**DevOps Case Study**

Below are some questions we would like you to work on. We expect to receive your feedback within 5 business days. Please feel free to contact us if you found the questions are unclear.

Configuration management

Suggested environment: Ubuntu 20 LTS, ansible 2.9.16, puppet 4.5 or above.

1. **Which ansible command can display all ansible\_ configuration for a host.**

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

**ansible -m setup <host>**

1. **Please configure a cron job that runs logrotate on all machines every 10 minutes between 2h - 4h.**

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

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

**[all]**

**machine1.example.com**

**machine2.example.com**

**machine3.example.com**

**[all:vars]**

**ansible\_user=ubuntu**

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

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**

1. Please deploy ntpd package to the following 3 servers:

app-vm1.fra1.internal (192.168.0.2)

db-vm1.fra1.db (192.168.0.3)

web-vm1.fra1.web (192.168.0.4)

with custom config of /etc/ntpd.conf:



We also need to deploy monitoring template onto our nagios server “monitoring.fra1.internal”, each of the above machines should use the following nagios templates:



**Ans**: **Create the Ansible Inventory File**

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

**[ntp\_servers]**

**app-vm1.fra1.internal ansible\_host=192.168.0.2**

**db-vm1.fra1.db ansible\_host=192.168.0.3**

**web-vm1.fra1.web ansible\_host=192.168.0.4**

**[monitoring]**

**monitoring.fra1.internal\_host=<ip>**

**[all:vars]**

**ansible\_user=ubuntu**

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

**Create the Ansible Playbook**

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

**---**

**- 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 app-vm1.fra1.internal**

**address 192.168.0.2**

**check\_command check-ping**

**active\_checks\_enabled 1**

**passive\_checks\_enabled 1**

**}**

**define service {**

**service\_description ntp\_process**

**host\_name app-vm1.fra1.internal**

**check\_command check\_ntp**

**check\_interval 10**

**}**

**define host {**

**host\_name db-vm1.fra1.db**

**address 192.168.0.3**

**check\_command check-ping**

**active\_checks\_enabled 1**

**passive\_checks\_enabled 1**

**}**

**define service {**

**service\_description ntp\_process**

**host\_name db-vm1.fra1.db**

**check\_command check\_ntp**

**check\_interval 10**

**}**

**define host {**

**host\_name web-vm1.fra1.web**

**address 192.168.0.4**

**check\_command check-ping**

**active\_checks\_enabled 1**

**passive\_checks\_enabled 1**

**}**

**define service {**

**service\_description ntp\_process**

**host\_name web-vm1.fra1.web**

**check\_command check\_ntp**

**check\_interval 10**

**}**

**dest: /etc/nagios/conf.d/ntp\_hosts.cfg**

**owner: nagios**

**group: nagios**

**mode: '0644'**

**- name: Restart Nagios service**

**service:**

**name: nagios**

**state: restarted**

**Run the Playbook**

To execute the playbook, use the following command:

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

# **Docker/Kubernetes**

**Suggested environment: Ubuntu 20 LTS, docker 19 or above**

1. **Prepare a docker-compose for a nginx server.**

**Requirements:**

* **nginx logs need to survive between nginx container restarts**
* **docker should use network bridge subnet 172.20.8.0/24**

**Ans**: To prepare a docker-compose.yml file for an Nginx server with the specified requirements, need to configure the following:

**Persistent Logging**: To ensure Nginx logs survive between container restarts, It 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), need to define a custom network in the Docker Compose file.

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

**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: <path>**

**Steps to Use This docker-compose.yml:**

Create a Directory for the Project:

**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:

docker-compose up -d

This command will start the Nginx server with the specified network configuration and persistent logging.

1. **Which Kubernetes command you will use to identify the reason for a pod restart in the project "internal" under namespace "production".**

**Ans:** To identify the reason for a pod restart in the project "internal" under the namespace "production," we can use the following Kubernetes commands:

Describe the Pod: This will provide detailed information about the pod, including recent events, which often contain information about why a pod was restarted.

**kubectl describe pod <pod-name> -n production**

Check Pod Logs: Logs from the containers within the pod can provide insight into what happened just before the restart.

**kubectl logs <pod-name> -n production --previous**

Get Pod Events: Events can provide detailed information about the lifecycle of the pod, including restarts.

**kubectl get events -n production --field-selector involvedObject.name=<pod-name>**

Get Pod Status: we can also check the status of the pod to see if there are any specific conditions indicating why it restarted.

**kubectl get pod <pod-name> -n production -o yaml**

**2. Consider the followings:**

**POD NAME CPU(cores) MEMORY(bytes)**

**java-app-7d9d44ccbf-lmvbc java-app 3m 951Mi**

**java-app-7d9d44ccbf-lmvbc java-app-logrotate 1m 45Mi**

**java-app-7d9d44ccbf-lmvbc java-app-fluentd 1m 84Mi**

**java-app-7d9d44ccbf-lmvbc mongos 4m 62Mi**

**Application pod has the following resource quota:**

* **Memory request & limit: 1000 & 1500**
* **CPU request & limit: 1000 & 2000**
* **Xmx of 1000M**

**Java-app keep restarting at random. From Kubernetes configuration perspective, what are the possible reasons for the pod restarts?**

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

**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.

**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.

**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.

**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.

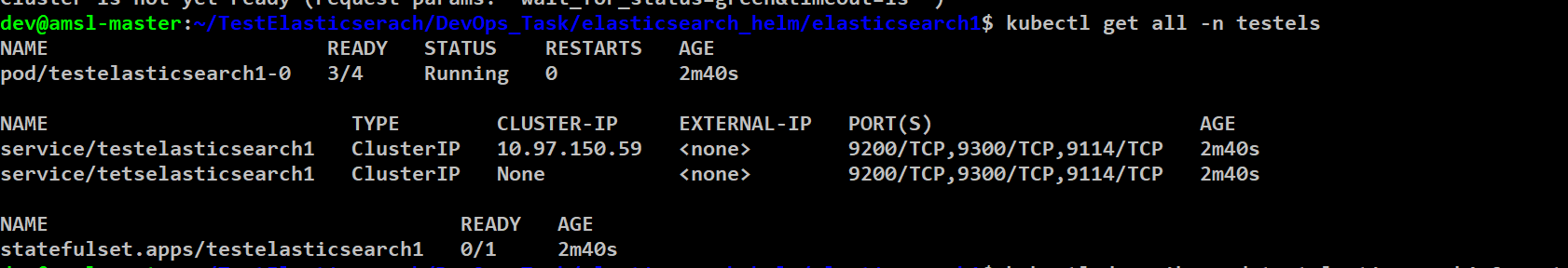
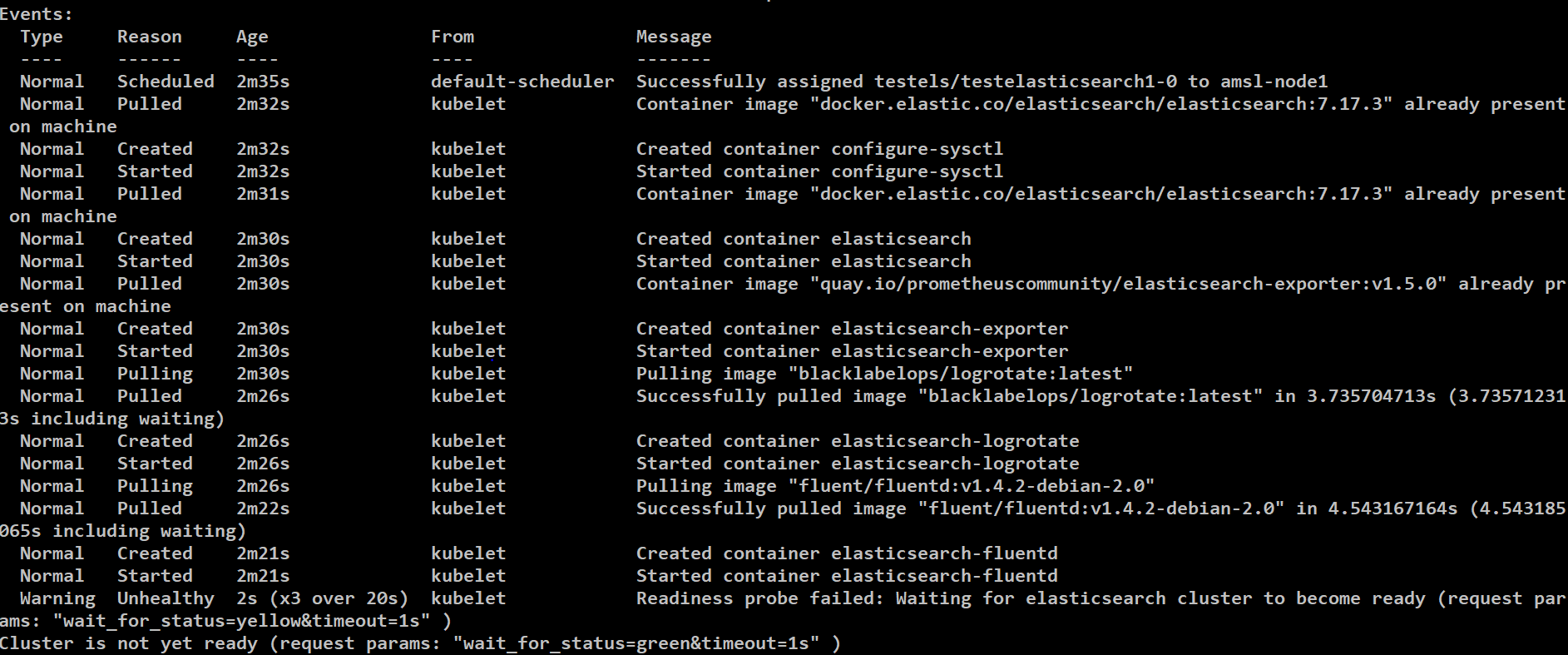
**Node Resource Pressure:**

* 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.

# **Helm**

**Please use the accompanied elasticsearch helm template to create a Kubernetes deployment of elasticsearch.**

**Provide a screenshot & deployment yaml of the resultant deployment in Kubernetes.**

**Ans: **

While deploying a single-node Elasticsearch cluster on Kubernetes using Helm charts, the pod is failing its readiness probe due to the Elasticsearch cluster not reaching a "green" status. I tried to resolve the issue ,still can figure out  
Al though I have created values.yaml file and have some configured statefulset, All document given in below link

<https://github.com/yeatun/DevOps_Task/tree/main/Helm>

# **Metrics**

1. **Explain how Prometheus work.**
2. **How do you create custom Prometheus alerts and alerting rules for Kubernetes monitoring? Provide an example alert rule and its configuration.**
3. **What is the Prometheus query you can use in Granfana to properly show usage trend of an application metric that is a counter?**

**Ans:**

**How Prometheus Works**

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. Here's an overview of how Prometheus works:

* **Data Collection (Scraping):** Prometheus collects metrics from targets (applications, services, nodes) by scraping HTTP endpoints that expose metrics in a specific format. Each target provides a set of metrics, typically at /metrics.
* **Data Storage:** The collected metrics are stored in a time-series database within Prometheus. Each metric is stored with a timestamp, a metric name, and a set of key-value pairs (labels).
* **Querying:** Prometheus provides a powerful query language called PromQL (Prometheus Query Language) to query and analyze the collected metrics. PromQL can be used to aggregate, slice, and dice the time-series data.
* **Alerting:** Prometheus can evaluate rules and trigger alerts based on the metrics. It uses Alertmanager to handle alerts, which can then be sent to different receivers like email, Slack, PagerDuty, etc.
* **Visualization:** Prometheus integrates with visualization tools like Grafana, allowing users to create dashboards and visualize the metrics.

**Creating Custom Prometheus Alerts and Alerting Rules for Kubernetes Monitoring**

To create custom Prometheus alerts and alerting rules, follow these steps:

* Define Alerting Rules: Create a configuration file that defines the alerting rules. Each rule specifies a PromQL expression that triggers the alert, along with conditions and labels.
* Load Rules into Prometheus: Add the rules file to the Prometheus configuration, ensuring it is loaded when Prometheus starts.
* Configure Alertmanager: Set up Alertmanager to handle and route the alerts to the desired notification channels.

**Example Alert Rule and Configuration**

**Alert Rule Definition (e.g., alert.rules.yml):**

**groups:**

**- name: example-alerts**

**rules:**

**- alert: HighCPUUsage**

**expr: sum(rate(container\_cpu\_usage\_seconds\_total{image!=""}[5m])) by (pod) > 0.8**

**for: 5m**

**labels:**

**severity: warning**

**annotations:**

**summary: "High CPU usage detected on pod {{ $labels.pod }}"**

**description: "CPU usage on pod {{ $labels.pod }} is above 80% for more than 5 minutes."**

**Prometheus Configuration (prometheus.yml):**

**rule\_files:**

**- "alert.rules.yml"**

**alerting:**

**alertmanagers:**

**- static\_configs:**

**- targets:**

**- alertmanager:9093**

**Alertmanager Configuration (alertmanager.yml):**

**global:**

**resolve\_timeout: 5m**

**route:**

**receiver: 'slack-notifications'**

**receivers:**

**- name: 'slack-notifications'**

**slack\_configs:**

**- api\_url: 'https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX'**

**channel: '#alerts'**

**text: "{{ range .Alerts }}{{ .Annotations.summary }}\n{{ .Annotations.description }}\n{{ end }}"**

**3. Prometheus Query in Grafana to Show Usage Trend of a Counter Metric**

When dealing with counter metrics in Prometheus, it’s important to use the rate or irate function to calculate the rate of change over time. This helps in understanding the usage trend.

**Example Query for a Counter Metric**

**Step-by-Step in Grafana**

* Add a New Panel: In your Grafana dashboard, add a new panel.
* Configure Data Source: Select Prometheus as the data source.
* Enter the Query: In the query editor, enter the PromQL query:

**rate(my\_application\_metric\_total[5m])**

* Set Visualization Options: Configure the visualization options to best display the trend, such as line chart, time series, etc.

**Databases**

**Suggested environment: Cassandra 4.0 or above, mongo 4.4.0 or above**

## **Cassandra**

**Query to db cluster returns different result each time. Users reported query result has data records that they deleted days ago.**

**Explain what the likely reason for the behavior and how to avoid it.**

**Ans**: It may be issues with data consistency in the Cassandra cluster. In Cassandra, such issues are often related to eventual consistency, replication, and how read and write operations are managed. Here's an explanation of the likely reasons for the behavior and how to avoid it:

Likely Reasons for Inconsistent Query Results

**Eventual Consistency:**

Cassandra is designed for eventual consistency, which means that all replicas will eventually become consistent, but they might not be consistent at any given moment. If read and write operations are not carefully managed, it can lead to scenarios where queries return stale or deleted data.

**Read Repair and Hinted Handoff:**

**Read Repair**: When a read request is made, Cassandra tries to fix any inconsistencies among replicas by reading from multiple nodes and writing back any differences. If this process fails or is incomplete, it can lead to stale data being read.

**Hinted Handoff**: When a node is down, Cassandra stores a hint on another node to be delivered later. If the hint is not delivered properly, it can cause inconsistencies.

**Tombstones and Compaction:**

Deleted data in Cassandra is marked with tombstones. If compaction processes are not running efficiently, tombstones might not be processed in a timely manner, causing deleted data to appear in query results.

**Consistency Levels:**

Cassandra allows different consistency levels for reads and writes, such as ONE, QUORUM, ALL, etc. If the consistency level used for reads is lower than the consistency level used for writes, it might result in reading stale data from nodes that have not yet received the latest updates.

How to Avoid Inconsistent Query Results

**Increase Consistency Levels:**

Use a higher consistency level for both reads and writes. For example, using QUORUM for both read and write operations ensures that a majority of replicas are consistent before acknowledging the operation.

SELECT \* FROM table\_name USING CONSISTENCY QUORUM;

INSERT INTO table\_name (...) VALUES (...) USING CONSISTENCY QUORUM;

**Ensure Efficient Compaction:**

Regularly monitor and optimize the compaction process to ensure that tombstones are properly handled and old data is purged from the system.

**Enable and Monitor Read Repair:**

Ensure that read repair is enabled and functioning correctly. Read repair helps in synchronizing the data among replicas during read operations.

**read\_repair\_chance: 0.1**

**Repair and Rebalance the Cluster:**

Regularly run nodetool repair to ensure that data is consistent across all nodes. This process helps in synchronizing data across the cluster and fixing any inconsistencies.

**nodetool repair**

**Tune Hinted Handoff Settings:**

Ensure that hinted handoff is properly configured and monitored. This can help in reducing the window during which a node might be missing updates.

hinted\_handoff\_enabled: true

hinted\_handoff\_throttle\_in\_kb: 1024

max\_hint\_window\_in\_ms: 10800000 # 3 hours

## **Mongo**

**We have mongodb replicaset\_1 with the following db and collections.**

****

**A sample record from company\_name:**

****

**Performance is bad as the hardware of replicaset\_1 is not capable to handle the database sanfrancisco. We added a new replicaset\_2.**

**Please provide all steps required to shard the collection sanfrancisco.company\_name based on \_id.**

**Ans:**

**1. Prepare the Sharding Environment**

Ensure that we have a config server and at least one mongos instance running.

**2. Connect to the mongos Instance**

Connect to our mongos instance using the MongoDB shell.

**mongo --host <mongos-host>:<mongos-port>**

**3. Enable Sharding on the Database**

Enable sharding on the sanfrancisco database.

**sh.enableSharding("sanfrancisco")**

**4. Add the Shards**

Add your existing replica sets as shards. Replace <replica1-host>:<port> and <replica2-host>:<port> with the actual hostnames and ports.

**sh.addShard("replicaset\_1/<replica1-host>:<port>")**

**sh.addShard("replicaset\_2/<replica2-host>:<port>")**

**5. Shard the Collection**

Enable sharding on the company\_name collection based on the \_id field.

**sh.shardCollection("sanfrancisco.company\_name", { "\_id": 1 })**

**6. Verify Sharding**

Verify that the collection is sharded correctly.

**sh.status()**

**with Placeholder Values**

If replica1-host is rs1.example.com with port 27017, and replica2-host is rs2.example.com with port 27017, the commands would be:

mongo --host mongos.example.com:27017

sh.enableSharding("sanfrancisco")

sh.addShard("replicaset\_1/rs1.example.com:27017")

sh.addShard("replicaset\_2/rs2.example.com:27017")

sh.shardCollection("sanfrancisco.company\_name", { "\_id": 1 })

sh.status()