KUBERNETES PLAYBOOK

Diagnostics

Find the different devices (nodes) in the cluster. This will help you identify the different computers that are running pods and services.

```
kubectl get nodes
```

This can be done on any node as long as it is communicating properly with the master node:
Here are the following problem status' a node can show when running this command:

NotReady: Ran into an issue and pods cannot run on it.
SchedulingDisabled: Node is unauthorized to run pods. use `kubectl uncordon <NODE-NAME>` to make it schedulable again and `kubectl cordon <NODE-NAME>` to unschedule it.
Unknown: Node is unreachable by the master node.

Finding Resources

__when listing these resources, it is important to understand if you don't specify a namespace, it will list available resources in the 'default' namespace. To specify a namespace, use the `-n <NAME-SPACE>`. Then the `-A` flag will append from all namespaces.

`find / -name *.yaml` - Use this to find all yaml files present in the host in order to figure out where your configuration files would be.
`kubectl api-resources -o wide` - lists all resources with extended output.
`kubectl cluster-info` - Displays important networking and internal information about cluster.
`kubectl get services` - This will list services running on the cluster, whether they're running correctly or not.
`kubectl get pods -o wide -A` - This will list all pods running in every namespace with more detail.
`kubectl get deployments` - This will list all current deployments
`kubectl get pod <POD-NAME> -o yaml` - This will specifically get a pod's .yaml or .json file if you specify it.
SOME PODS NEED TO USE PERSISTENT VOLUMES AND THEIR RESPECTIVE CLAIMS:
`kubectl get pv` - This will list persistent volumes

`kubectl get pvc` - This will list persistent volume claims that should go hand-in-hand with persistent volumes (common point of failure)
`kubectl get secret` - This will list the secrets generated in the cluster (IMPORTANT)
`kubectl get events` - List events going on in the cluster and filter by warning events by adding `--types=Warning` to the end of the command
`kubectl logs <RESOURCE-NAME>` - This will list the logs of any running resource you'd like (pods, deployments, etc.)

describe: use this kubectl argument to describe any resource you'd like (pods, deployments, nodes, etc.) Here's an example:

```
kubectl describe pod <POD-NAME> -A
```

Sidenote: you can also execute the `kubectl config view` command to show the kubeconfig settings. Along with `kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser --password=kubepassword` to create a new user for the cluster if needed.

Debuging for NotReady State

- Verify each node has exactly one kube-proxy pod and is in the running state with `kubectl get pods -n kube-system -o wide` if any pod is in some state other than running, use `kubectl describe pod <POD-NAME> -n kube-system` to get some information about what could be going wrong. You can also use `kubectl logs <POD-NAME> -n kubesystem` to get logs about the pods running.
- If a node does not have a kube-proxy pod then you can use `kubectl describe daemonset kube-proxy -n kube-system` to figure out what the issue could be since this daemon is responsible for providing the kube-proxy pods to each node
- use `kubectl describe node <NODE-NAME>` to get some information about the node itself.
    - Verify kubelet is running on the node. memorypressure, diskpressure, etc. status read unknown, then this could mean the kubelet process has run into some issues. So go into the node having this issue and `systemctl status kubelet` to try and figure it out. You can also use `journalctl -u kubelet` to check logs.
    - If status read NetworkUnavailable then check firewall and network rules to ensure these computers can still communicate with each other on the correct and needed ports.
- Resource issues:
    - **PIDPRESSURE**: If this value is set to 'true' then this means that there are too many processes running on the node.

- **MemoryPressure**: if this is true then this means node memory is low
- **DiskPressure**: If this is 'true' then this means the disk capacity is low

Maintenance

For either injects or remediation, we may need to run maintenance operations in order to ensure our environment is both accessible to orange team and easily recoverable from red team.

`kubectl create namespace <NAMESPACE-NAME>` - Creates a namespace.
`kubectl create secret generic <NAME-OF-NEW-SECRET> --from-file=/directoryto/file` - This will create a new secret based on an already existing file. You can also generate secrets from .yaml files, which will be talked about later on
`kubectl delete <RESOURCE-TYPE> <RESOURCE-NAME>` - This will delete whatever resource you specify. Remember, you might need to specify the namespace, as you do with all resource manipulation.
`kubectl exec -it <POD_NAME> -- bash` - This will run your pod as an interactive shell using bash. You can also use another shell, such as sh.
`kubectl run nginx --image=nginx -n <NAMESPACE-NAME>` - This will run a single instance of a nginx pod in the namespace you specify
`kubectl attach <POD-NAME> -c <CONTAINER-NAME>` - This will attach a pod to running container.
`kubectl port-forward <POD-NAME> 5000:6000` - This will listen on port 5000 on the localhost and forward to port 6000 on the pod you specify
`kubectl port-forward svc/<SVC-NAME> 5000` - local host listens on port 5000 and forwards to port 5000 on service
`kubectl port-forward svc/<SVC-NAME> 5000:<SVC-PORT>` - local port listens on port 5000 and forwards to specified target port
`kubectl exec <POD-NAME> -- <CMD>` - This will execute a command to the pod you specify
`kubectl top pod <POD_NAME>` - Displays metric information for the given pod. You can also specify its containers with the `--containers` flag as well as sorting this metric information with `--sort-by=cpu`.
`kubectl cp /directoryto/file_dir <POD-NAME>:/directoryto/file_dir` - This will copy a local directory to the pods remote directory.
`kubectl cp /directoryto/file <POD-NAME>:/directoryto/file` - This will copy a local file to a pod's remote directory.
`kubectl cp <NAMESPACE-NAME>/<POD-NAME>:/directoryto/file /directoryto/file` - This copies pod's remote file to the localhost.
Note that 'kubectl cp' will fail if the 'tar' binary is not present in the container's image.
`kubectl apply -f <FILE-NAME>` - This will apply a configuration to a resource by

filename. If the resource does not exist yet, it will be created. This can be done with any resource such as deployment, secrets, services, etc.

You may need to create a local persistent volume manually. If so, here is how to do it: `kubectl create -f <LOCAL-VOLUME-NAME>.yaml` Here is an example of what that yaml file might look like:

```
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-volume-1
  labels:
    type: local
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/data/lv-1
  persistentVolumeReclaimPolicy: Recycle
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-volume-2
  labels:
    type: local
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/data/lv-2
  persistentVolumeReclaimPolicy: Recycle
```

You can deploy the Kubernetes dashboard if you'd like. While this makes administration easier, it also widens attack surface.

```
GITHUB_URL=https://github.com/kubernetes/dashboard/releases
VERSION_KUBE_DASHBOARD=$(curl -w '%{url_effective}' -I -L -s -S
${GITHUB_URL}/latest -o /dev/null | sed -e 's|.*/||')
sudo k3s kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/${VERSION_KUBE_DASHBO
ARD}/aio/deploy/recommended.yaml
```

Updating

`kubectl set image <DEPLOYMENT>/<FRONTEND> www=image:v2` - This will update the web containers, which would be the frontend of the deployment, and thus updating the image.
`kubectl rollout history <DEPLOYMENT>/<FRONTEND> `- This will check the history of the deployments and all updates involved with them.
`kubectl rollout undo <DEPLOYMENT>/<FRONTEND>` - This will rollback to previous deployment
`kubectl rollout undo <DEPLOYMENT>/<FRONTEND> --to-revision=2` - This will rollback to a specific revision
`kubectl rollout restart <DEPLOYMENT>/<FRONTEND>` - This will initiate a rolling restart whatever is the frontend of specified deployment

For k3s

check the version of k3s with `k3s --version`.
For manual updates, you need to use `k3s-killall.sh` to stop all production deployments and then `curl -sfL https://get.k3s.io | sh -` This will automatically update the cluster.
If the script doesn't work, you're going to have to update from the binary. (look it up. I'm so tired at the time of writing this)
`curl -sfL https://get.k3s.io | sh -s - server --secrets-encryption` use this so that you can start to the cluster with secret encryption which will be very important.

Security Best Practices

There are going to only be a couple of things we can do in a CCDC competition environment in order to secure our cluster in the fastest way possible. Those things are going to include:

- RBAC implementation
- Secret at rest encryption
- etcd encryption and backup/snapshots

- token generation/ management
- audit logging
- CIS hardening checklist

Encryption

k3s comes with a utility called secrets-encryption that enables automatic control over secret encryption, encryption management, rotation, and more.

Before we can do anything with encryption, remember this encryption service is not enabled on an existing server without restarting it with the correct flag.
`curl -sfL https://get.k3s.io | sh -s - server --secrets-encryption`
Here is an example of what this configuration file might look like:

```
{
  "kind": "EncryptionConfiguration",
  "apiVersion": "apiserver.config.k8s.io/v1",
  "resources": [
    {
      "resources": [
        "secrets"
      ],
      "providers": [
        {
          "aescbc": {
            "keys": [
              {
                "name": "aescbckey",
                "secret": "xxxxxxxxxxxxxxxxxxx"
              }
            ]
          }
        },
        {
          "identity": {}
        }
      ]
    }
  ]
}
```

1. prepare this utility with `k3s secrets-encrypt prepare`
2. kill and restart the server

```
systemctl restart k3s   #if using systemd
rc-servicek3s restart   #if using openrc
```

3. rotate `k3s secrets-encrpyt rotate`
4. kill and restart k3s with the arguments mentioned above
5. reencrypt with `k3s secrets-encrypt reencrypt`

__To re-enable secret encrypt on a single node cluster:

1. enable with `k3s secrets-encrypt enable`
2. restart k3s server
3. re-encrypt with these flags `k3s secrets-encrypt reencrypt --force --skip`

__You can see the encryption status of the secrets with `k3s secrets-encrypt status`

etcd snapshots

run `k3s etcd-snapshot` to save a snapshot of the etcd from the get go.
To restore from a snapshot, you would enter this command:

```
k3s server \
  --cluster-reset \
  --cluster-reset-restore-path=<PATH-TO-SNAPSHOT>
```

Tokens

Remember to copy server token and place in a safe place so that red team doesn't steal it. The token can be located at `/var/lib/rancher/k3s/server/agent-token`
you can also generate a new token by using `k3s token generate` and `k3s token delete` to delete tokens. Provide either the full token or the token ID.

Audit Logging

k3s doesn't enable audit logging by default, so this needs to be configured manually.

Create the directory that will be storing logs:
`mkdir -p -m 700 /var/lib/rancher/k3s/server/logs`
Then create an audit policy named 'audit.yaml' and store that configuration file in

/var/lib/rancher/k3s/server/
This audit policy should be this:

```
apiVersion: audit.k8s.io/v1 # This is required.
kind: Policy
# Don't generate audit events for all requests in RequestReceived stage.
omitStages:
  - "RequestReceived"
rules:
  # Log pod changes at RequestResponse level
  - level: RequestResponse
    resources:
    - group: ""
      # Resource "pods" doesn't match requests to any subresource of pods,
      # which is consistent with the RBAC policy.
      resources: ["pods"]
  # Log "pods/log", "pods/status" at Metadata level
  - level: Metadata
    resources:
    - group: ""
      resources: ["pods/log", "pods/status"]

  # Don't log requests to a configmap called "controller-leader"
  - level: None
    resources:
    - group: ""
      resources: ["configmaps"]
      resourceNames: ["controller-leader"]

  # Don't log watch requests by the "system:kube-proxy" on endpoints or
services
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
    - group: "" # core API group
      resources: ["endpoints", "services"]

  # Don't log authenticated requests to certain non-resource URL paths.
  - level: None
```

```
    userGroups: ["system:authenticated"]
    nonResourceURLs:
    - "/api*" # Wildcard matching.
    - "/version"

  # Log the request body of configmap changes in kube-system.
  - level: Request
    resources:
    - group: "" # core API group
      resources: ["configmaps"]
    # This rule only applies to resources in the "kube-system" namespace.
    # The empty string "" can be used to select non-namespaced resources.
    namespaces: ["kube-system"]


  # Log configmap and secret changes in all other namespaces at the Metadata
level.
  - level: Metadata
    resources:
    - group: "" # core API group
      resources: ["secrets", "configmaps"]



  # Log all other resources in core and extensions at the Request level.
  - level: Request
    resources:
    - group: "" # core API group
    - group: "extensions" # Version of group should NOT be included.

  # A catch-all rule to log all other requests at the Metadata level.
  - level: Metadata
    # Long-running requests like watches that fall under this rule will not
    # generate an audit event in RequestReceived.
    omitStages:
      - "RequestReceived"
```

or it can be something as simple as below, although this will collect a lot more data than the log previously mentioned:

```
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata
```

Since the server is already started and install, you can add these to the k3s systemd service configuration file usually located in /etc/systemd/system/k3s.service

```
ExecStart=/usr/local/bin/k3s \
    server \
    '--kube-apiserver-arg=audit-log-
path=/var/lib/rancher/k3s/server/logs/audit.log' \
    '--kube-apiserver-arg=audit-policy-
file=/var/lib/rancher/k3s/server/audit.yaml' \
```

Then restart the service with these two commands:

```
sudo systemctl daemon-reload
sudo systemctl restart k3s.service
```

CIS Self Assessment Checklist

- **Ensure that the Container Network Interface file permissions are set to 644**
  - chmod 644 /Directory/to/netinterfacefile
- **Ensure that the Container Network Interface file ownership is set to root:root**
  - chown root:root /path/to/netinterfacefile
- __Ensure that the admin.conf file ownership is set to root:root
  - chown root:root /var/lib/rancher/k3s/server/cred/admin.kubeconfig
- __Ensure that the scheduler.conf permissions are set to 644
  - chmod 644 /var/lib/rancher/k3s/server/cred/scheduler.kubeconfig
- __Ensure scheduler.conf ownership is set to root:root
  - chown root:root /var/lib/rancher/k3s/server/cred/scheduler.kubeconfig
- __Ensure that the controller-manager.conf file permissions are set to 644 or more restrictive
  - chmod 644 /var/lib/rancher/k3s/server/cred/controller.kubeconfig
- __Ensure that the controller-manager.conf file ownership is set to root:root
  - chown root:root /var/lib/rancher/k3s/server/cred/controller.kubeconfig
- __Ensure that the Kubernetes PKI directory and file ownership is set to root:root
  - chown -R root:root /etc/kubernetes/pki/

- __Ensure that the Kubernetes PKI certificate file permissions are set to 644 or more restrictive
  - chmod -R 644 /etc/kubernetes/pki/*.crt
- __Ensure that the Kubernetes PKI key file permissions are set to 600
  - chmod -R 600 /etc/kubernetes/pki/*.key
- __Ensure that the --anonymous-auth argument is set to false
  - Ensure this is true in the /etc/kubernetes/manifests/kube-apiserver.yaml configuration file
- __Ensure that the --token-auth-file parameter is not set
  - Ensure this is true in the /etc/kubernetes/manifests/kube-apiserver.yaml configuration file
- __Ensure that the --DenyServiceExternalIPs is not set
  - Ensure this is true in the /etc/kubernetes/manifests/kube-apiserver.yaml configuration file
- __Ensure that the --authorization-mode argument includes RBAC
  - Ensure that --authorization-mode=Node,RBAC is configured in the /etc/kubernetes/manifests/kube-apiserver.yaml configuration file
- __Ensure that the admission control plugin AlwaysAdmit is not set
  - remove the --enable-adminssion-plugins /etc/kubernetes/manifests/kube-apiserver.yaml
- __Ensure that the admission control plugin NodeRestriction is set
  - set a value to --enable-adminssion-plugin=...,NodeRestriction,... /etc/kubernetes/manifests/kube-apiserver.yaml
- __Ensure that the --audit-log-path argument is set
  - Edit the /etc/kubernetes/manifests/kube-apiserver.yaml configuration file so that --audit-log-path parameter writes all the audit logs to where it should be EXAMPLE: --audit-log-path=/var/log/apiserver/audit.log
- __Ensure that the --service-account-lookup argument is set to true
  - Edit the /etc/kubernetes/manifests/kube-apiserver.yaml so that the --service-account-lookup=true
- __Ensure that the API Server only makes use of Strong Cryptographic Ciphers
  - Ensure that /etc/kubernetes/manifests/kube-apiserver.yaml configuration file has this parameter --tls-cipher-suites=TLS_AES_128_GCM_SHA256,TLS_AES_256_GCM_SHA384,TLS_CHACHA20_POLY1305_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_

    256_GCM_SHA384,
    TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256,
    TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,
    TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256,TLS_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,
    TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_GCM_SHA384
- __Ensure that the --terminated-pod-gc-threshold argument is set as appropriate
    - Edit the Controller manager configuration file /etc/kubernetes/manifests/kube-controller-manager.yaml on the control plane node so that it has this parameter --terminated-pod-gc-threshold=10
- __If proxy kubeconfig file exists ensure permissions are set to 644 or more restrictive
    - chmod 644 /var/lib/rancher/k3s/agent/kubeproxy.kubeconfig
- __If proxy kubeconfig file exists ensure ownership is set to root:root
    - chown root:root /var/lib/rancher/k3s/agent/kubeproxy.kubeconfig

Worker Node Configuration Files

- __Ensure that the kubelet service file permissions are set to 644 or more restrictive
    - chmod 644 /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
- __Ensure that the kubelet service file ownership is set to root:root
    - chown root:root /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
- __If proxy kubeconfig file exists ensure permissions are set to 644 or more restrictive
    - chmod 644 /var/lib/rancher/k3s/agent/kubeproxy.kubeconfig
- __If proxy kubeconfig file exists ensure ownership is set to root:root
    - chown root:root /var/lib/rancher/k3s/agent/kubeproxy.kubeconfig
- __Ensure that the --kubeconfig kubelet.conf file permissions are set to 644 or more restrictive
    - chmod 644 /var/lib/rancher/k3s/server/cred/admin.kubeconfig
- __Ensure that the --kubeconfig kubelet.conf file ownership is set to root:root
    - chown root:root /var/lib/rancher/k3s/server/cred/admin.kubeconfig

Set Kernel Parameters

CIS api-server compliant configurations:

Supplemental Help / Information

Kubectl is your best friend. So whenever you kind yourself stuck, use the -h flag

when using the `--all-namespaces` flag use the shorthand instead which is `-A`