P1 Full Defense Command Sheet

```
# ── Pre-flight: host machine (BEFORE vagrant up) ──────────────────────
# Confirm libvirt network exists
virsh net-list --all
# Must show:
# iot56   active   yes

# If inactive:
virsh net-start iot56

# Confirm you are in repo root
ls p1/Vagrantfile && echo "Repo root OK"

# ── Start up ──────────────────────────────────────────────────────────
cd p1/
vagrant up
# → Provisions alrahmouS first (server), then alrahmouSW (worker)
# → Server script detects interface, installs K3s in server mode
# → Worker waits for server API ready, then joins cluster

# Verify both VMs are running
vagrant status
# → alrahmouS    running (libvirt)
# → alrahmouSW   running (libvirt)

# ── Connect to Server ─────────────────────────────────────────────────
vagrant ssh alrahmouS

# ── Evalsheet checks on SERVER (run inside alrahmouS) ─────────────────

# Hostname
hostname
# → alrahmouS

# IP verification (evalsheet: use ip a, NOT ifconfig)
ip a | grep 192.168.56.110
# or:
ip a show <interface_name>
# → inet 192.168.56.110

# K3s service — evalsheet checks BOTH active AND enabled
sudo systemctl is-active k3s && sudo systemctl is-enabled k3s
# → active
# → enabled

# Cluster nodes — the critical command
kubectl get nodes -o wide
# → NAME          STATUS   ROLES                AGE   IP
# → alrahmouS     Ready    control-plane,master   Xm    192.168.56.110
# → alrahmouSW    Ready    <none>               Xm    192.168.56.111

# Both nodes STATUS: Ready
# Both nodes correct INTERNAL-IP
```

```
# Evaluator will ask you to explain this output — see table below

# ── Connect to Worker (separate terminal) ──────────────────────────────
vagrant ssh alrahmouSW

# ── Evalsheet checks on WORKER (run inside alrahmouSW) ─────────────────

# Hostname
hostname
# → alrahmouSW

# IP verification
ip a | grep 192.168.56.111

# K3s agent service — evalsheet checks BOTH active AND enabled
sudo systemctl is-active k3s-agent && sudo systemctl is-enabled k3s-agent
# → active
# → enabled

# ── Back on SERVER — additional verification ───────────────────────────

# Confirm token was used (cluster joined correctly)
sudo cat /var/lib/rancher/k3s/server/node-token
# → token exists (the same IOT42ClusterToken used in Vagrantfile)

# Confirm K3s config was applied
sudo cat /etc/rancher/k3s/config.yaml
# → write-kubeconfig-mode: "0644"
# → token: IOT42ClusterToken
# → node-ip: 192.168.56.110
# → advertise-address: 192.168.56.110

# Confirm kubeconfig is accessible without sudo
kubectl get nodes
# → works without sudo

# Full cluster info
kubectl cluster-info
# → Kubernetes control plane running at https://192.168.56.110:6443
```

Key Explanation Table (Evaluator Will Ask)

Question	Answer
"*Explain `kubectl get nodes -o wide` output*"	"Two nodes: `alrahmouS` is the control-plane running the API server, scheduler, controller manager. `alrahmouSW` is the worker agent — it receives pod scheduling from the server. Both are `Ready` meaning they are healthy and registered. `-o wide` shows the internal IP confirming they communicate over the `192.168.56.x` private network"
"*What is K3s?*"	"A lightweight, CNCF-certified Kubernetes distribution by Rancher — ships as a single binary, replaces etcd with SQLite, ideal for VMs and edge. Full K8s API compatibility"
"*What is the difference between server and agent mode?*"	"Server mode runs the control plane — API server, scheduler, controller manager. Agent mode runs only the kubelet and kube-proxy — it receives workloads from the server and runs pods"
"*What is Vagrant?*"	"A CLI tool that automates VM creation via a declarative Vagrantfile — describes the OS, network, resources, and provisioning scripts. One `vagrant up` creates both VMs reproducibly"
"*How does the worker know the server token?*"	"The token `IOT42ClusterToken` is pre-shared in the Vagrantfile via an environment variable — passed to both scripts at provision time. No manual copy step needed"
"*Why `config.ssh.insert_key = false`?*"	"Keeps the default insecure Vagrant key for both VMs — ensures `vagrant ssh` works without a password on both machines, as the subject requires"

Cleanup / Re-provision

```
# If something broke during provisioning:
vagrant reload --provision

# Full reset:
vagrant destroy -f
vagrant up
```

```
# SSH into specific VM:
vagrant ssh alrahmouS
vagrant ssh alrahmouSW

# Stop VMs without destroying (save resources between defenses):
vagrant halt
vagrant up   # resume
```