LAB ACTION PLAN FOR WEEK 10

Working with minikube and Nagios

1. Hands-on practice of creating, running and scaling pods in minikube.
2. Running Nginx server on specified port number by explaining the Nginx monitoring tool
3. Running Nagios server and Understanding the Monitoring tool using Docker.
4. AWS-free Trier account Creation steps
5. Upload the screenshots for the tasks

Kubernetes:

Kubernetes is a tool that automates how we run and manage applications inside the container.

Dockers will only run containers, if in any case the container fails/stopped/killed, the docker will not help us, here is where Kubernetes plays an important role, Kubernetes cluster will be responsible in creating a new container and managing various containers.

POD: In Kubernetes, a Pod is the smallest deployable unit that you can create and manage.

Minikube:

Minikube creates a VM on your local machine and deploys a simple Kubernetes cluster with one node. It's a lightweight implementation. Minikube is a version of Kubernetes.

Nagios:

Nagios is an **open-source IT infrastructure monitoring tool**. It monitors

- Servers
- Network devices
- Applications and services

It **alerts admnistrators** when issues occur and notifies when they are resolved.

Step 1: Install Prerequisites

Before installing Minikube, ensure the following are installed:

1. **Virtualization Support**:
   - Verify virtualization is enabled:

2. **Hypervisor**:
    - Minikube supports multiple hypervisors (e.g., **Hyper-V**, **VirtualBox**, or **Docker** as a driver).
    - Install one of the following:
        - **Hyper-V** (pre-installed on Windows 10/11 Pro or Enterprise).
        - **Docker Desktop** (if you want to use Docker as the driver).

Step 2: Download Minikube

1. Open a PowerShell or Command Prompt with administrator privileges.
2. Download the latest Minikube executable using this command:
3. `curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe`
4. Install Minikube by running the installer:
5. `.\minikube-installer.exe`

Step 3: Add Minikube to PATH

If Minikube is not automatically added to your PATH during installation:

1. Open **System Properties → Environment Variables**.
2. Add the directory where Minikube is installed (e.g., `C:\Program Files\Minikube`) to your PATH variable.

Step 4: Start Minikube

1. Open a terminal (PowerShell or CMD). Do the following commands
2. Start Minikube with a specified driver (e.g., Hyper-V, Docker, or VirtualBox). For example:
    - **Hyper-V**:
    - `minikube start --driver=hyperv`
    - **Docker**:
    - `minikube start --driver=docker`
3. Verify Minikube is running:
4. `minikube status`

Step 5: Interact with Minikube

kubectl is a command-line tool used in Kubernetes to interact with and manage Kubernetes clusters.

Once Minikube is running:

1. Use `kubectl` to interact with the cluster.
    - Install `kubectl` if not already installed:
    - `minikube kubectl -- get pods -A`
    - Or download it separately from the official Kubernetes site.
2. Open the Minikube dashboard (optional):
3. `minikube dashboard`

Optional: Check Your Installation

Run the following to verify the installation:

Minikube version

```
kubectl version --client
```

Troubleshooting

1. **If Minikube fails to start**:
    - Ensure your hypervisor (Hyper-V/Docker/VirtualBox) is installed and running.
    - Check the Minikube logs:
    - `minikube logs`
2. **Updating Minikube**:
3. `minikube update-check`
4. `minikube update`

Minikube Automation Steps

Step 1: Start Minikube Cluster

- Open your terminal and run the command:

```
minikube start
```

Step 2: Create and Manage Deployment

1. **Create an application in Kubernetes:**

    - Command:

```
kubectl create deployment mynginx --image=nginx

if already created then

kubectl set image deployment/myngnix nginx=nginx:latest
```

- Verify the deployment using: Kubernetes responds by showing you a list that includes the names of your deployment groups

```
kubectl get deployments
```

  - Ensure `mynginx` appears in the output.

Check the following commands:

- kubectl get pods
- kubectl describe pods

2. **Expose Deployment as a Service**:
   - Command:

```
kubectl expose deployment mynginx --type=NodePort --port=80 --target-port=80
```

Step 3: Scale the Deployment

Command:Scales the Nginx deployment to 4 replicas (pods).

```
kubectl scale deployment mynginx --replicas=4

kubectl get service myngnix
```

Step 4: Access the Nginx App

1. **Using Port Forwarding**:
   - Command:

```
kubectl port-forward svc/mynginx 8081:80
```

- Access the app via http://localhost:8081.

If Error, use this option, **Using Minikube Tunnel**:

- Start the tunnel:

```
minikube tunnel
```

- Retrieve the service URL:

```
minikube service mynginx --url
```

- Open the provided URL in your browser.
- Open the kubernets dashboard

- Open the minikube dashboard

  Minikube dashboard

Step 5: Stop and Clean Up

1. **Stop Nginx Deployment**:
   - Commands:

```
kubectl delete deployment mynginx
kubectl delete service mynginx
```

2. **Stop Minikube (Optional)**:
   - Command:

```
minikube stop
```

3. **Delete Minikube Cluster (Optional)**:
   - Command:`minikube delete`

Nagios Automation Steps

Step 1: Pull the Nagios Docker Image

- Open a terminal and run:

```
docker pull jasonrivers/nagios:latest
```

Step 2: Run Nagios

- Command:

```
docker run --name nagiosdemo -p 8888:80 jasonrivers/nagios:latest
```

Step 3: Access Nagios Dashboard

- Open your browser and navigate to:

```
http://localhost:8888
```

    - **Login Credentials**:
        - Username: `nagiosadmin`
        - Password: `nagios`
    - Once logged in, explore:
        - Hosts: View systems being monitored.
        - Services: Check tasks being monitored (e.g., CPU usage).
        - Alerts: Access recent notifications.

Step 4: Monitoring Host Details

1. **Navigate to the Host Information Page**:
    - Select a host from the **Hosts** menu.
2. **Key Details**:
    - Host Status: Indicates if the system is UP or DOWN.
    - Metrics: View CPU usage, memory status, and network activity.

- Actions: Reschedule checks, disable notifications, or schedule downtime.

Step 5: Stop and Remove Nagios

1. **Stop the Container**:
   - Command:

```
docker stop nagiosdemo
```

2. **Delete the Container**:
   - Command:

```
docker rm nagiosdemo
```

3. **Remove the Image** (Optional):
   - List images:

```
docker images
```

   - Delete the Nagios image:

```
docker rmijasonrivers/nagios:latest
```

4. Observe the docker containers in DockerHub, we can see the latest Nagios Installed running on port:8888

Steps for AWS-free trier account creation

- open https://aws.amazon.com
- click on create AWS account
- Provide email id and name details for AWS account creation then a screen will appear as below
- Enter verification code received in your mail
- Provide your contact details and agree Terms and conditions, then click on create

- Provide billing details, click on verify and continue
- Amazon chagres 2 rs and it will be credit back into your account once verification is over
- Once payment success a screen appear as below, provide your working contact number after selecting country, and click on send SMS after entering the details
- Provide verification code received in your mobile
- Now select basic plan and click on complete sign up
- Now a screen appear as follows then click on AWS management console
- You will navigating to the aws page ..where you need to Select role as Academic/Researcher and interest as DevOps and click on submit
- Sign into AWS console as Root user
- A console screen appears aws will appear

SBQ's

1. Your Pod keeps restarting repeatedly. What will you do?
2. A Kubernetes pod is stuck in a "Pending" state. What could be the possible reasons, and how would you troubleshoot it?
3. How would you debug a failed deployment in Kubernetes?
4. You have a Kubernetes Deployment with multiple replicas, and some pods are failing health checks. How would you identify the root cause and fix it?
5. How do you roll back a faulty deployment?
6. How do you debug a running Pod?
7. You need to expose a local service externally.How to do?
8. How to Start and stop Nagios
9. You installed Nagios but the web interface shows *"Unable to connect to Nagios process"*.how to resolve this?
10. You added a new host in Nagios, but it's not appearing on the web interface.how to check?
11. How can you check whether Nagios is running properly?
12. How do you view Nagios logs in real-time
13. What are the advantages of using Nagios?

SBQ

Nice set — I'll answer each one with practical steps and commands you can run immediately.

---

1 — Your Pod keeps restarting repeatedly. What will you do?

Quick checklist + commands:

1. See status & restart reason

```bash
kubectl get pod <pod-name> -o wide
kubectl describe pod <pod-name>
kubectl get events --sort-by=.metadata.creationTimestamp
```

2. Inspect logs (current and previous)

```bash
kubectl logs <pod-name>          # for single-container pod
kubectl logs <pod-name> -c <ctr>  # multi-container
kubectl logs <pod-name> --previous  # logs of last terminated container
```

3. If CrashLoopBackOff / OOMKilled:

* CrashLoopBackOff → check startup failure (entrypoint, env, args, missing config)
* OOMKilled → check resource limits/requests in the Pod/Deployment

```bash
```

kubectl describe pod <pod> | grep -i OOMKilled -A3

```

4. Check liveness/readiness probes — overly aggressive probes cause restarts. Tune `initialDelaySeconds`, `periodSeconds`, `timeoutSeconds`.

5. Check image pull issues or health of dependent services (DB, API).

6. Reproduce locally or exec into container (if alive) to inspect filesystem, configs:

```bash

kubectl exec -it <pod> -- /bin/sh

```

7. If needed, scale down and run a single replica with debug image or add extra logging.

 # 2 — Pod stuck in **Pending. Possible reasons & troubleshooting**

Common causes:

*** No schedulable node (insufficient CPU/memory)**

*** Unsatisfied **PersistentVolumeClaim** (PVC not bound)**

*** Image pull secret / image not found**

*** Node selectors, taints/tolerations mismatch**

*** Namespace resource quota reached**

*** Pod affinity/anti-affinity rules unsatisfiable**

Troubleshooting commands & checks:

```bash

kubectl describe pod <pod> # events will show scheduling reason (e.g., Unschedulable)

kubectl get nodes # node status

kubectl describe node <node>

kubectl get pvc # check PVC status

kubectl describe pvc <pvc-name>

kubectl get quota -n <ns> # resource quotas

kubectl get rs,deploy --all-namespaces

kubectl get pods -o wide

kubectl get events --sort-by=.metadata.creationTimestamp

```

Look for messages like `0/5 nodes are available: insufficient memory`, `pod has unbound immediate PersistentVolumeClaims`, `ImagePullBackOff`, or `node(s) had taints that the pod didn't tolerate`.

Fixes: add resources, adjust requests/limits, provision PV, correct image path/secret, change nodeSelector/taints or add tolerations, relax affinity.

 # 3 — How would you debug a failed Deployment?

Steps & commands:

1. Check rollout status:

```bash

kubectl rollout status deployment/<name>

```

2. Inspect Deployment & ReplicaSets:

```bash

kubectl describe deployment <name>

kubectl get rs -l app=<label>

kubectl describe rs <replicaset-name>

```

3. List pods and their status:

```bash

kubectl get pods -l app=<label> -o wide

kubectl describe pod <pod-name>

```

4. Check pod logs (and previous logs if crash):

```bash

kubectl logs <pod-name> --previous

```

5. Check events and cluster-level issues:

```bash

kubectl get events --sort-by=.lastTimestamp

kubectl get nodes

```

6. If deployment uses ConfigMap/Secret, ensure they are mounted and correct.

7. If rollout introduced a bad image: rollback or `kubectl set image` to a working image, or:

```bash

kubectl rollout undo deployment/<name>

kubectl rollout history deployment/<name>

kubectl rollout status deployment/<name>

```

8. For deeper issues: port-forward to pod and test, or exec into the container.

 # 4 — Deployment with multiple replicas; some pods failing health checks. How to identify root cause & fix?

Steps:

1. Identify failing pods:

```bash

kubectl get pods -l app=<label> -o wide

kubectl describe pod <failing-pod>

```

Look at `Events` and probe failures in the describe output.

2. Check logs for failing container:

```bash

kubectl logs <pod> -c <container>

kubectl logs <pod> -c <container> --previous

```

3. Examine readiness/liveness probe configuration in Deployment spec:

*** Are probe endpoints correct? (path/port)**

*** Are timeouts and initial delay long enough?**

*** Is the probe using the correct protocol (HTTP/TCP/exec)?**

4. Test probe endpoint from within cluster:

```bash

 # from a debug pod

kubectl run -it --rm debug --image=busybox --restart=Never -- /bin/sh

wget -qO- http://<pod-ip>:<port>/health

```

Or `kubectl port-forward` and curl locally.

5. Check resource constraints: pods may be throttled or slow due to CPU/memory limits. Increase requests/limits if needed.

6. Check for intermittent dependency failures (DB, external API) causing probes to fail.

7. Fixes:

*** Correct application bug causing failing health endpoint.**

*** Adjust probe settings (increase `initialDelaySeconds`, `timeoutSeconds`, `failureThreshold`).**

*** Increase resource requests/limits or optimize app.**

*** Ensure network connectivity to dependencies.**

 # 5 — How do you roll back a faulty deployment?

Simple commands:

1. Check rollout history:

```bash

kubectl rollout history deployment/<name>

```

2. Roll back to previous revision:

```bash

kubectl rollout undo deployment/<name>

```

3. Roll back to a specific revision:

```bash

kubectl rollout undo deployment/<name> --to-revision=<rev-number>

```

**4. Verify:**

```bash
kubectl rollout status deployment/<name>
kubectl get pods -l app=<label>
```

If the deployment strategy is `RollingUpdate`, rollback will gradually replace pods. If you used `kubectl apply` with a bad image tag, change the image to a stable tag with `kubectl set image` and monitor rollout.

 # 6 — How do you debug a running Pod?

Useful commands with purpose:

* Overview:

```bash
kubectl get pod <pod> -o wide
kubectl describe pod <pod>
```

* Live logs:

```bash
kubectl logs -f <pod> [-c <container>]
```
```

```

* Exec into container (interactive shell):

```bash
kubectl exec -it <pod> -c <container> -- /bin/sh    # or /bin/bash
```

* Run one-off debug pod in same namespace for network tests:

```bash
kubectl run -it --rm debug --image=busybox --restart=Never -- /bin/sh
```

* Attach to a process (if necessary):

```bash
kubectl attach -it <pod> -c <container>
```

* Check resource usage (metrics server / kubelet):

```bash
kubectl top pod <pod>       # requires metrics-server
kubectl top node
```

* Inspect mounted volumes, env vars, certificates, config maps by `exec` and reading files.
* If container crashed recently, use `kubectl logs --previous`.

```

 # 7 — You need to expose a local service externally. How to do?

Options depending on environment:

1. **kubectl port-forward (local dev, temporary):**

```bash
kubectl port-forward svc/<service-name> 8080:<service-port>
# then access http://localhost:8080
```

2. **Service Type=NodePort (exposes on node IP + port 30000–32767):**

```yaml
apiVersion: v1
kind: Service
metadata: {name: mysvc}
spec:
  type: NodePort
  selector: {app: myapp}
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30080
```

Access via `http://<node-ip>:30080`.

3. **Service Type=LoadBalancer (cloud providers expose external LB):**

```yaml

spec:

type: LoadBalancer

```

Provider provisions an external IP.

4. **Ingress (recommended for HTTP/S with host-based routing and TLS):**

*** Deploy an Ingress Controller (nginx, contour, traefik).**

*** Create an Ingress resource mapping host/path to services.**

*** Use TLS via cert-manager or provider.**

5. **MetalLB (on bare-metal clusters) to get LoadBalancer functionality.**

Choose based on your cluster environment. For short tests, use port-forward; for production HTTP, use Ingress + TLS.

 # 8 — How to Start and Stop Nagios

(On most Linux systems)

Using `systemd`:

```bash
sudo systemctl start nagios
sudo systemctl stop nagios
sudo systemctl restart nagios
sudo systemctl status nagios
```

Older `init.d`:

```bash
sudo service nagios start
sudo service nagios stop
```

Note: also ensure your web server is running (`apache2` or `httpd`), because the Nagios UI uses it:

```bash
sudo systemctl status apache2   # Debian/Ubuntu
sudo systemctl status httpd     # CentOS/RHEL
```

 # 9 — Nagios web shows “Unable to connect to Nagios process”. How to resolve?

Troubleshoot steps:

1. Confirm Nagios process is running:

```bash
ps aux | grep nagios
sudo systemctl status nagios
```

2. Check nagios error logs:

```bash
sudo tail -n 200 /var/log/nagios/nagios.log
# or /usr/local/nagios/var/nagios.log depending on install
sudo journalctl -u nagios -n 200
```

3. Verify external command pipe / permissions (web UI uses this to submit commands):

*** Check path: `/var/lib/nagios3/rw/nagios.cmd` or `/var/spool/nagios/cmd` depending on distro.**

*** Ensure apache user (www-data/httpd) can write to the external command file:**

```bash
ls -l /var/spool/nagios /var/spool/nagios/cmd
sudo chown nagios:www-data /var/spool/nagios/cmd
```

sudo chmod 660 /var/spool/nagios/cmd

```

4. Validate Nagios config:

```bash

sudo nagios -v /etc/nagios/nagios.cfg # path may vary

```

Fix reported errors and restart.

5. Ensure webserver CGI config is correct (Alias for cgi-bin, ScriptAlias), and Apache has the CGI enabled and the right permissions.

6. Check SELinux or AppArmor blocking access. Temporarily disable SELinux (or set permissive) to confirm.

7. After fixes, restart Nagios + webserver:

```bash

sudo systemctl restart nagios

sudo systemctl restart apache2 # or httpd

```

 # 10 — You added a new host in Nagios, but it’s not appearing on the web interface. How to check?

Checklist & commands:

1. Verify Host configuration file is in an included directory and syntax is correct:

```bash

 # example config validation

sudo nagios -v /etc/nagios/nagios.cfg

```

Fix any errors and warnings shown.

2. Ensure you reloaded/restarted Nagios after changes:

```bash

sudo systemctl reload nagios # or restart

```

3. Check the status.dat / objects.cache update:

*** `status.dat` (often `/var/cache/nagios/status.dat` or `/usr/local/nagios/var/status.dat`) contains current status; check timestamps.**

4. Permissions: webserver user must be able to read status and command files.

5. Check for duplicates or mis-typed hostnames/hostgroups that prevent display.

6. Look at Nagios logs for errors about reading the new config:

```bash
sudo tail -n 200 /var/log/nagios/nagios.log
```

7. Confirm the host is not set with `register 0` or `active_checks_enabled` disabled, which may affect visibility.

8. If using Director/CGI config GUI, ensure it pushed/persisted the config correctly.

 # 11 — How can you check whether Nagios is running properly?

Basic checks:

1. Systemd:

```bash
sudo systemctl status nagios
```

2. Process:

```bash
ps aux | grep nagios
```

3. Nagios internal validation:

```bash
sudo nagios -v /etc/nagios/nagios.cfg
```

4. Check log file for no recent critical errors:

```bash
sudo tail -n 100 /var/log/nagios/nagios.log
```

5. Check web UI — overview & host/service statuses.

6. Check external command pipe & `status.dat` timestamp to ensure updates are happening.

7. Confirm webserver is serving the CGI pages (permissions, apache/nginx config).

 # 12 — How do you view Nagios logs in real-time?

Tail the log or use journalctl:

*** Traditional log file:**

```bash
sudo tail -f /var/log/nagios/nagios.log
# or
sudo tail -f /usr/local/nagios/var/nagios.log
```

*** If Nagios is managed by systemd:**

```bash

sudo journalctl -u nagios -f

```

You can grep for specific hosts/services:

```bash

sudo tail -f /var/log/nagios/nagios.log | grep <hostname>

```

 # 13 — What are the advantages of using Nagios?

Short list of key advantages:

*** **Mature & stable**: long-time proven monitoring solution.**

*** **Extensible plugin ecosystem**: thousands of plugins and community checks.**

*** **Flexible configuration**: granular checks for hosts/services, dependencies, escalations.**

*** **Alerting and notifications**: multiple notification channels (email, SMS through plugins).**

*** **Supports distributed monitoring**: NRPE, NCPA for remote checks.**

*** **Performance data**: can feed RRD/graphing tools for trends.**

*** **Customizable**: custom checks, event handlers, and notifications.**

*** **Lightweight**: low overhead agentless checks where appropriate.**

*** **Large community & documentation**: many integrations and guides.**

If you want, I can convert this into a printable cheat-sheet (one-page) with the exact commands you use most often, or give real examples for a particular Kubernetes YAML or Nagios config file you have. Which would be most useful?

Conclusion: In this week we learnt how to deploy pods in minikube and how to monitor our local system using Nagios tool and will know how to create the aws free trier account.