INDEX

Name : Shreyas Gune

Std. : LOW Div. : F Roll No. : ∞

Sub. : KUBERNETES

School : OF LIFE

Sundaram®
Books for Success...

Sr. No.	Date	Subject	Page No.	Sign.
1	13/1 - 15/1	Kube Basics Refresher	1 - 14	
2	15/1 - 16/1	Kube Networking	15 - 23	
3	17/1 - 17/1	Storage	24 - 30	
4	18/1 - 18/1	Cluster Scaling	31 - 33	
5	18/1 - 18/1	ConfigMap & Secrets	34 - 36	
6	19/1 - 19/1	Primitives	37 - 40	
7	19/1 - 19/1	Sers & Jobs	41 - 45	
8	22/1 - 22/1	Schedulers	46 - 49	
9	23/1 - 23/1	Resources	50 - 53	
10	24/1 - 24/1	Cluster in Azure [DEMO]	54 - 59	
11	28/1 - 28/1	Access control	60 - 69	
12	30/1 - 30/1	Cluster Upgrades	70 - 71	
13	30/1 - 01/02	Cluster Monitoring & Backup-Restore	72 - 76	
14	02/02 - 02/02	Kubernetes Troubleshooting	77 - 83	
15	02/02 - 02/02	Application Monitoring	84 - 88	
16	03/02 - 03/02	Practical Troubleshooting	89 - 100	
17	06/03 -	Symantic Versioning	101 -	

Sr. No.

To create K8's resource YAML without creating the resource in the cluster:

kubectl create
<resource-type>
<resource-name>
--dry-run=client
-o yaml

PAGE No. 1
DATE 13/01/2024

Kubernetes Basics Refresher

1) A Cluster is a collection of computational & storage resources in a distributed system.

- A K8s Cluster is made up nodes, of type control plane and type worker.

- Control plane is made up of a set of master nodes. These are useful in managing the system.
- The node can be a VM or a bare metal machine

- The actual microservices that you "deploy", which is basically running images in a container runtime, that exists on the worker nodes

> You can do a single node cluster where the control plane and worker stuff runs on the same VM / bare-metal machine. Minikube does this

- You need compute, networking and storage to get a functional kubernetes cluster going.
- You can get these resources on-premise, in a cloud setting or as managed service (GKE, AKS, AWS-kube)
easiest but you're at the mercy of the cloud provider.

2) Some HW nuts and bolts:

- Get at least 2vCPU & 2GB RAM on your kube node.
- Before you install the container runtime, you need to setup forwarding of IPv4 traffic and letting IP tables see bridged traffic.
- IPv4 forwarding allows the kernel to forward packets between network interfaces, which is crucial for communication between pods.
- IP tables allowed to see bridged traffic ensure proper handling of network rules and policies for container communication within the cluster.

- Container runtime to prefer is containerd. It's simple, stable and compatible. Ideal for kube managed orchestration. RunC is a low-level container runtime bundled with docker or containerd.
- CNI Plugin: Container Networking Interface (CNI) plugins help manage network connectivity
  1) Isolation: ensure that pods dont talk directly to each other, but talk via a network segmentation.
  2) Handle IP management: assign & mange Pod IPs.
  3) Network Policies: You can define rules.
  4) Overlay Networking: Enable pods on different nodes to be able to communicate (helps scalability)
  5) Integrate with Container runtimes.

Cluster creation
Demo on page 54-59

3) About Kubelet

- Primary node agent that runs on each node (worker + master)
- Registers node to be added to the cluster.
- Takes instructions to start a container & ensures their health.
- Works with container runtime to launch pods & their containers.

> container runtime NEEDS to conform to CRI standards so that kubelet can interact with it.

4) kubeadm - create cluster
kubectl - manage cluster

> Need to have SELINUX run in permissive mode to allow containers to access the host file

5) Networking Bits

- CNI based Pod Network : so pods can talk to each other.
- CoreDNS (depends on this pod network) and is needed to provide DNS service.
- Pod Network is set up using Calico or Flannel.
- Flannel uses 10.244.0.0/16 by default

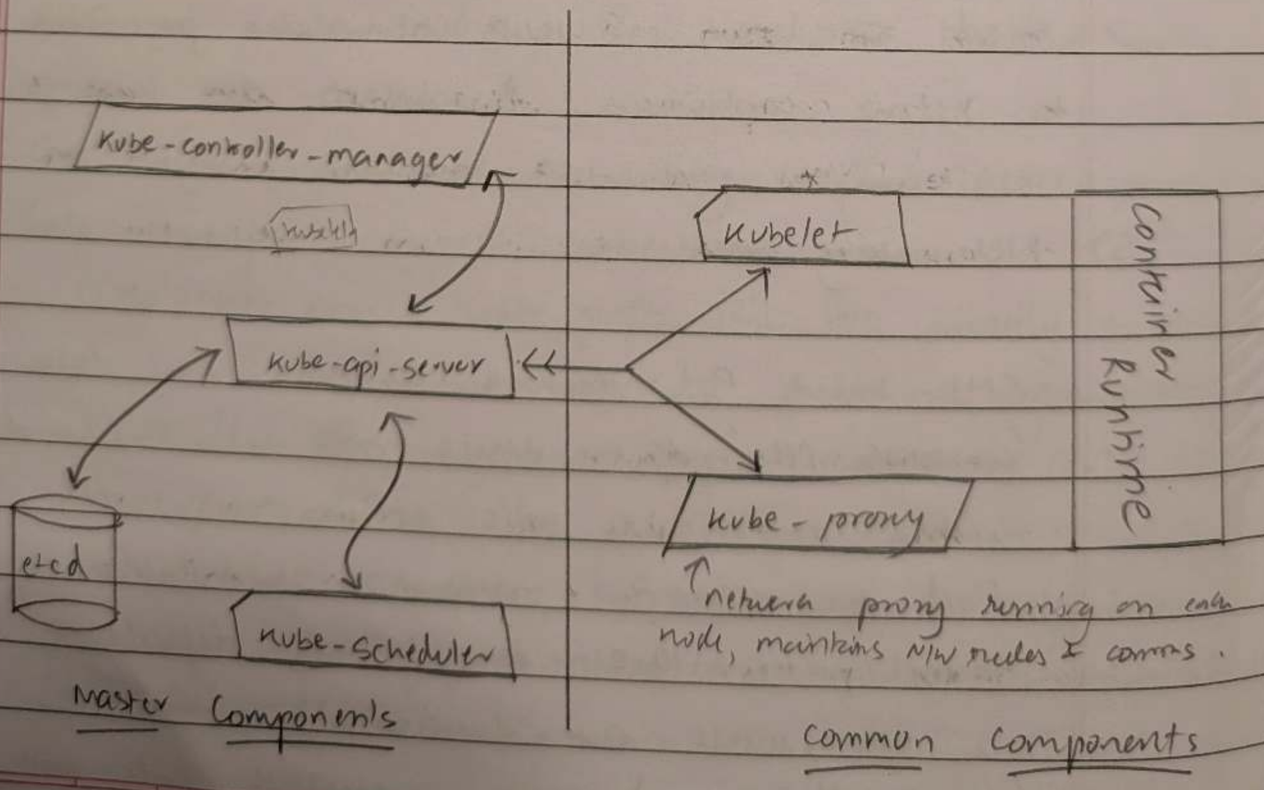
> Once you create the cluster using kubeadm, you can then install flannel & the CoreDNS pods will come up.

6) Adding a worker

- A new VM/box, install containerD, plugins & related binaries (kubectl, kubeadm etc)
- Use the kubeadm to register this worker node to the cluster.
- If successful, you can go back to the master node and do a kubectl get nodes, to see your registered node.
- To assign a role to it, you need to assign a label to node, as 'worker'

kubectl label node <node-name> node-role.kubernetes.io/worker=worker

7) Cluster components



- kube-api-server
1) front-end for the k8s control plane
2) kubectl is basically talking to this, via REST.
3) It runs as a pod, in the kube-system namespace

- etcd
1) stores config data for the cluster. This represents the state of the cluster (which nodes & pods are running and where)
2) also runs as a pod in kube-system namespace

- kube-scheduler
1) watches newly created pods, picks a node for them to run on.
2) This component looks at the resource numbers, affinity stuff that we mention in our pod manifests.
3) also runs as a pod, on kube-system

- kube-controller-manager
1) Runs as a pod, that has a combo of many controllers, compiled into a single binary, so a single process.

Controllers included:
- Node Controller: ensures node comes back if dead
- Job Controller: ensures jobs get finished
- EndpointSlice Controller: helps join k8-svc with k8-pods
- Service Account Controller: Create SA & API tokens for k8-ns

> Only kubelet & Container Runtime does NOT run as pods, everything else, runs as pods (in ns=kube-system)

8) Accessing the Cluster (so that we don't need to ssh into master)
1) gotta have kubeconfig (found @ ~/.kube/config)
2) Once you get the kubeconfig, make sure that the 'server' address field under -cluster: key has the master's external IP address / or DNS name
3) also open port 6443 for inbound traffic to the master node.
4) You also need to go to the api server certs and update it with this external IP address.
-) gotta " ls -l /etc/kubernetes/pki/apiserver.* " on master.
-) You're going to see old certs, nuke them.
-) Make fresh certs

sudo kubeadm init phase certs apiserver --apiserver-cert-extra-sans = <external IP address>

-) This makes new .pem & .crt files.

to copy stuff, use SCP
> scp -i (somekey.pem) <node-address>:<path-to-file>

9) Pods & Containers

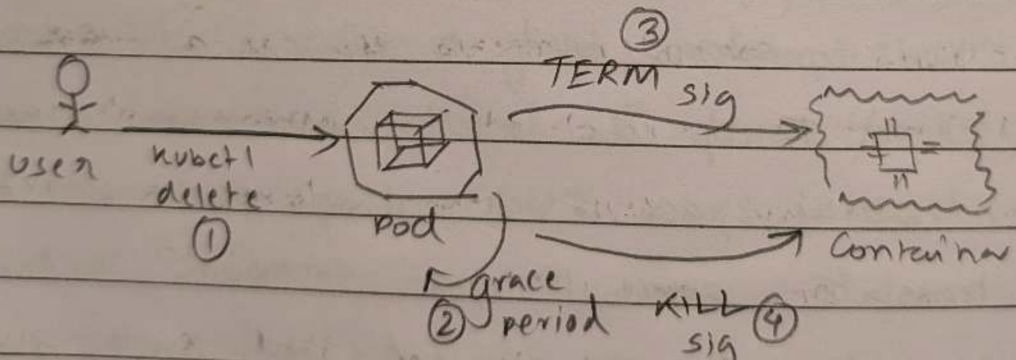
- Pods are the smallest unit in a kube system, and they run containers.
- They do not run by themselves and need an external controller to manage its state (running, replication & healing)
- Controller types = deployments, stateful sets, Daemonsets, etc
- In addition to running containers, pods can
  - init containers - run first and finish a specific task
  - app container - runs the actual app image
  - storage resources - mount volumes in pods.
  - Unique IP on pod, containers can talk to each other on localhost within the pod network
  - when containers need to communicate OUTSIDE the pod, they use the host-machines (worker node) network space. Host machine performs NAT to translate source IP from containerIP to host-IP, so when it gets a reply packet, the replies packet destination is host IP [which gets NAT'd back to (containerIP: Port)]
  - Each container within a pod has a unique IP but all containers in a pod share the same network namespace & port space.
- container comms within pod : use container IP
  container comms outside pod : use pods IP

10) Pod Phases

.) Pending : cluster is aware, node is downloading images etc
.) Running : running, starting, restarting container
.) successful / completed : all containers gucci
.) Failed : all containers terminated, at least one failed (non zero)
. Unknown : dunno what happened

> to run a pod forever :
command: ["sh", "-c", "tail -f /dev/null]

11) Pod Termination



default grace period: 30 seconds
> can specify flag in delete cmd : --grace-period=<x seconds>
> force delection == insta nuked from cluster & etcd

12) Namespaces

- If you don't define the namespace in your deployment command or manifests, it will end up in 'default' NS
- In linux, namespaces are meant to isolate processes from one another
- In K8s, namespaces isolate resources within the cluster.
- some info about baked-in namespaces:-
  1) kube-system: objects created by kube-system
  2) kube-public: objects that need to be accessed by all ns.
  3) kube-node-lease: lease objects that kubelet needs look at to determine node health

> kubelet creates lease objects (and periodically renews) on the node. Node-lifecycle controller treats this as a health signal.

  4) kube-flannel / kube-calico: networking bits namespace

- some low-level objects like volumes & nodes, don't belong to any namespace
- to find out what resources are tied to a namespace, run:

```
kubectl api-resources --namespace=true
```

↑ false (for non-ns resources)

- all resources in a particular namespace need to have unique names for that 'type'. so: (pod: hello, pod: hello ✗) (pod: hello, svc: hello ✓)
- Namespaces are generally cool for different flavors of system, different 'branches' of a game, or different teams, on the same cluster

13) Resource limits in a Namespace
- we dont want resources in a namespace to hog up all the resources available on the worker nodes.
- got to assign resource quotas to each namespace
- >cluster level resources available to kubeadmins
>namespace level resources available to app-developers.

14) Organisational tools terminology
Labels : >key-value pairs attached to K8 objects, helps structure.
>can add them in manifest or later via kubectl.
>labels are optional, but each key needs to be unique

Selectors : >used to identify a set of objects; we add labels to objects and use selectors to identify and group objects based on labels.
> Equality based (==, =, !=) : matching objects gotta satisfy all specified labels.
> set-based : match groups
in : checks if key-value is present
notin : checks if key-value not present
exists : checks if key exists

Annotations : > Used to provide additional metadata to objects.
> NOT used to identify or select objects
> annotations go in:
metadata:
annotation:
key: "value"

15
in the con

15) Deployments

- Basically a controller, to whom we submit a desired state of the object & the controller ensures that state persists.
- Ideal for stateless applications (apps that do not store data or app-state)
- The statelessness makes it very scalable
- Deployments create Replicasets, that then go create the pods.
- If you want to nuke pods, you need to nuke the whole deployment, as if you nuke the pod, the replicaset will just replace it with a new pod.

16) Commands & Arguments (on Containers)

- Usually each container has a default command thats running (because it was baked in the Dockerfile).
- If you want to check it, find the pod name, then> kubectl -n <your-namespace> exec -it <pod-name> -- sh

in the container: # ps -ef

- If you want to override the default, you would need to add that to the deployment manifest, args field eg:-

```
spec:
  containers:
  - name: ubuntu-gman
    image: ubuntu
    tag: latest
    command: ["printenv"]  } <- override
    args: ["HOSTNAME"]
```

>Its going to go into CrashLoopBackoff if it does the command and has nothing else to do, it keeps restarting

PAGE No. 12
DATE / /

- If you have many commands or a shell script, you will need to use) command: ['sh']
  args: ['-c', '<your script>']

- If you update the command/arg, the old pods are DELETED and new ones will come up.

17) Got a shell script, wanna kube-it-up, HOW?

a) Configmap it.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myscript-config
data:
  script.sh: |
    #!/bin/sh
    echo "Hello There!"
```

b) Reference that in a Volume on Pod manifest

```
spec:
  volumes:
  - name: script-volume
    configMap:
      name: myscript-config
      defaultMode: 0777
```

to make script executable, as you can't do chmod +x on read-only FS.

PAGE No. 13
DATE / /

c) Mount the volume on the container

```
spec:
  containers:
  - name: mycontainer
    volumeMounts:
    - name: script-volume
      mountPath: /usr/local/bin/
```

d) Use args & command to execute it

```
spec:
  containers:
  - name: mycontainer
    command: ["/bin/sh", "-c"]
    args:
    - /usr/local/bin/script.sh && tail -f /dev/null
```

to sustain the pod

d) Commands used to verify this whole deal:

- kubectl create ns gman
- kubectl apply -f script-configmap.yaml -n gman
- kubectl -n gman apply -f mypod.yaml
- kubectl -n gman logs mypod
- kubectl -n gman exec -it mypod -- sh
- # ls -la /usr/local/bin
- # cat /usr/local/bin/script.sh

PAGE No. 72
DATE 30/01/2024

Cluster Monitoring in K8s
+ Backup-Restore

1) Probes

a) Liveness Probe: to determine if a container is still running and healthy. occurs periodically while container is running. If probe fails, K8s will restart the container.
(used in DB cases)

b) Readiness Probe: to determine if a container is ready to receive traffic. periodically while the container is running. If probe fails, K8s will STOP routing traffic to the container
(used in app traffic checks)

c) Startup: to determine if a container has successfully started up, happens when the container has first started. If this fails, no traffic will be routed.
(used in apps taking long to start)

2) Parameters

a) initialDelaySeconds: how long should the probe wait before firing the check
b) periodSeconds: how often is the check run (frequency)
c) TimeoutSeconds: time to wait before firing the subsequent checks.
d) SuccessThreshold: Number of consecutive successes before the container is considered to be healthy
e) FailureThreshold: Number of consecutive failures before the container is considered to be unhealthy

PAGE No. 73
DATE 01/02/2024

3) Probes (other Types)

Command - executes a command (exit code = 0 : healthy, = 1 : non-healthy)
HTTP GET - makes an HTTP request to a specific endpoint
TCP Check - establish a TCP connection to the container on a specific port.

good for non-web app containers

This is specified in the 'type' field of a probe spec:

```
containers:
- name: my-nginx
  image: nginx
  (readinessProbe/livenessProbe/startup):
    type: (tcp or exec) OR httpGet:
```

4) Cluster Monitoring

a) Node Not Ready
- kubectl describe node <node name> : To check the conditions & events on the node
- kubectl logs node/<node-name> : check the logs
- kubectl get pods -o wide : check the pods across nodes, the nodes 'may not be connected'

b) Scheduling Disabled
- Manually disabled during cluster upgrades
- Resource requirement have not been met
- Nodes have been tainted (invisible for scheduling)
- you could manually edit the node spec to fix this

c) Unreachable Control Plane
> Got SSH into the master VM to run this cuz you can't hit the control plane via kubectl on workers.
- kubectl get componentstatuses
- kubectl logs <componentName>
- Check VM logs & Network logs.

5) Etcd Backups
a) Create the backup
sudo ETCDCTL_API=3 etcdctl snapshot save snapshot.db --cacert
/etc/kubernetes/pki/etcd/ca.crt --cert
/etc/kubernetes/pki/etcd/server.crt --key
/etc/kubernetes/pki/etcd/server.key

b) Verify the backup
sudo ETCDCTL_API=3 etcdctl snapshot status snapshot.db
sudo ETCDCTL_API=3 etcdctl --write-out=table snapshot status snapshot.db

c) Backup the certs (for the particular date)
umask 002 && sudo tar -czvf 'date "+%Y-%m-%d-%H-%M-%S"' -etcd.tgz /etc/kubernetes/pki/etcd

d) Verify cert backup
Check tgz with tar -tzv

6) Etcd Restore
a) Check settings in etcd.yaml
sudo cat /etc/kubernetes/manifests/etcd.yaml | head -n35 | tail -n 24

from data in that file, perform:

b) Restore command
ETCDCTL_API=3 etcdctl snapshot restore snapshot.db
--endpoints=< >,< >
--cacert = /etc/kubernetes/pki/etcd/ca.crt
--cert = " " " /server.crt
--key = " " " /server.key

... continued command

```
--name = kube-master-2
-- data-dir= /var/lib/etcd
--initial-cluster = kube-master-1=<ip: 2380>, kubemaster-2=<ip:2380>
--initial-cluster-token= kube-master-2
--initial-advertise-peerurls= <master-2- IP :2380>
```


Kubernetes Troubleshooting

1) Cluster Level Logging - it a way of managing logs, involves collecting logs from all the pods and storing them in a central location.

- How to collect these logs?

a) Node level logging agent

b) Include a sidecar container (like fluentd)

c) Push logs directly to a backend from inside an application

example YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: busybox-count
    image: busybox:1.28
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$i : $(date)" >> /var/log/1.log;
        echo "$(date) INFO $i" >> /var/log/2.log;
        i=$((i+1)); sleep 1;
      done
```

volumeMounts:
- name: varlog
  mountPath: /var/log
volumes:
- name: varlog
  emptyDir: {}

d) External Solutions
- Elasticsearch
- Fluent
- EFK
- Prometheus / Grafana / Loki

e) Features of Cluster Level Logging
- Centralized Storage
- Search & Filtering: you can search and filter logs by podname, container-name, timestamps or other criterias.
- Alerting: set up alerts to get notified when certain events occur like pod crashing or error spikes.
- Integration with other tools: for data analytics & monitoring.

2) Node Level Logging: Collect logs from individual node in the cluster.
- Kubernetes uses the kubelet component to collect logs from each node and store them in a local file on the node.
- You can use kubectl logs / kubectl exec to view logs from individual containers on the node.

kubectl logs (pod)
kubectl logs (pod) -c (container)

- Benefits
a) Increased visibility - a centralized view of all the logs in the node.
b) Improved troubleshooting - can search & filter logs to find where the problem exists.
c) Reduced Noise - filters out irrelevant or duplicate messages.
d) Improved compliance - can help meet compliance requirements by providing a centralized audit trail.

- Challenges
a) Data volume - Node level logging can generate a lot of data, could be a challenge to store & manage.

5) Complexity - it could be hard to set up and manage.
6) Cost - costs could skyrocket based on tools & storage class tied to the cluster

- Guard Rails

You can set up size limits for log size + number of maximum files on container level to guard against logs getting out of hand.

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    volumeMounts:
    - name: logs
      mountPath: /var/log/nginx
  volumes:
  - name: logs
    emptyDir: {}
```



```yaml
spec (contd..):
  terminationGracePeriodSeconds: 30
  dnsPolicy: ClusterFirst
  restartPolicy: Never
  securityContext:
    runAsUser: 1000
    fsGroup: 2000
  containerLogMaxSize: 10Mi
  containerLogMaxFiles: 5
```

guard rails

> Cluster level Logs vs Node Level Logs

- Cluster level logs collect logs from all the nodes, and provides a centralized view of the logs, making it easier to monitor & troubleshoot issues across the cluster.

- Node level logs collect logs from individual nodes, provides a detailed view from each node, helps with troubleshooting of issues for each node

Kibana is for **logs** not **metrics**

3) Practicals (Logging)

- The logging pods for an external solution goes into its own namespace.
- For EFK (Elastic - Fluent - Kibana) demo here, we're putting it in namespace = kube-logging
- kubectl get pods -n kube-logging

  pods in the namespace:

  es-cluster-0

  es-cluster-1

  ES-Cluster-2

  kibana-pod

  fluentd-pod-node1 ← it will be on each node

> Elastic Aggregates logs

> Kibana displays indexes & reports on them

> fluentd, fluentbit, logstash -> help port forward the logs.

- Use port-forward on the kibana pod on port 5601, to see the UI

  kubectl port-forward -n kube-logging <kibana-pod> 5601:5601

  you can goto localhost:5601 to discover & interact with the logs

- The only exposed services are elasticsearch (Port: 9200/9300) + kibana (Port: 5601)

  > fluent is NOT exposed

4) Practicals (Metrics)

- Install prometheus using the helm chart.
- Install kube-state-metrics chart
- Install Grafana Chart
  - ↳ make sure to get the POD-NAME & secret (password) from the helm install output
  - ↳ kubectl --namespace grafana-monitoring port-forward $POD-NAME 3000
  - ↳ browse to localhost:3000 and add in admin as user + password.
  - ↳ go to administration > Data sources > Prometheus
    - ↳ record the prometheus-server CLUSTER IP & PORT, add it to HTTP sections URL field. > SAVE & TEST
  - ↳ In Explore tabs, you can see all the metrics.

> You can use Elastic Search as a data source in grafana. You do not need to use kibana

↳ you may not have an IP, so just use the service name in the URL (this line in format) along with port 9200.

http://elasticsearch.kube-logging.svc.cluster.local:9200

PAGE No. 84
DATE 02/02/2024

Application Monitoring

1) Fundamentals

a) Stderr & stdout of container logs
(Stderr: standard err; stdout: standard output)

> • docker logs <container-name> displays stdout & stderr simultaneously.

- For stdout only: docker logs <container-name> 2>&1 | grep -v '^E'
- For stderr only: docker logs <container-name> 2>&1 | grep '^E'

> Apply same grep for kubectl logs

? grep -v '^E'
- -v: inverts
- '^E': regex that matches for stuff starting with 'E'

? 2>&1 means redirect stderr to stdout
- 2: file descriptor for stderr
- 1: stdout
- >: redirect

b) Kube-dash tool : A UI to look at kube stuff

c) Prometheus-Grafana to troubleshoot cluster component failures

d) Troubleshoot networking

* look at page (67)

PAGE No. 85
DATE / /

2) Set up & Access kube-dash

> either you need a service account token with sufficient rights or log in with kubeconfig.

* > usually, create a SA token with a SA user & log-in

a) kubectl get secret dashboard-sa-token -o json | jq -r .data.token | base64 --decode && echo

b) kubectl get svc -n kubernetes-dashboard

c) kubectl port-forward svc/kubernetes-dashboard -n kubernetes-dashboard 8443:8443

3) What to look for >

a) Services: name, labels, type, IP, internal endpoints, external endpoints (when type=LB)

b) Pods: CPU usage, mem usage, metadata, Resource infer, conditions
  - → No cpu/mem usage - check networking (for routing)
  - → High cpu/mem - pods overloaded, expand replicaset

c) Nodes: IP, no. of pods, status,

d) Edit YAML, look at ConfigMaps, look at secrets

4) Install kube-dash

a) kubectl apply -f <official kube dashboard git url>.yaml

Creates:
- namespace/kubernetes-dashboard
- serviceaccount/kubernetes-dashboard
- service/kubernetes-dashboard
- and more

b) If you dont have the token, create a new SA

1) kubectl create sa dashboard -n default
2) kubectl create clusterrolebinding dashboard-admin
   -n default --clusterrole=cluster-admin
   --serviceaccount=default:dashboard
3) Getting the secret

OLD kube:

kubectl get secret $(kubectl get serviceaccount dashboard
   -o jsonpath="{.secrets[0].name}")
   -o jsonpath="{.data.token}" | base64
   --decode

NEW kube way:

Create a secret.yaml with an annotation

apiVersion: v1
kind: secret
metadata:
   name: gman-kube-dashboard-sa-secret
   annotation:
      kubernetes.io/service-account.name: dashboard
type: kubernetes.io/service-account-token

kubectl apply secret.yaml
kubectl get secret gman-kube-dashboard-sa-secret -o yaml
then base 64 --decode the 'token' key

4) kubectl get svc -n kubernetes-dashboard
5) kubectl port-forward svc/kubernetes-dashboard 8443:443
   -n kubernetes-dashboard

local machine port

6) make sure you go to https://localhost:8443

7) On the UI, add the token, log in

5) Log viewing

a) kubectl logs <pod> -c <container name>
   * for stdout & stderr, go to page 84 1.a)
b) kube-dashboard (page 85-86)
c) EFK stack (page 82)
d) Grafana + Loki (page 83)
e) External Tools (docs on datadog/newrelic etc)
f) Pipe commands
   - kubectl logs <podname> -c <container-name> | (commands)

   pipe commands
   head -n5 : top 5 lines of output
   tail -n5 : bottom 5 lines of output
   more : page through the results on the screen
      useful for debugging.

g) container Options

- kubectl logs <pod-name> --previous

output from the LAST RUN of the pod

- kubectl logs <pod-name>

  => current pod output

h) Exec

If you KNOW the log path

kubectl exec <pod-name> cat /var/log/dpkg.log | head -10

preffered use: kubectl exec [POD] -- [COMMAND]

if you WANT TO GET INSIDE the pod

kubectl exec -it [POD] -- /bin/sh

Practical Troubleshooting

1) Fundamental routes

a) use kubectl to check status, logs, events

b) Use connectivity tools like ping, curl, traceroute, wget

c) check CNI configuration to see if there's N/W issues.

d) 3rd party kube tools

f) Network analysis (prop & more)

2) General approach

a) Check

- Service Selector Labels
- Resource Limits (if you have continuous restarts this could be)
- Ports (maybe you have the wrong ports)
- Image tags (do you see imagepullbackoff)
- Some YAML settings are missing

b) Metrics - too many metrics or too little metrics

c) Logs

d) Port forward & check for localhost (port)

e) Exec into the container to investigate

f) Force sleep in Dockerfile to look further.

> Force sleep in Dockerfile
\# comment out your app exec CMD

CMD ["sh", "-c", "tail -f /dev/null"]

3) Practical command approach

\# set alias to make commands simpler
alias k="kubectl" && alias kn='kubectl config set-context --current --namespace'

a) get pods & services across your namespace
kn gmam-ns
k get po,svc --show-labels -owide

→ if nothing seems wrong,

b) Port-forward the svc & hit the port on localhost
k port-forward svc/gmam-front 8888:80
my host ↑ ↑ svc port

→ 502 Bad Gateway
Look at label selectors, make sure they match

c) Issues with pod status
k describe pod <podname>, then look at events!

d) Issues with speed & traffic served: Look at CPU & Mem limits

> If you want to benchmark your endpoint:port combo, you can bake-in the following tools in your dockerfile
RUN apt-get install -y apache2-utils # for Ubuntu
RUN yum install httpd-tools # CentOS

→ Run ab -n 1000 -c 10 <your-service-url>
no. of requests concurrency level

Then analyse: Request/sec, Time per Req, Transfer rate

IF your app is NOT a webservice, you can use profiling tools like perf or gperftools

perf record -g ./game-binary
perf report

e) Check targetPort or selectors on the svc labels, usernames
f) Check ENV vars on the deployment YAMLs.

4) Cluster Component Failures

a) kubectl get componentstatuses
shows us the health status for the components in our cluster
> usually see 'scheduler' & 'component-manager' show up as healthy

b) kubectl describe node for status check

c) check scheduler logs & System Logs (on node)
d) check Network Connectivity (on the node via SSH)

5) Cluster Logs
a) Control plane logs
/var/log/kube-apiserver.log
/var/log/kube-scheduler.log
/var/log/kube-controller-manager.log

b) Worker node Logs
/var/log/kubelet.log
/var/log/kube-proxy.log

> some file systems use journalctl

c) Journal & Journalctl
- journal is a centralized logging system implemented by systemd, and replaces the traditional flat text log files with a binary format.

- usage
> journalctl = view all logs
> journalctl -u <service_name> = logs for particular service
> journalctl --since "yyyy-mm-dd HH:MM:SS" = time filter
> journalctl -f = follow realtime logs.

6) Causes
- API server shutdown or crashing
- API server losing storage
- services VM down/crashing
- Node shutdown
- Network Issues
- kubelet software fault
- cluster operation error

7) Mitigations
- Restart VM
- Change storage
- use High Availability Configuration
- Snapshot API server volumes
- App designs to be fault tolerant

8) Component Failure Practicals

→ Scenario: control-plane not ready
→ get pods tells us status is terminating for bunch of pods across many namespaces.
→ Describe node with bad status, look for:
Conditions:
memoryPressure
DiskPressure
PIDPressure
Ready
} kubelet may have stopped on node
→ check componentstatuses, if fine, that means the control plane is responding
→ SSH into the node
↳ check disk using : df -h ✓
check processes using : htop ✓
try to look for metrics on:
1) /usr/local/bin/kube-apiserver
2) /usr/local/bin/etcd
3) /usr/local/bin/kube-controller-manager
4) /usr/local/bin/kube-scheduler
5) /usr/local/bin/kubelet
6) /usr/local/bin/kube-proxy
7) /usr/bin/docker or /usr/bin/containerd
8) /usr/systemd/system/... stuff

→ Try restarting process
1) Check ExecStartPre=/sbin/modprobe br_netfilter
ExecStartPre=/sbin/modprobe overlay } Purge (97)
2) stop service
systemctl stop kubelet
3) start
systemctl start kubelet

> when you wanna check logs: cat /var/log/kube-scheduler.log

→ check syslog : tail -n10 /var/log/syslog
→ check kernel log: tail -n10 /var/log/kern.log

→ If all fails, restart the whole cluster, then describe the control-plane node and look at all the bootstrap step in the Events: section

> modprobe is a linux utility used to manage kernel modules, which are dynamically loadable/unloadable, which can be added to/removed from the kernel, on the fly, without system reboot
ADD: modprobe <module-name>
REMOVE: modprobe -r <module-name>
LIST modules: lsmod ; INFO: modinfo <module-name>
LOCATION: /sbin/modprobe

8) Component Failure Practicals

→ scenario: control-plane not ready
→ get pods tells us status is terminating for bunch of pods across many namespaces.
→ Describe node with bad status, look for:
Conditions:
memoryPressure
DiskPressure
PIDPressure
Ready
} kubelet may have stopped on node

→ check componentstatuses, if fine, that means the control plane is responding
→ ssh into the node
↳ check disk using: df -h ✓
check processes using: htop ✓
try to look for metrics on:
1) /usr/local/bin/kube-apiserver
2) /usr/local/bin/etcd
3) /usr/local/bin/kube-controller-manager
4) /usr/local/bin/kube-scheduler
5) /usr/local/bin/kubelet
6) /usr/local/bin/kube-proxy
7) /usr/bin/docker or /usr/bin/containerd
8) /usr/systemd/system/.. stuff

→ Try restarting process
1) Check ExecStartPre=/sbin/modprobe br_netfilter
ExecStartPre=/sbin/modprobe overlay } purge (7)
2) stop service
systemctl stop kubelet
3) start
systemctl start kubelet

> when you wanna check logs: cat /var/log/kube-scheduler.log

→ check syslog: tail -n10 /var/log/syslog
→ check kernel log: tail -n10 /var/log/kern.log

→ If all fails, restart the whole cluster, then describe the control-plane node and look at all the bootstrap steps in the Events: section.

> modprobe is a linux utility used to manage kernel modules, which are dynamically loadable/unloadable, which can be added to/removed from the kernel on the fly, without system reboot.
ADD: modprobe <module-name>
REMOVE: modprobe -r <module-name>
LIST modules: lsmod | INFO: modinfo <module-name>
LOCATION: modprobe is located at /sbin/modprobe

9) Network Troubleshooting

a) Check the status and logs of
Pods, nodes, services, network Policies

b) NW tools
ping, traceroute, wget, telnet, nc, tcpdump, iptables, etc.

c) Inspect NW namespaces, interfaces, routes, DNS configs of pods and nodes.
ifconfig, route, nslookup, dig, ip

d) Check the configuration & logs of CNI plugin & kube-proxy component on the nodes.
cat /var/log/cni.log

d) validate cluster installation (if configured by you)
kubeadm, kops, kubespray
$ kubeadm config view or $ kubeadm init phase preflight

e) diagnostic tools for in-cluster
kubectl-debug, ksniff, kube-netc, kube-netdiag

implementation @ Page 58 12.a)

Note on br-netfilter & overlay kernel modules

a) br-netfilter
- provides support for iptables filtering in the Linux kernel's bridge implementation.
- allows for bridge specific iptable rules, allowing for network filtering
- In k8s, container runtimes rely on Linux bridges, ensuring that iptable rules can be applied to network policies.
  - like allow/deny traffic between pods

[docker0]

b) overlay
- storage & networking driver used in container runtimes
- Facilitates the creation of overlay filesystems for containers & enables container networking across multiple nodes in k8s cluster.
- This is crucial to allow pods to communicate with each other across different nodes, by abstracting the underlying network infrastructure, providing a virtual network overlay that spans the entire cluster.

Network Policy example @ : Page 23

10) Network tools Usage

a) ping : test nw connect of a host by sending icmp echo request

use : ping google.com

b) traceroute : traces route that packet takes to reach destination, showing IP's along the way

use : traceroute google.com

c) wget : download files from the internet (HTTP, FTP, HTTPS)

use : wget https://example.com/file.txt

d) telnet : comms with other host using Telnet protocol

use : telnet example.com 80 ← port number

e) nc : netcat reading from & writing to TCP/UDP connections

use : nc -zv example.com 80

f) tcpdump : packet analyser that allows user to display TCP, UDP, and other packets

use : tcpdump -i eth0 tcp port 80

interface

protocol

g) iptables : setup, maintain and inspect the tables of IP packet filter rules in the Linux kernel.

use iptables -A INPUT -p tcp -dport 22 -j ACCEPT

h) route : show/edit IP routing tables

use : route -n

i) nslookup : query DNS servers and obtain info about domain names and IP addresses.

use : nslookup example.com

j) dig : used to interrogate DNS name servers.

use : dig example.com

k) ip : show/edit/modify routing, devices, policy routing and tunnels

use : ip address show

l) kops : create, update, delete K8s on AWS

m) kubectl debug : K8s plugin to troubleshoot pods by opening an interactive SHELL

use : kubectl debug <podname> -it -- bash

n) ksniff : sniff K8s network traffic on a specific pod

use: ksniff -p <podname> -n gmon-ns

o) kube-netc : capture network traffic for a specific pod

use: kube-netc -p <podname> -n gmon-ns

p) kube-netdiag : tool to diagnose K8s NW Issues

use kube-netdiag -p <podName> -n gmon-ns

output network interfaces:
- eth(N) <IP>

Routes :
- Destination : IP
- Gateway : IP
- Interface :

DNS Resolution:
- Resolving <address-url> <IP>

Connectivity Tests:
- Pinging 8.8.8.8 <success/fail>
- Connecting to <url><port> <success/fail>

Symantic Versioning

Semantic versioning is a scheme used primarily to convey information and interpretation of the nature of changes in a release.

<MAJOR>.<MINOR>.<PATCH>

→ MAJOR : incremented when there are incompatible changes that break existing functionality

→ MINOR : incremented when new features are added in a backwards-compatible manner

→ PATCH : incremented when changes are backwards-compatible bug fixes

example:
Package first built: 1.0.0
→ New feature that is backward compatible : 1.1.0
→ Bug fix : 1.1.1
→ Breaking change : 2.0.0

PAGE No. 100
DATE / /

n) ksniff : sniff K8s network traffic on a specific pod

use: ksniff -p <podname> -n gman-ns

o) kube-netc : capture network traffic for a specific pod

use: kube-netc -p <podname> -n gman-ns

p) kube-netdiag : tool to diagnose K8s NW issues

use: kube-netdiag -p <podName> -n gman-ns

output: Network interfaces:
- eth(N) <IP>

Routes:
- Destination : IP
- Gateway : IP
- Interface :

DNS Resolution:
- Resolving <address-url> <IP>

Connectivity Tests:
- Pinging 8.8.8.8 <success/fail>
- connecting to <url><port> <success/fail>

Symantic Versioning

PAGE No. 101
DATE 06/03/2024

Semantic versioning is a scheme used primarily to convey information and interpretation of the nature of changes in a release

<MAJOR>.<MINOR>.<PATCH>

→ MAJOR: incremented when there are incompatible changes that break existing functionality

→ MINOR: incremented when new features are added in a backwards-compatible manner.

→ PATCH: incremented when changes are backwards-compatible bug fixes.

example:
Package first built: 1.0.0
→ New feature that is backward compatible: 1.1.0
→ Bug fix : 1.1.1
→ Breaking change: 2.0.0