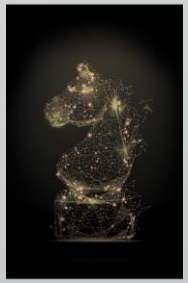
K8s Admission Controllers

Nishaanth Guna

Security Consultant, MDSec



Agenda

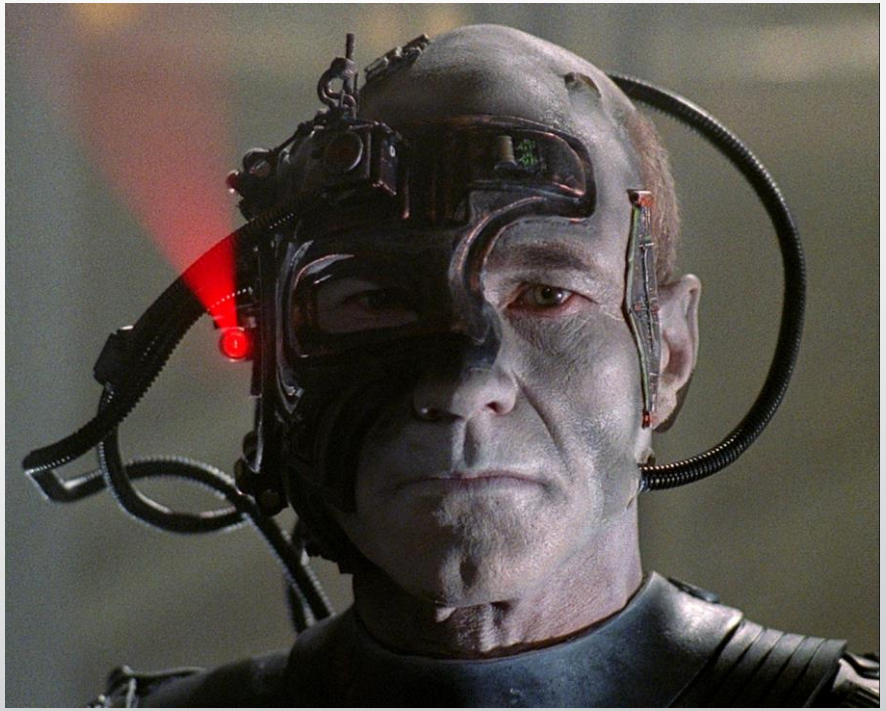
- What is K8s and why?
- Desired state principle
- Components in K8s
- K8s Architecture
- Setting up K8s locally
- Common misconfigurations
- Admission Controller
- Writing Controller logic w Webhooks
- Setup production-grade cluster w Vagrant

Kuberwhat?!!

- k(j)uːbərˈnɛtɪs, -ˈneɪtɪs, -ˈneɪtiːz, -ˈnɛtiːz
- koo-ber-net-ees
- K8s (8 letters between K and s)
- κυβερνήτης / kubernḗtēs: Greek for "steersman, navigator" or "guide"
- Also the etymological root of cybernetics

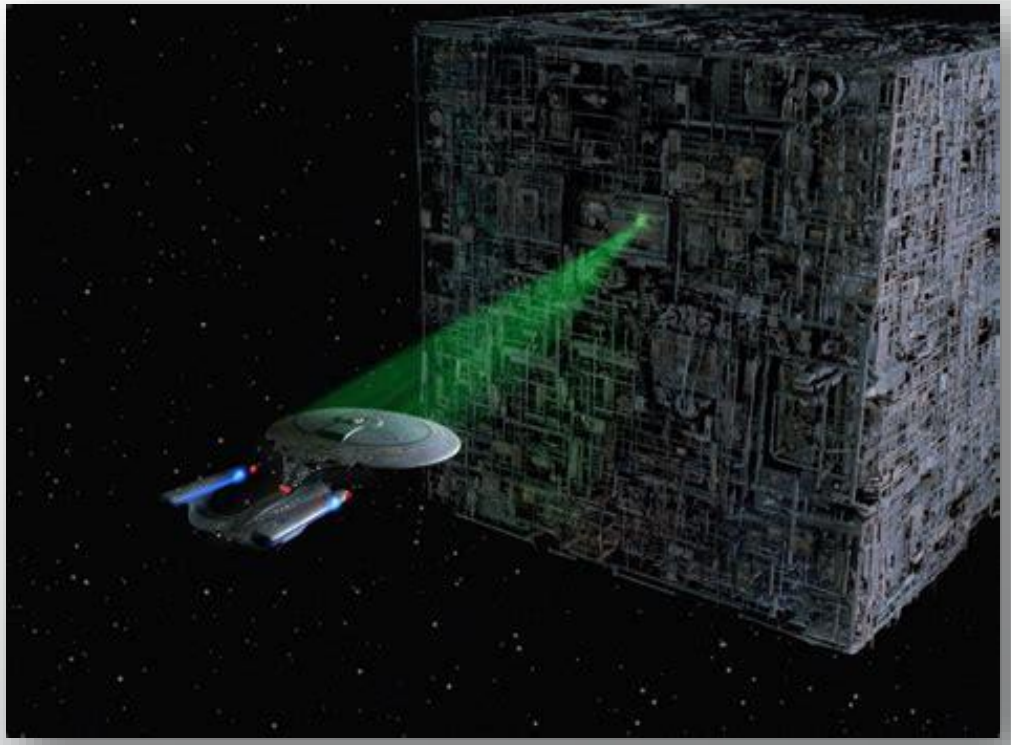
What is it?

- "Kubernetes, also known as K8s, is an open-source orchestration system for automating deployment, scaling, and management of containerised applications."
- Originated from "Project Borg"
- Originally designed by Google
- Redundancy (like the Borg)
- Open-sourced in 2014



Why K8s?

- Move away from monolith architecture
- K8s uses microserver architecture
- Reduce/almost no downtime
- Lose a ~~Borg cube~~ kube? Not an issue
- Easy to orchestrate a large number of containers
- Relies on desired state principle

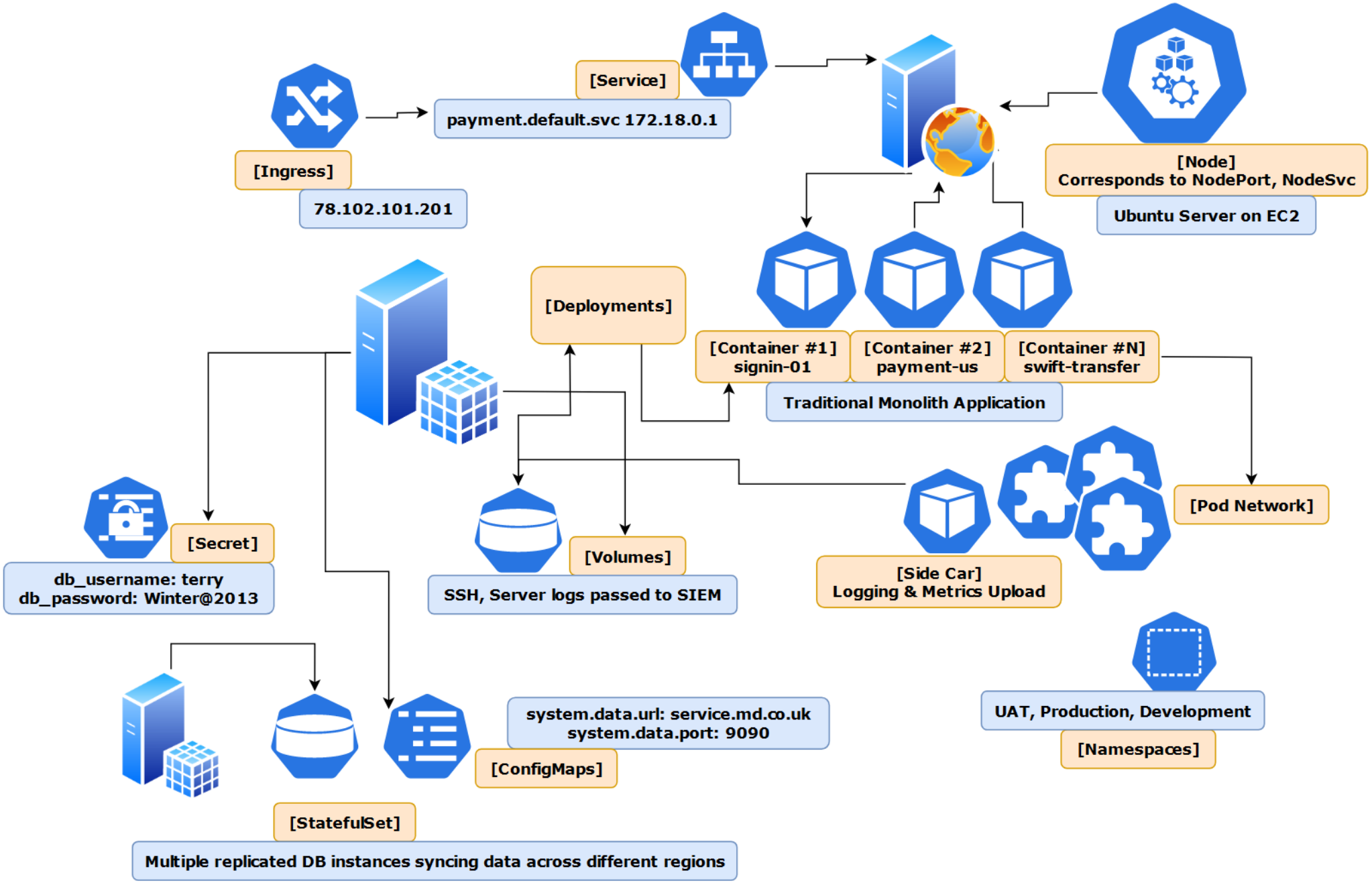


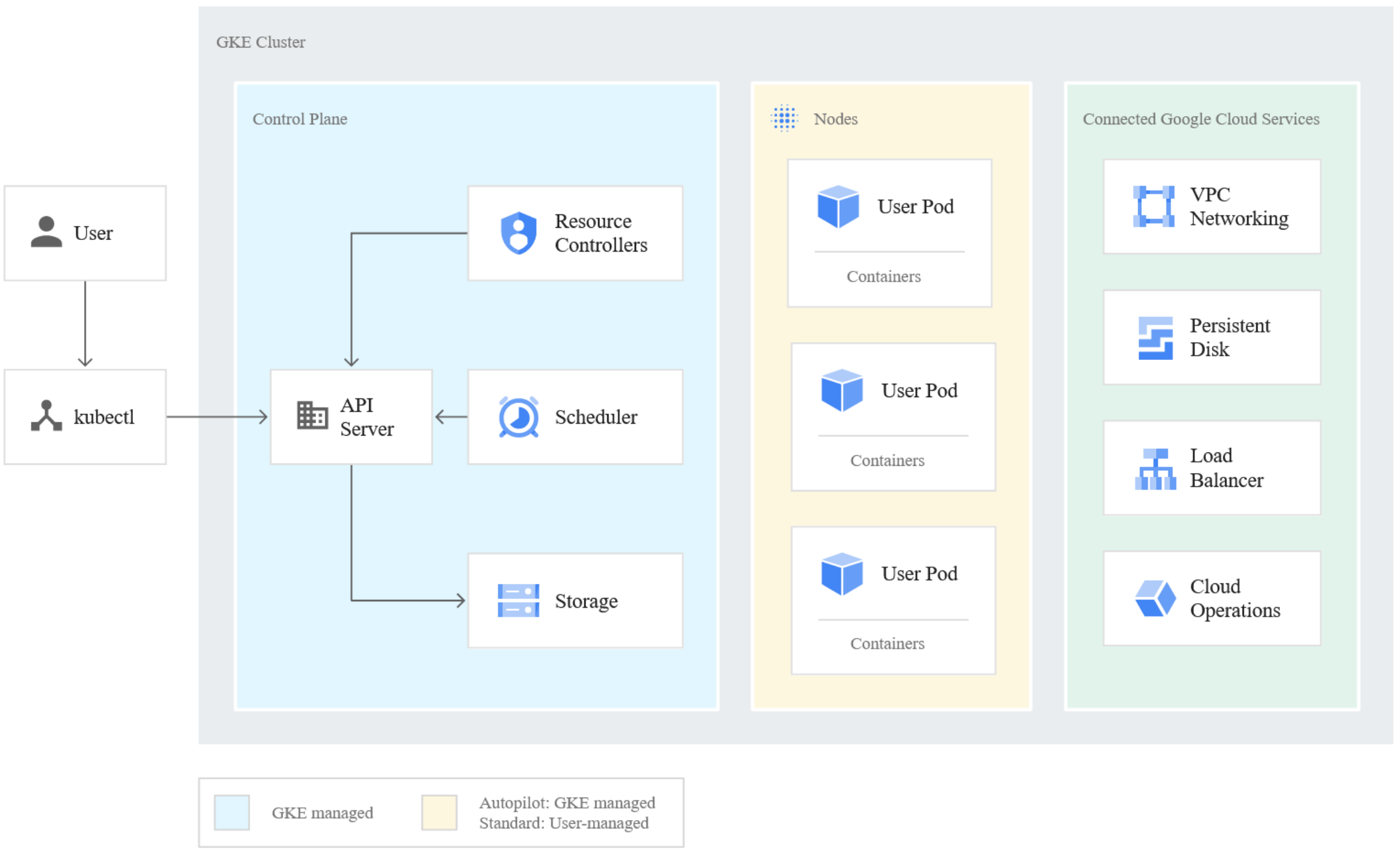
Desired state principle

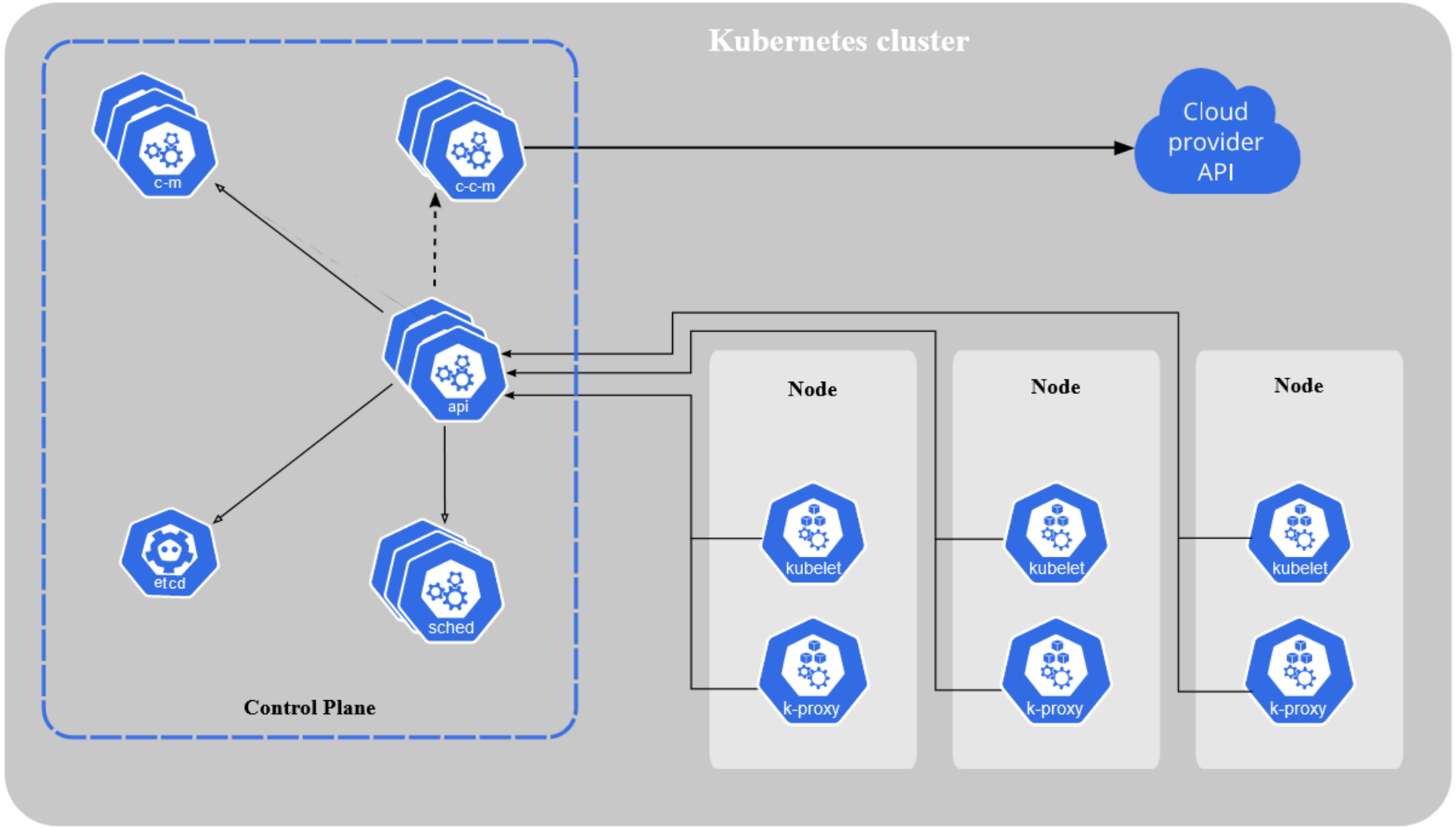
- Any created object will run the exact number of times specified, at any given point
- If a container crashes, the *kube-controller-manager* should detect this
- There would then be a mismatch between desired state and actual state
- A new container will be launched to obtain desired state

ggwp@metaaoh: ~

```
GNU nano 6.2
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: web
  labels:
    env: dev
    role: web
spec:
  replicas: 4
  selector:
    matchLabels:
      role: web
  template:
    metadata:
      labels:
        role: web
    spec:
      containers:
      - name: testnginx
        image: nginx
```


[Service]
payment.default.svc 172.18.0.1
[Ingress]
78.102.101.201
[Node]
Corresponds to NodePort, NodeSvc
Ubuntu Server on EC2
[Deployments]
[Container #1]
signin-01
[Container #2]
payment-us
[Container #N]
swift-transfer
Traditional Monolith Application
[Pod Network]
[Secret]
db_username: terry
db_password: Winter@2013
[Volumes]
SSH, Server logs passed to SIEM
[Side Car]
Logging & Metrics Upload
UAT, Production, Development
[Namespaces]
system.data.url: service.md.co.uk
system.data.port: 9090
[ConfigMaps]
[StatefulSet]
Multiple replicated DB instances syncing data across different regions


GKE Cluster
Control Plane
User
kubectl
API Server
Resource Controllers
Scheduler
Storage
Nodes
User Pod
Containers
User Pod
Containers
User Pod
Containers
Connected Google Cloud Services
VPC Networking
Persistent Disk
Load Balancer
Cloud Operations
GKE managed
Autopilot: GKE managed
Standard: User-managed



API server

Cloud controller manager *(optional)*

Controller manager

etcd (persistence store)

kubelet

kube-proxy

Scheduler

Control plane

Node

Setting up K8s locally

- K8S can be setup locally with a minimal set-up for testing. To start a local cluster, need to install docker and minikube binary. Minikube creates a VM and runs master, worker nodes in the same place.
- Once the initial setup is done, start the docker daemon and then minikube. Is not production-ready.

```
$ curl -LO
https://storage.googleapis.com/minikube/releases/latest/minik
ube-linux-amd64 && sudo install minikube-linux-amd64
/usr/local/bin/minikube
$ curl -L –s
https://dl.k8s.io/release/stable.txt/bin/linux/amd64/kubectl
$ sudo groupadd docker
$ sudo usermod -aG docker $USER && newgrp docker
$ sudo systemctl start docker && minikube start
```

- Deploying everything in *default* namespace – Default NS should not be used for production since the default service token can be abused to get untethered access to all the resources in the cluster, defeats the principle of ns isolation.
- Bad Pods - Containers with the ability to privilege escalate, escape to the host OS using a shared socket/folder/IPC/network. Running containers with *CAP_SYS_ADMIN* permission allows root access with no cgroup limits.
- Malicious Images – In the event of a compromise, if an attacker can load a malware image from externally hosted registry or poison the DevOps pipeline via Jenkins
- Insecure RBAC – Since companies use custom permissions for resources and there is no user accounts out-of-the box from K8S (OIDC providers for provisioning AD accounts), RBAC can get tricky.
- Plugins – Abusing weak RBAC on the network plugins + container escape + service token abuse == Cluster Admin!

```
  GNU nano 6.2
FROM ubuntu:latest

RUN apt-get update -y && apt-get install -y python3-pip
RUN pip3 install flask

WORKDIR /app

COPY gkeeper.py /app
RUN mkdir certs
COPY certs/gkeeperkey.pem /app/certs/gkeeperkey.pem
COPY certs/gkeepercrt.pem /app/certs/gkeepercrt.pem

CMD python3 gkeeper.py
```

Initial Access

pa[redacted] · Documentation/componen[redacted] reStructuredText · master

```
84    kubeadm join --token <token> <master-ip>:<master-port> \
85        --discovery-token-ca-cert-hash sha256:<hash>
86
135   sudo kubeadm join --token 85856d[redacted] 130.192.[redacted]
136       --discovery-token-ca-cert-hash sha256:2c3f07b126bdc772e113306f1082ece6c406[redacted]
137
```

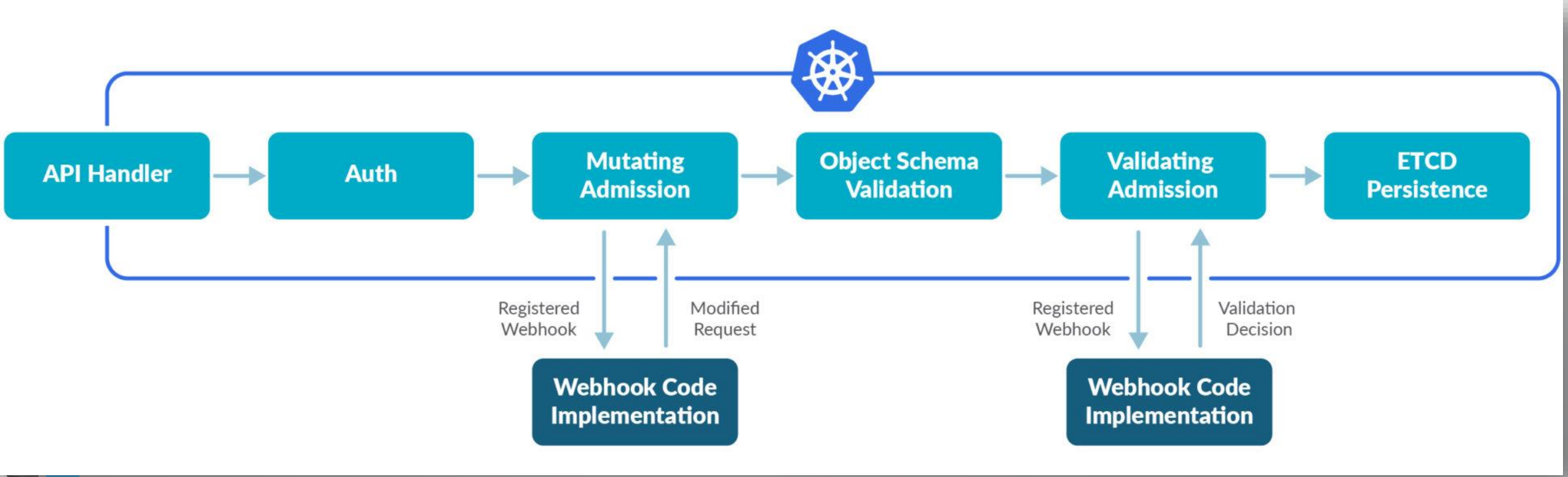
```
vagrant@enron:~$ ls -la /etc/kubernetes/admin.conf
-rw-------  1 root root 5637 Jul  7 08:46 /etc/kubernetes/admin.conf
vagrant@enron:~$
```

```
  GNU nano 6.2
FROM ubuntu:latest

RUN apt-get update -y && apt-get install -y python3-pip
RUN pip3 install flask

WORKDIR /app

COPY gkeeper.py /app
RUN mkdir certs
COPY certs/gkeeperkey.pem /app/certs/gkeeperkey.pem
COPY certs/gkeepercrt.pem /app/certs/gkeepercrt.pem

CMD python3 gkeeper.py
```

```
  GNU nano 6.2
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0 ...
    server: https://100.10.121.111:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: colby
  name: colby-context
current-context: colby-context
kind: Config
preferences: {}
users:
- name: colby
  user:
    client-certificate: /home/colby/.certs/colby.cert
    client-key: /home/colby/.certs/colby.key # echo this | base64
```

Admission Controllers

- Gatekeeps the cluster from object writes to ETCD if 'X' does not conform with a policy (e.g., images not pulled from internal.mdsec.docker.registry.com, do not run any container as root)
- Possible to create HTTP callbacks/webhooks with custom logic (e.g., flask web application) to decide if a resource should be let in.
- Mutating and Validating Controllers
- Possible to intercept an API request and PATCH the request to manipulate a struct or an object in it.
- 30+ shipped with K8S, compiled into the kube-apiserver binary
- NameSpaceLifecycle, PodSecurityPolicy, ValidatingAdmissionWebhook and MutatingAdmissionWebhook (dynamic), ServiceAccount
- Flask Server monitoring for webhooks -> Deploy as a SVC -> Register a Webhook Controller -> Test


API Handler
Auth
Mutating Admission
Object Schema Validation
Validating Admission
ETCD Persistence
Registered Webhook
Modified Request
Registered Webhook
Validation Decision
Webhook Code Implementation
Webhook Code Implementation

https://github.com/kubernetes/api/blob/master/admission/v1/types.go 120%

master | api / admission / v1 / types.go | Top

Code | Blame | 169 lines (148 loc) · 9.16 KB | Raw

```go
type AdmissionRequest struct {
	// UID is an identifier for the individual request/response. It allows us to distinguish instances of requests which are
	// otherwise identical (parallel requests, requests when earlier requests did not modify etc)
	// The UID is meant to track the round trip (request/response) between the KAS and the WebHook, not the user request.
	// It is suitable for correlating log entries between the webhook and apiserver, for either auditing or debugging.
	UID types.UID `json:"uid" protobuf:"bytes,1,opt,name=uid"`
	// Kind is the fully-qualified type of object being submitted (for example, v1.Pod or autoscaling.v1.Scale)
	Kind metav1.GroupVersionKind `json:"kind" protobuf:"bytes,2,opt,name=kind"`
	// Resource is the fully-qualified resource being requested (for example, v1.pods)
	Resource metav1.GroupVersionResource `json:"resource" protobuf:"bytes,3,opt,name=resource"`
	// SubResource is the subresource being requested, if any (for example, "status" or "scale")
	// +optional
	SubResource string `json:"subResource,omitempty" protobuf:"bytes,4,opt,name=subResource"`

	// RequestKind is the fully-qualified type of the original API request (for example, v1.Pod or autoscaling.v1.Scale).
	// If this is specified and differs from the value in "kind", an equivalent match and conversion was performed.
	//
	// For example, if deployments can be modified via apps/v1 and apps/v1beta1, and a webhook registered a rule of
	// `apiGroups:["apps"], apiVersions:["v1"], resources: ["deployments"]` and `matchPolicy: Equivalent`,
	// an API request to apps/v1beta1 deployments would be converted and sent to the webhook
	// with `kind: {group:"apps", version:"v1", kind:"Deployment"}` (matching the rule the webhook registered for),
	// and `requestKind: {group:"apps", version:"v1beta1", kind:"Deployment"}` (indicating the kind of the original API reque
	//
```

Symbols

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols

- class AdmissionReview
- class AdmissionRequest
- class AdmissionResponse
- type PatchType
- const PatchTypeJSONPatch
- type Operation
- const Create
- const Update
- const Delete
- const Connect

https://github.com/kubernetes/api/blob/master/admission/v1/types.go

master api / admission / v1 / types.go

Top

Code Blame 169 lines (148 loc) · 9.16 KB

Raw

```go
type AdmissionResponse struct {
	// UID is an identifier for the individual request/response.
	// This must be copied over from the corresponding AdmissionRequest.
	UID types.UID `json:"uid" protobuf:"bytes,1,opt,name=uid"`

	// Allowed indicates whether or not the admission request was permitted.
	Allowed bool `json:"allowed" protobuf:"varint,2,opt,name=allowed"`

	// Result contains extra details into why an admission request was denied.
	// This field IS NOT consulted in any way if "Allowed" is "true".
	// +optional
	Result *metav1.Status `json:"status,omitempty" protobuf:"bytes,3,opt,name=status"`

	// The patch body. Currently we only support "JSONPatch" which implements RFC 6902.
	// +optional
	Patch []byte `json:"patch,omitempty" protobuf:"bytes,4,opt,name=patch"`

	// The type of Patch. Currently we only allow "JSONPatch".
	// +optional
	PatchType *PatchType `json:"patchType,omitempty" protobuf:"bytes,5,opt,name=patchType"`

```

```yaml
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: "pod-policy.example.com"
webhooks:
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   [""]
    apiVersions: ["v1"]
    operations:  ["CREATE"]
    resources:   ["pods"]
    scope:       "Namespaced"
  clientConfig:
    service:
      namespace: "example-namespace"
      name: "example-service"
    caBundle: <CA_BUNDLE>
  admissionReviewVersions: ["v1"]
  sideEffects: None
  timeoutSeconds: 5
```

vagrant@enron: ~

```
  GNU nano 6.2                    /etc/kubernetes/manifests/kube-apiserver.yaml *
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.13.37:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.13.37
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction,ValidatingAdmissionPolicy,NamespaceLifecycle
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
```

```
vagrant@enron:~$ ls -ls /etc/kubernetes/manifests/
total 16
4 -rw------- 1 root root 2427 Jul  7 08:46 etcd.yaml
4 -rw------- 1 root root 4038 Jul  7 08:46 kube-apiserver.yaml
4 -rw------- 1 root root 3544 Jul  7 08:46 kube-controller-manager.yaml
4 -rw------- 1 root root 1463 Jul  7 08:46 kube-scheduler.yaml
vagrant@enron:~$
```

```
vagrant@enron:~$ kubectl get pods
E0706 23:51:21.965303   10388 memcache.go:265] couldn't get current server API group list: Get "http
s://192.168.13.37:6443/api?timeout=32s": dial tcp 192.168.13.37:6443: connect: connection refused
E0706 23:51:21.967860   10388 memcache.go:265] couldn't get current server API group list: Get "http
s://192.168.13.37:6443/api?timeout=32s": dial tcp 192.168.13.37:6443: connect: connection refused
E0706 23:51:21.969893   10388 memcache.go:265] couldn't get current server API group list: Get "http
s://192.168.13.37:6443/api?timeout=32s": dial tcp 192.168.13.37:6443: connect: connection refused
E0706 23:51:21.973042   10388 memcache.go:265] couldn't get current server API group list: Get "http
s://192.168.13.37:6443/api?timeout=32s": dial tcp 192.168.13.37:6443: connect: connection refused
E0706 23:51:21.974415   10388 memcache.go:265] couldn't get current server API group list: Get "http
s://192.168.13.37:6443/api?timeout=32s": dial tcp 192.168.13.37:6443: connect: connection refused
The connection to the server 192.168.13.37:6443 was refused - did you specify the right host or port
?
```

```
vagrant@enron:~$ sudo crictl ps
CONTAINER           IMAGE                                                                   CREATED
 STATE               NAME                      ATTEMPT             POD ID                   POD
b7c60c537ae41       08a0c939e61b7340db53ebf07b4d0e908a35ad8d94e2cb7d0f958210e567079a   2 seconds ago
 Running             kube-apiserver            0                   014e21e323019            kube-apiserver-enro
n.corp.k8s.local
6a04422cd33c9       7cffc01dba0e151e525544f87958d12c0fa62a9f173bbc930200ce815f2aaf3f   9 seconds ago
 Running             kube-controller-manager   3                   14fd03bd92d96            kube-controller-man
ager-enron.corp.k8s.local
c1dc74e74d2ff       41697ceeb70b3f49e54ed46f2cf27ac5b3a201a7d9668ca327588b23fafdf36a   18 seconds ago
 Running             kube-scheduler            3                   c2b429989ca16            kube-scheduler-enro
n.corp.k8s.local
c8f23e4eb412e       ead0a4a53df89fd173874b46093b6e62d8c72967bbf606d672c9e8c9b601a4fc   2 hours ago
 Running             coredns                   0                   4749512bd370c            coredns-5d78c9869d-
d84gr
a0481fc08b212       ead0a4a53df89fd173874b46093b6e62d8c72967bbf606d672c9e8c9b601a4fc   2 hours ago
```

```yaml
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
address: "192.168.0.8"
port: 20250
serializeImagePulls: false
evictionHard:
    memory.available:  "200Mi"
```

$ kubectl apply -f clusterconfig.yaml

kubeadm-config.yaml

```yaml
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
kubernetesVersion: v1.14.3 # change according to kubeadm supported version
apiServer:
  certSANs:
  - 127.0.0.1
  - cluster-api.example.com # change according to your requirements
  extraArgs:
    authorization-mode: Node,RBAC
    feature-gates: "TTLAfterFinished=true"
    audit-policy-file: "/etc/kubernetes/audit-policy.yaml"
    audit-log-path: "/etc/kubernetes/audit/kube-apiserver-audit.log"
    audit-log-maxage: "2"
  extraVolumes:
  - name: "audit-policy"
    hostPath: "/etc/kubernetes/audit-policy.yaml"
    mountPath: "/etc/kubernetes/audit-policy.yaml"
    readOnly: false
    pathType: File
  - name: "audit-volume"
    hostPath: "/var/log/kubernetes/audit"
    mountPath: "/etc/kubernetes/audit"
    readOnly: false
    pathType: DirectoryOrCreate
  timeoutForControlPlane: 4m0s
```

```
$ docker build -t gkeeper-beta:1.2 .
Sending build context to Docker daemon   25.6kB
Step 1/9 : FROM ubuntu:latest
 ---> 5a81c4b8502e
Step 2/9 : RUN apt-get update -y && apt-get install -y python3-pip
 ---> Using cache
 ---> 54aaddaecd17
Step 3/9 : RUN pip3 install flask
 ---> Using cache
 ---> 60dee29ec023
Step 4/9 : WORKDIR /app
 ---> Using cache
 ---> 799ee02b6551
Step 5/9 : COPY gkeeper.py /app
 ---> 43cca7ac3126
Step 6/9 : RUN mkdir certs
 ---> Running in 968749ba15a9
Removing intermediate container 968749ba15a9
 ---> a59377bfb054
```

```
ggwp@metaaoh:$ docker tag gkeeper-beta:1.2 nishaanthguna/gk
eta:1.2
ggwp@metaaoh:$ docker push nishaanthguna/gkeeper-beta:1.2
The push refers to repository [docker.io/nishaanthguna/gkeeper-beta]
e049c69854bb: Pushed
53c9ef2838df: Pushed
5cb4e6b0b1da: Pushed
e6988f57e2c5: Pushed
bdfd10f9841e: Layer already exists
5dadf87db00f: Layer already exists
4cb764170a2f: Layer already exists
59c56aee1fb4: Layer already exists
1.2: digest: sha256:7694ddfcce658d88397122ddade5b70aaea73eb295930b88533721ef049be271 size: 1989
```

```
GNU nano 6.2                                                    register-controller
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  name: validating-webhook
webhooks:
  - name: validate.default.svc
    failurePolicy: Fail
    sideEffects: None
    admissionReviewVersions: ["v1","v1beta1"]
    rules:
      - apiGroups: ["apps", ""]
        resources:
          - "deployments"
        apiVersions:
          - "*"
        operations:
          - CREATE
    clientConfig:
      service:
        name: validate
        namespace: default
        path: /validate
      caBundle: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURLVENDQWhHZ0F3SUJBZ0lVQ
```

```
GNU nano 6.2
apiVersion: v1
kind: Service
metadata:
  name: validate
spec:
  selector:
    app: validate
  ports:
  - port: 443
```

```
  GNU nano 6.2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: validation-webhook
  labels:
    app: validate
spec:
  replicas: 1
  selector:
    matchLabels:
      app: validate
  template:
    metadata:
      labels:
        app: validate
    spec:
      containers:
      - name: webhook
        image: nishaanthguna/gkeeper-beta:1.2
        ports:
        - containerPort: 443
```

```
-rwxrwxrwx 1 vagrant vagrant  193 Jul  6 03:00 gitlab-runner.yaml
vagrant@enron:/vagrant/manifests$ kubectl apply -f webhook.yaml
deployment.apps/validation-webhook created
vagrant@enron:/vagrant/manifests$ kubectl apply -f svc.yaml
service/validate created
vagrant@enron:/vagrant/manifests$ kubectl apply -f register-controller.yaml
validatingwebhookconfiguration.admissionregistration.k8s.io/validating-webhook created
vagrant@enron:/vagrant/manifests$
```

```
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  5m15s  default-scheduler  Successfully assigned default/validation-webhook-766695796f-5dfj7 to enron.corp.k8s.local.e39e305
c
  Normal  Pulling    5m15s  kubelet            Pulling image "nishaanthguna/gkeeper-beta:1.2"
  Normal  Pulled     16s    kubelet            Successfully pulled image "nishaanthguna/gkeeper-beta:1.2" in 4m58.089296916s (4m58.08931001s inc
luding waiting)
  Normal  Created    16s    kubelet            Created container webhook
  Normal  Started    15s    kubelet            Started container webhook
vagrant@enron:/vagrant/manifests$
```

https://github.com/k-mitevski/kubernetes-validating-webhook/blob/master/validate.py

kubernetes-validating-webhook / validate.py

Top

Code Blame 43 lines (31 loc) · 1.64 KB

Raw

```python


if "LABEL" not in environ:
    webhook.logger.error("Required environment variable for label isn't set. Exiting...")
    exit(1)


@webhook.route('/validate', methods=['POST'])
def validating_webhook():
    request_info = request.get_json()
    uid = request_info["request"].get("uid")

    if request_info["request"]["object"]["metadata"]["labels"].get(webhook.config['LABEL']):
        webhook.logger.info(f'Object {request_info["request"]["object"]["kind"]}/{request_info["request"]["object"]["met
        return admission_response(True, uid, f"{webhook.config['LABEL']} label exists.")
    else:
        webhook.logger.error(f'Object {request_info["request"]["object"]["kind"]}/{request_info["request"]["object"]["me
        return admission_response(False, uid, f"The label \"{webhook.config['LABEL']}\" isn't set!")
```

```python
@gkeeper.route('/validate', methods=['POST'])
def webhook_handler():
    kube_request = request.get_json()
    uid = kube_request["request"].get("uid")
    container = kube_request["request"]["object"]["spec"]["containers"]

    try:
        if (container.image != env.ALLOWED_REGISTRIES):
            return webhook_response(True, uid, "Pod pulling image from trusted source")
    except:
        return webhook_response(False, uid, "Not allowed to pull images from arbitraty registries")
    return webhook_response(False, uid, "Can't pull the image from the registry")

```

```python
def webhook_response(allowed, uid, message):
    return jsonify(
        {
            "apiVersion": "admission.k8s.io/v1",
            "kind": "AdmissionReview",
            "response":
            {
                "allowed": allowed,
                "uid": uid,
                "status": {
                    "message": message
                }
            }
        }

    )

```

```
$ kubectl apply –f insecure-registry-pod.yaml
Error from server: error when creating “insecure-registry-pod.yaml”: admission webhook from “enron.whitelist-private-reg” denied the request: nginx image comes from an untrusted registry! (docker.io/nginx:latest). Only images from https://2122029192.amazonaws.com
```

```
$ kubectl apply –f secure-registry-pod.yaml
pod/nginx created
```

```
$ diff -w test-webhook.yaml test-webhook-1.yaml
10c10
<       image: nginx
---
>       image: 101290192019.dkr.ecr.us-east-1.amazonaws.com/nginx
```

Vagrantfile

```ruby
Vagrant.configure("2") do |config|

    config.vm.define "k8s-master" do |master|
        master.vm.box = IMAGE_NAME
        master.vm.network "private_network", ip: API_SERVER_IP
        master.vm.hostname = "#{CLUSTER_NAME}-k8s-control-plane"
        master.vm.provider :virtualbox do |vb|
            vb.name = "#{CLUSTER_NAME}-k8s-control-plane"
            vb.memory = VM_MEMORY
            vb.cpus = 2
        end
        master.vm.provision "master-common", type: "shell",
         env: {
            "API_SERVER_IP": API_SERVER_IP,
            "DNS_SERVER": DNS_SERVER,
            "KUBERNETES_VERSION": KUBERNETES_VERSION,
            "CRI_VERSION": CRI_VERSION,
            "CLUSTER_NAME": CLUSTER_NAME,
            "OS": OS
         },
         path: "init/common.sh"
         master.vm.provision "cluster-start", type: "shell",
         env: {
            "API_SERVER_IP": API_SERVER_IP,
            "POD_CIDR": POD_CIDR,
            "CLUSTER_NAME": CLUSTER_NAME,
            "SERVICE_CIDR": SERVICE_CIDR
         },
         path: "init/master.sh"
        #  master.vm.provision "resource-deploy", type: "shell", path: "init/deploy.sh"
    end

    (1..WORKER_NODES).each do |i|
        config.vm.define "k8s-worker-#{i}" do |node|
            node.vm.box = IMAGE_NAME
            node.vm.network "private_network", ip: "#{WORKER_IP_RANGE}.#{i + 37}"
```

Ln 44, Col 1 (89 selected) Spac

init > $ common.sh

```
apt update -y
apt install cri-o cri-o-runc -y
systemctl daemon-reload
systemctl enable crio --now

if [ ! -d /etc/modules.load.d/ ]; then
    mkdir /etc/modules.load.d/
fi
cat <<EOF | sudo tee /etc/modules.load.d/crio.conf
overlay
br_netfilter
EOF

modprobe overlay
modprobe br_netfilter

cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sysctl --system

# Install K8S
apt install -y apt-transport-https ca-certificates curl
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key --keyring /usr/share/keyrings/cloud.goog
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | gpg --batch --dearmor -o /etc/apt/keyrings
echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-

apt update -y
apt install -y jq kubelet="$KUBERNETES_VERSION" kubectl="$KUBERNETES_VERSION" kubeadm="$KUBERNETES_VERSION"

# config replace regex from https://superuser.com/questions/1485847/command-to-disable-password-login-via-ssh
```

init > $ master.sh

```bash
#!/bin/bash
#
set -euxo pipefail

# control plane init and image pull
kubeadm config images pull
kubeadm init --apiserver-advertise-address=$API_SERVER_IP --apiserver-cert-extra-sans=$API_SERVER_IP --pod-ne
echo "[-+-+-+-+-+-+-+-+!K8S Control Panel Up!-+-+-+-+-+-+-+-+]"

# copy token script and kubeconfig from master
kubeadm token create --print-join-command | tee /vagrant/node_join.sh
cp /etc/kubernetes/admin.conf /vagrant/admin.config
echo "[-+-+-+-+-+-+-+-+!K8S Credentials copied to host OS /vagrant folder!-+-+-+-+-+-+-+-+]"

# copy kube config to master home folder
mkdir -p /home/vagrant/.kube
cp /etc/kubernetes/admin.conf /home/vagrant/.kube/config
chown -R vagrant: /home/vagrant
echo "[-+-+-+-+-+-+-+-+!K8S Credentials exported to /home/vagrant/.kube/!-+-+-+-+-+-+-+-+]"
```

Command Prompt - vagrant up

```
0e478
    k8s-master: + echo '[-+-+-+-+-+-+-+-+!K8S Control Panel Up!-+-+-+-+-+-+-+-+]'
    k8s-master: [-+-+-+-+-+-+-+-+!K8S Control Panel Up!-+-+-+-+-+-+-+-+]
    k8s-master: + tee /vagrant/node_join.sh
    k8s-master: + kubeadm token create --print-join-command
    k8s-master: kubeadm join 192.168.13.37:6443 --token xel22w.gn1ccd2d8n9q5oxy --discovery-token-ca-cert-hash sha256:1
810ef29a5d70491114fa7633636cfb0862d0cf50abfa97a9cde253b40e478
    k8s-master: + cp /etc/kubernetes/admin.conf /vagrant/admin.config
    k8s-master: [-+-+-+-+-+-+-+-+!K8S Credentials copied to host OS /vagrant folder!-+-+-+-+-+-+-+-+]
    k8s-master: + echo '[-+-+-+-+-+-+-+-+!K8S Credentials copied to host OS /vagrant folder!-+-+-+-+-+-+-+-+]'
    k8s-master: + mkdir -p /home/vagrant/.kube
    k8s-master: + cp /etc/kubernetes/admin.conf /home/vagrant/.kube/config
    k8s-master: + chown -R vagrant: /home/vagrant
    k8s-master: [-+-+-+-+-+-+-+-+!K8S Credentials exported to /home/vagrant/.kube/!-+-+-+-+-+-+-+-+]
    k8s-master: + echo '[-+-+-+-+-+-+-+-+!K8S Credentials exported to /home/vagrant/.kube/!-+-+-+-+-+-+-+-+]'
==> k8s-worker-1: Importing base box 'bento/ubuntu-22.04'...
==> k8s-worker-1: Matching MAC address for NAT networking...
==> k8s-worker-1: Checking if box 'bento/ubuntu-22.04' version '202303.13.0' is up to date...
==> k8s-worker-1: Setting the name of the VM: enron.corp.k8s.local-k8s-worker-node-1
==> k8s-worker-1: Fixed port collision for 22 => 2222. Now on port 2200.
==> k8s-worker-1: Clearing any previously set network interfaces...
==> k8s-worker-1: Preparing network interfaces based on configuration...
    k8s-worker-1: Adapter 1: nat
    k8s-worker-1: Adapter 2: hostonly
```

```
    k8s-worker-1: Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
    k8s-worker-1:
    k8s-worker-1: [-+-+-+-+-+-+-+-+!Worker node joined successfully!-+-+-+-+-+-+-+-+]
    k8s-worker-1: + echo '[-+-+-+-+-+-+-+-+!Worker node joined successfully!-+-+-+-+-+-+-+-+]'
==> k8s-worker-2: Importing base box 'bento/ubuntu-22.04'...
==> k8s-worker-2: Matching MAC address for NAT networking...
==> k8s-worker-2: Checking if box 'bento/ubuntu-22.04' version '202303.13.0' is up to date...
==> k8s-worker-2: Setting the name of the VM: enron.corp.k8s.local-k8s-worker-node-2
==> k8s-worker-2: Fixed port collision for 22 => 2222. Now on port 2201.
==> k8s-worker-2: Clearing any previously set network interfaces...
```

```
$ kubectl get nodes
NAME                          STATUS   ROLES           AGE    VERSION
enron.corp.k8s.local          Ready    control-plane   4h7m   v1.27.1
enron.corp.k8s.local.1612b401 Ready    <none>          4h2m   v1.27.1
enron.corp.k8s.local.9f8b2815 Ready    <none>          3h57m  v1.27.1
$
```

```
$ nmap -p6443 192.168.13.37 –vv -n
Starting Nmap 7.80 ( https://nmap.org ) at 2023-07-07 02:11 BST
Nmap scan report for 192.168.13.37
Host is up (0.0025s latency).

PORT     STATE SERVICE
6443/tcp open  sun-sr-https

Nmap done: 1 IP address (1 host up) scanned in 1.20 seconds
```

```
Nmap scan report for 192.168.13.37
Host is up, received user-set (0.023s latency).
Scanned at 2023-07-08 09:58:19 BST for 30s
Not shown: 65529 closed ports
Reason: 65529 conn-refused
PORT      STATE SERVICE      REASON
22/tcp    open  ssh          syn-ack
2379/tcp  open  etcd-client  syn-ack
2380/tcp  open  etcd-server  syn-ack
6443/tcp  open  sun-sr-https syn-ack
10250/tcp open  unknown      syn-ack
10256/tcp open  unknown      syn-ack

Nmap scan report for 192.168.13.38
Host is up, received user-set (0.037s latency).
Scanned at 2023-07-08 09:58:19 BST for 30s
Not shown: 65532 closed ports
Reason: 65532 conn-refused
PORT      STATE SERVICE REASON
22/tcp    open  ssh     syn-ack
10250/tcp open  unknown syn-ack
10256/tcp open  unknown syn-ack

Nmap scan report for 192.168.13.39
Host is up, received user-set (0.034s latency).
Scanned at 2023-07-08 09:58:19 BST for 31s
Not shown: 65532 closed ports
Reason: 65532 conn-refused
PORT      STATE SERVICE REASON
22/tcp    open  ssh     syn-ack
10250/tcp open  unknown syn-ack
10256/tcp open  unknown syn-ack
```

Try it out yourself!

https://github.com/nishaanthguna22/vagrant-deployment-files/tree/master
https://t.ly/bVMv

References

- https://en.wikipedia.org/wiki/Kubernetes
- https://kubernetes.io/
- https://cloud.google.com/static/kubernetes-engine/images/gke-architecture.svg
- https://gist.github.com/nilesh93/c743205d34fedb5f48ae4d37d959ba4b
- https://sysdig.com/blog/kubernetes-admission-controllers/
- https://github.com/kubernetes/apimachinery/blob/master/pkg/apis/meta/v1/types.go
- https://github.com/kubernetes/api/blob/master/admission/v1/types.go
- https://i.blackhat.com/USA-22/Thursday/US-22-Avrahami-Kubernetes-Privilege-Escalation-Container-Escape-Cluster-Admin.pdf