1. Ansible Command to Display All Configuration for a Host

To display all Ansible facts for a host, you can use the following Ansible command:

ansible -m setup <host>

Replace <host> with the actual hostname or group of hosts you want to query.

2.To run logrotate on all machines using Ansible, you can create a playbook that sets up the cron job on each machine. Here’s a step-by-step guide:

Create an Ansible Playbook: Create a file named logrotate\_cron.yml with the following content:

---

- name: Configure logrotate cron job on all machines

hosts: all

become: yes

tasks:

- name: Ensure logrotate cron job is present

cron:

name: "logrotate"

minute: "\*/10"

hour: "2-4"

job: "/usr/sbin/logrotate /etc/logrotate.conf"

Run the Playbook: Execute the playbook using the ansible-playbook command:

ansible-playbook -i inventory logrotate\_cron.yml

Make sure to replace inventory with the path to your inventory file that lists all the machines.

Inventory File: Your inventory file should list all the machines you want to manage. Here’s an example of what it might look like:

[all]

machine1.example.com

machine2.example.com

machine3.example.com

This setup will ensure that the cron job to run logrotate every 10 minutes between 2 AM and 4 AM is configured on all specified machines.

3 1. Create the Ansible Inventory File

Create a file named hosts (or inventory) and populate it with the information for your servers.

ini

Copy code

[ntp\_servers]

ntp-server ansible\_host=13.232.182.64

[monitoring]

nagios-server ansible\_host=13.232.39.78

[all:vars]

ansible\_user=ubuntu

ansible\_ssh\_private\_key\_file=~/.ssh/your-key.pem

2. Create the Ansible Playbook

Create a file named deploy\_ntp\_and\_nagios.yml with the following content:

yaml

Copy code

---

- name: Deploy NTP configuration and Nagios monitoring templates

hosts: ntp\_servers

become: true

tasks:

- name: Install NTP package

apt:

name: ntp

state: present

- name: Configure NTP

copy:

dest: /etc/ntp.conf

content: |

tinker panic 0

restrict default nomodify notrap nopeer noquery

restrict -6 default kod nomodify notrap nopeer noquery

restrict 127.0.0.1

restrict ::1

server 192.168.0.252 minpoll 4 maxpoll 8

server 192.168.0.253 minpoll 4 maxpoll 8

server 192.168.0.0 # local clock

fudge 192.168.0.0 stratum 10

driftfile /var/lib/ntp/drift

keys /etc/ntp/keys

notify: Restart NTP service

- name: Ensure NTP is running and enabled

service:

name: ntp

state: started

enabled: true

handlers:

- name: Restart NTP service

service:

name: ntp

state: restarted

- name: Deploy Nagios monitoring templates

hosts: monitoring

become: true

tasks:

- name: Install Apache and Nagios

apt:

name:

- apache2

- build-essential

- libgd-dev

- openssl

- libssl-dev

- unzip

state: present

- name: Install Nagios Core

command: /bin/bash -c 'cd /tmp && wget https://assets.nagios.com/downloads/nagioscore/releases/nagios-4.4.6.tar.gz && tar -zxvf nagios-4.4.6.tar.gz && cd nagios-4.4.6 && ./configure --with-httpd-conf=/etc/apache2/sites-enabled && make all && make install-groups-users && usermod -a -G nagios www-data && make install && make install-daemoninit && make install-commandmode && make install-config && make install-webconf'

- name: Configure Apache for Nagios

apache2\_module:

name: rewrite

state: present

- name: Configure Apache for CGI

apache2\_module:

name: cgi

state: present

- name: Create Nagios admin user

command: htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin

args:

creates: /usr/local/nagios/etc/htpasswd.users

- name: Restart Apache

service:

name: apache2

state: restarted

- name: Start Nagios

service:

name: nagios

state: started

enabled: true

- name: Deploy Nagios configuration for NTP servers

copy:

dest: /usr/local/nagios/etc/objects/ntp\_hosts.cfg

content: |

define host {

host\_name ntp-server

address 13.232.182.64

check\_command check-ping

active\_checks\_enabled 1

passive\_checks\_enabled 1

}

define service {

service\_description ntp\_process

host\_name ntp-server

check\_command check\_ntp

check\_interval 10

}

notify: Restart Nagios

handlers:

- name: Restart Nagios

service:

name: nagios

state: restarted

3. Run the Playbook

To execute the playbook, use the following command:

ansible-playbook -i hosts deploy\_ntp\_and\_nagios.yml

**docker/kubernets**

1.To prepare a docker-compose.yml file for an Nginx server with the specified requirements, you'll need to configure the following:

Persistent Logging: To ensure Nginx logs survive between container restarts, you can use a Docker volume to store the logs outside the container's filesystem.

Custom Network Configuration: To use a specific network bridge subnet (172.20.8.0/24), you'll need to define a custom network in the Docker Compose file.

Here’s a sample docker-compose.yml file that meets these requirements:

yaml

Copy code

version: '3.8'

services:

nginx:

image: nginx:latest

container\_name: nginx\_server

ports:

- "80:80"

volumes:

- nginx\_logs:/var/log/nginx

networks:

custom\_bridge\_network:

ipv4\_address: 172.20.8.2

networks:

custom\_bridge\_network:

driver: bridge

ipam:

config:

- subnet: 172.20.8.0/24

volumes:

nginx\_logs: {local-path}

Steps to Use This docker-compose.yml:

Create a Directory for the Project:

bash

Copy code

mkdir nginx\_docker

cd nginx\_docker

Create the docker-compose.yml File:

Create a file named docker-compose.yml in the nginx\_docker directory and copy the above content into it.

Start the Docker Compose Setup:

bash

Copy code

docker-compose up -d

Verify the Setup:

Check Logs Persistence:

bash

Copy code

docker-compose exec nginx\_server ls /var/log/nginx

Inspect the Network:

bash

Copy code

docker network inspect nginx\_docker\_custom\_bridge\_network

2.To identify the reason for a pod restart in the project “internal” under the namespace “production,” you can use the following Kubernetes command:

kubectl describe pod <pod-name> -n production

Replace <pod-name> with the actual name of the pod you’re investigating. This command will provide detailed information about the pod, including events that might indicate why the pod was restarted.

If you need to find the pod name first, you can list all pods in the “production” namespace with:

kubectl get pods -n production

Once you have the pod name, you can use the describe command to get more details. Let me know if you need any more help with Kubernetes!

3. From the Kubernetes configuration perspective, the following are the possible reasons for the java-app pod restarting at random:

1. Memory Limit Exceeded

The java-app container has a memory limit of 1500Mi.

The JVM's Xmx setting is 1000M, which means the maximum heap size is 1000Mi.

However, the container also uses non-heap memory (e.g., Metaspace, stack, native memory). This additional memory usage can cause the total memory consumption to exceed 1500Mi, leading to an OutOfMemoryError and the pod being killed by the Kubernetes OOM (Out-Of-Memory) killer.

2. CPU Limit Reached

The java-app container has a CPU limit of 2000 millicores (2 cores).

If the container attempts to use more CPU than this limit, it will be throttled, which can lead to performance issues.

If the application is CPU-bound and is being throttled excessively, it may lead to timeouts or errors, causing the application to crash and restart.

3. CPU Request and Throttling

The CPU request for the java-app container is set to 1000 millicores (1 core).

If the node is under CPU pressure, the container might not get enough CPU cycles, leading to performance degradation and potentially causing the application to restart.

4. Resource Contention

The pod consists of multiple containers: java-app, java-app-logrotate, java-app-fluentd, and mongos.

The total memory usage of all containers is close to the memory limit of the pod:

java-app: 951Mi

java-app-logrotate: 45Mi

java-app-fluentd: 84Mi

mongos: 62Mi

Total: 951Mi + 45Mi + 84Mi + 62Mi = 1142Mi

While the total is below the limit, spikes in memory usage from any of these containers could push the total above the 1500Mi limit, leading to the pod being killed.

5. JVM Settings

The Xmx setting of 1000M for the JVM may not be sufficient, leading to frequent garbage collection or OutOfMemoryError.

Improper JVM tuning can cause the application to crash and restart.

6. Node Resource Pressure

If the node running the java-app pod is under resource pressure (high CPU or memory usage), Kubernetes might evict the pod or it may experience resource contention.