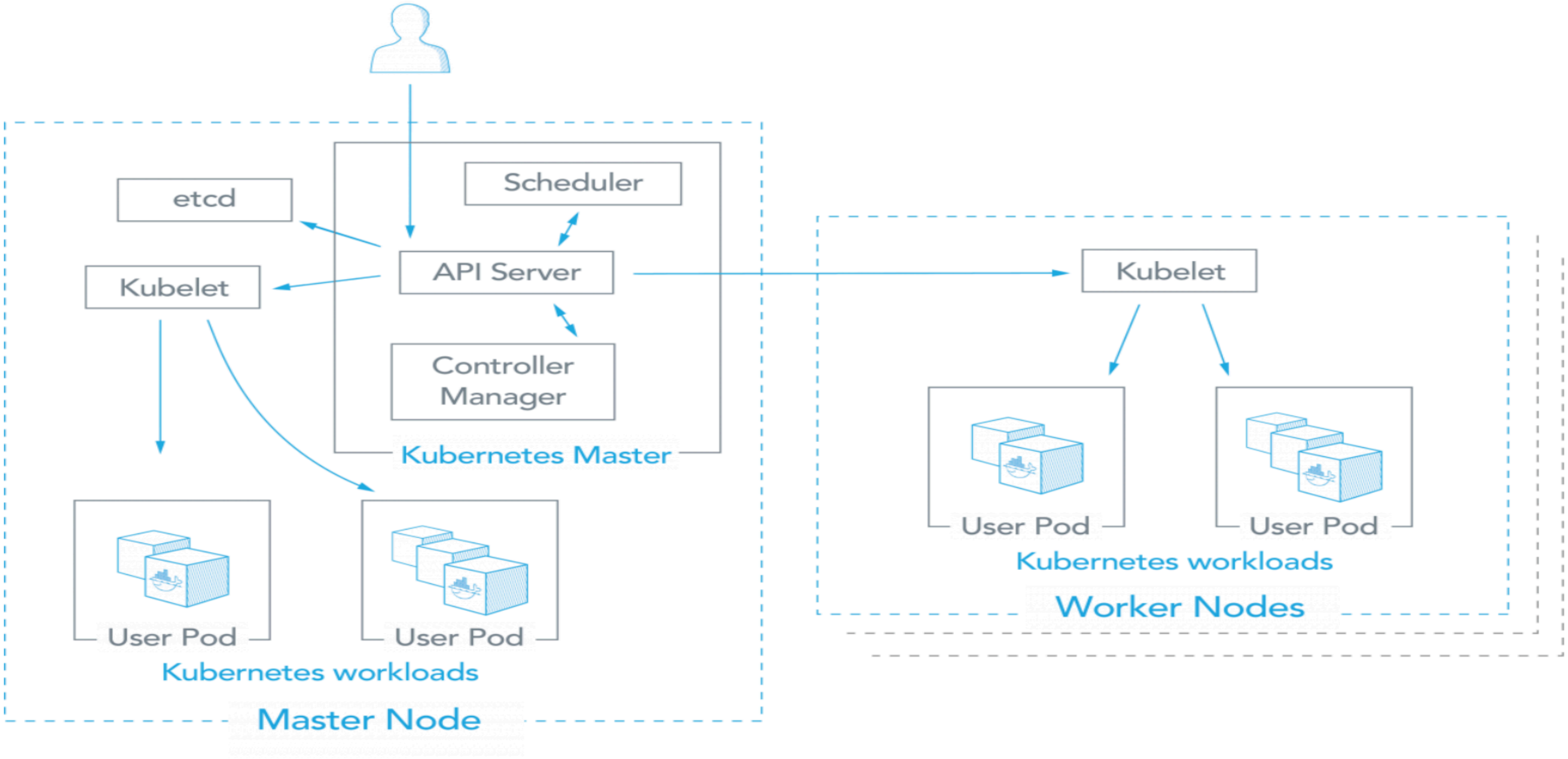
- Why not use GKE ??
  - IP space restrictions
  - Image prepulls – not supported with coreos

- GCS mounts on GPU
  - Use mount propagation

- Multi-zonal or multi-regional cluster setup for HA

- IP CIDR Allocation
  - Avoid conflicts
  - Keep room for long term increase in pods/services

Kubernetes Internals



API Server: management hub for Kubernetes
Scheduler: places a workload on the appropriate Node
Controller Manager: scales workloads up/down
etcd: stores configuration data which can be accessed by API Server

Kubelet: Receives pod specifications from API Server, updates Nodes
Master Node: places workloads on Nodes
Worker Nodes: receives requests from Master Nodes and dispatches them
User Pod: a group of containers with shared resources