Tab 1

## **ELK Stack Assessment Questions**

## **Module 1: Introduction to ELK Stack**

## **Why ELK?**

1. What is the ELK stack and what does each letter stand for?
2. Why do organizations choose the ELK stack for data analytics?
3. Explain the key problems ELK solves in log management.
4. Describe two scenarios where using ELK is beneficial.
5. How does ELK facilitate real-time data analysis?
6. What are the primary advantages of centralized logging?

## **Use Cases & Benefits**

1. Name three common use cases for the ELK stack.
2. How is the ELK stack used in security monitoring?
3. Describe how ELK can help with troubleshooting application errors.
4. What are the business benefits of ELK for infrastructure monitoring?

## **Architecture of ELK Stack**

1. Illustrate the high-level architecture of the ELK stack.
2. Explain data flow between Logstash, Elasticsearch, and Kibana.
3. What roles do nodes and clusters play in ELK architecture?
4. Describe the ingestion pipeline from data source to visualization.

## **Components Overview**

## **Elasticsearch**

1. What is Elasticsearch and what type of database is it?
2. List the fundamental features of Elasticsearch.
3. How does Elasticsearch store and retrieve data?

## **Logstash**

1. What is Logstash's main function in the ELK stack?
2. Describe the data processing pipeline of Logstash.

## **Kibana**

1. What is the purpose of Kibana?
2. Mention the types of visualizations supported by Kibana.

## **Module 2: Setting Up the ELK Stack**

## **Installation**

1. What are prerequisites for installing ELK stack on Linux?
2. Outline steps to install Elasticsearch on Linux.
3. How do you install Logstash on an Ubuntu server?
4. Describe the process of installing Kibana.
5. What does an on-premises installation of ELK involve?
6. Which ports are used by Elasticsearch, Logstash, and Kibana by default?

## **Managing Clusters**

1. Define an Elasticsearch cluster.
2. What are nodes in Elasticsearch and how are they managed?
3. How do you add a new node to an existing ELK cluster?
4. What is node discovery in Elasticsearch?

## **Dev Tools & Console**

1. What are the Kibana Dev Tools?
2. Give an example of a basic query run in Kibana’s console.
3. How can developers troubleshoot Elasticsearch queries using Dev Tools?

## **Module 3: Logstash – Data Ingestion & Parsing**

## **Introduction to Logstash**

1. What is Logstash and where does it fit in the ELK stack?
2. List three input data sources supported by Logstash.

## **Role in the ELK Stack**

1. How does Logstash enhance data flowing into Elasticsearch?

## **Logstash vs Filebeat**

1. When should you use Logstash over Filebeat?
2. Under what circumstances is Filebeat preferable?

## **Logstash Architecture**

1. Describe the plugin architecture of Logstash.
2. What is the purpose of input, filter, output, and codec plugins in Logstash?

## **Logstash Template Structure**

1. Explain the structure of a typical Logstash configuration file.

## **Data Indexing Demos**

1. How do you index a CSV file using Logstash?
2. What is the approach to ingesting log files from a folder path?
3. Outline the steps to index a JSON file with Logstash.

## **Multiple Inputs/Outputs**

1. How do you configure Logstash for multiple input and output streams?

## **Module 4: Kibana – Visualization & Dashboards**

## **Introduction to Kibana**

1. What features does Kibana provide for data visualization?

## **Loading & Discovering Data**

1. How do you load sample data into Kibana?
2. How does the Discover tab help users analyze raw data?

## **Building Visualizations & Dashboards**

1. Describe the steps involved in creating a dashboard with multiple panels, filters, and custom links in Kibana.

Tab 2

## **Module 1: Advanced Introduction to ELK Stack**

1. Explain the CAP theorem as it relates to Elasticsearch clustering, and discuss how you would mitigate the risks of partition tolerance impacting availability.
2. For a high-throughput log analytics system, what are the possible bottlenecks in the ELK stack, and how would you monitor and resolve them?
3. In a regulated environment (e.g., PCI DSS), what ELK architectural decisions must be made regarding data retention and auditability?
4. Design a multi-region ELK deployment: what are the primary concerns, and how would you address cross-region data consistency?
5. How can you secure inter-component (Logstash, Elasticsearch, Kibana) communications in an untrusted network? List specific protocols and configuration mechanisms.

## **Module 2: ELK Stack Advanced Setup and Administration**

1. Describe the process of automating full ELK stack deployment (Elasticsearch, Logstash, Kibana) using configuration management tools (e.g., Ansible, Puppet, Chef).
2. When performing a rolling upgrade of Elasticsearch, what are the steps and precautions to ensure zero downtime and data consistency?
3. Discuss best practices for configuring Elasticsearch heap space and JVM options for nodes with differing roles (master, data, ingest, etc.).
4. Explain the implications of split-brain scenarios in Elasticsearch clusters and detail your recovery strategy.
5. How would you implement centralized authentication and access control across the ELK stack components, compatible with Active Directory/LDAP?
6. Which monitoring tools or APIs would you use to detect and diagnose cluster performance degradation before it becomes a crisis?
7. How would you automate hot-warm-cold index lifecycle management, and what benefits does this bring to company operations?
8. DevOps often use Docker and Kubernetes. Outline the key configuration steps for deploying a resilient, auto-scaling ELK stack on Kubernetes.
9. After a sudden disk failure on an Elasticsearch node, what steps would you take to restore service integrity and avoid further data loss?
10. Discuss how index sharding and replication factors should be tuned for a cluster expected to handle petabyte-scale log ingestion.

## **Module 3: Logstash – Advanced Data Ingestion & Parsing**

1. Given a scenario where Logstash requires ingesting logs from distributed, unreliable sources, how would you ensure pace control and data integrity?
2. Compare and contrast persistent queues and dead letter queues in Logstash. When would you use each, and how would you configure them?
3. Describe the impact and mitigation strategies of a memory or CPU spike caused by a poorly written Logstash filter plugin.
4. When ingesting terabytes of daily log data with both Logstash and Filebeat, how would you architect the data flow for maximum resiliency and efficiency?
5. How would you design a Logstash pipeline to handle multi-format log ingestion (e.g., mixed JSON, CSV, and syslog), and what is the trade-off in pipeline complexity vs. maintainability?
6. Explain how to use conditionals and mutation filters in Logstash to normalize data fields across different input formats.
7. A third-party Logstash plugin fails during pipeline startup. What diagnostic steps do you perform, and how can you safeguard production pipelines?
8. You suspect a Logstash pipeline introduces latency. Detail a step-by-step method for isolating and resolving bottlenecks.
9. Describe how to design Logstash pipelines for at-least-once and exactly-once data delivery semantics.
10. Discuss plugin version compatibility; how would you manage and test plugin upgrades to avoid downtime or data loss?

## **Module 4: Elasticsearch – Scaling, Performance, and Operations**

1. How do you mitigate mapping explosion in dynamic log environments, and what are the operational impacts if not managed?
2. Elasticsearch is reporting frequent circuit breaker exceptions. What do these mean, and how do you address them short and long term?
3. Describe a backup and disaster recovery plan for production Elasticsearch indices under 24/7 SLA constraints.
4. Given high cardinality aggregations on large datasets, what query, index, and hardware optimizations can you perform?
5. Explain the role and configuration of ILM (Index Lifecycle Management) for indices with unpredictable data retention requirements.
6. A developer complains that search results are stale. How could index refresh settings or replica lag contribute, and how do you adjust these settings?
7. Explain how you would reindex a petabyte-scale index with zero downtime, and the challenges you anticipate.
8. Describe how to enable encryption-at-rest for Elasticsearch indices and the operational implications.
9. Under log surges (e.g., DDoS attack), what measures protect indexing throughput and ensure search performance?

## **Module 5: Kibana – Security, Visualizations, and Dashboards**

1. If Kibana's dashboards are timing out due to slow Elasticsearch queries, how do you debug and optimize both the query and the dashboard?
2. A sensitive dashboard must be accessible only to select users. How would you implement role-based access in Kibana?
3. Describe the best way to create parameterized (DRY) visualizations in Kibana for use in multiple dashboards.
4. DevOps teams want a real-time infrastructure health overview. How would you architect Kibana dashboards for actionable monitoring and alerting?
5. Custom plugins are needed for new visualization types. Describe the process and deployment pipeline you would set up for Kibana plugin development and updates.
6. How do you securely expose Kibana to external users while protecting Elasticsearch from direct public access?
7. A compliance auditor requests a change history for dashboards and visualizations. What options does Kibana offer for auditability, and how would you supplement if native options are limited?
8. Explain the use of Kibana's Canvas or Lens for visualizing custom KPIs, and how you’d integrate them with external data sources.

## **Module 6: Troubleshooting, Upgrades, and Advanced Operations**

1. Describe your strategy for blue-green or canary deployments of new ELK configurations or plugins without user disruption.
2. Elasticsearch’s JVM heap usage keeps growing and triggers out-of-memory errors. Walk through your diagnostic and remediation steps.
3. Network partition occurs between Kibana and some Elasticsearch nodes. What symptoms appear in user dashboards, and how do you resolve?
4. A recurring complaint is slow ingest or delayed dashboards during specific business hours. How would you investigate and present findings to management?
5. During an upgrade, an Elasticsearch index becomes read-only due to disk watermark. What steps do you take to resume indexing?
6. Describe how you would set up CI/CD pipelines for Logstash configuration management and rollout, handling secrets securely.
7. If a developer wants production-like ELK stack test data for local debugging, how do you provide it safely and cost-effectively?
8. For compliance, you’re required to export all ELK configurations and mappings regularly. How do you automate this task?

Tab 3

## **Module 1: Advanced Introduction to ELK Stack**

1. CAP theorem & Elasticsearch:  
Elasticsearch clusters prioritize availability (A) and partition tolerance (P) over consistency (C) due to distributed nature. Split-brain situations may lead to inconsistency. Mitigation: set minimum\_master\_nodes (older versions) or use voting exclusions/quorum, use smaller odd numbers of master nodes, monitor cluster health.

2. ELK Bottlenecks & Resolution:  
Bottlenecks: disk I/O (Elasticsearch), memory/heap (Elasticsearch/Logstash), network, CPU (filters, queries), JVM GC, mapping explosions. Monitoring: Stack Monitoring, node stats, hot\_threads API, Logstash monitoring APIs. Resolution: Scale-out, shard tuning, heap tuning, demote data, optimize pipelines.

3. Regulated ELK Architectural Decisions:  
Decisions: secure storage (encryption), fine-grained access control, audit logging, retention (ILM), on-prem deployment, regular backups, immutable audit logs.

4. Multi-region ELK Primary Concerns:  
Concerns: network latency, data consistency, cross-region replication (CCR), election domains, index lifecycle policies per region. Use CCR, cross-region snapshots, region-local read replicas, dedicated master-eligible nodes per region.

5. Securing Inter-component Communications:  
Protocols: TLS/SSL for REST and transport layers, HTTPS for Kibana, filebeat/logstash beats, authentication via x-pack/security, API keys, firewalled network, VPN/tunnels.

## **Module 2: Advanced Setup and Administration**

6. Automating ELK Deployment:  
Use Ansible, Puppet, or Chef to define playbooks/manifests for package installation, template config files, systemd services, secure secrets, automate licensing, monitor via playbook outcomes.

7. Rolling Upgrade Elasticsearch:  
Take snapshot, verify health, disable shard allocation, sequentially upgrade non-master nodes, then masters, then clients. Maintain quorum—ensure minimum of master-eligible nodes always up. Test API compatibility.

8. JVM Tuning by Node Role:  
Masters: smaller heap (4-8GB), high CPU, avoid data.  
Data nodes: large heap (≤50% RAM, ≤32GB), fast disk, high I/O.  
Ingest nodes: moderate heap/CPU.  
Set -Xms = -Xmx; monitor garbage collection.

9. Split-brain Scenarios:  
Happens if network partition and two master-eligible groups form. Set discovery.zen.minimum\_master\_nodes (before 7.x, now automatic voting). Recovery: restore network, manually elect correct master, re-integrate nodes, monitor data consistency.

10. Centralized Authn/Authz:  
Integrate with Elasticsearch x-pack security, configure LDAP/AD realms, roles mapping, configure Kibana/Logstash for same realm, restrict API access.

11. Cluster Monitoring:  
Use Elastic/X-Pack Monitoring, APIs: \_cat/health, \_cat/thread\_pool, Hot Threads, Prometheus exporters, alerting on node departures, queue backlogs.

12. Automate Hot-Warm-Cold ILM:  
Define ILM policies: hot (fast), warm (slower), cold (archival), automate allocation with index templates, rollovers, automated deletion. Improves performance+cost.

13. Kubernetes ELK Deployment:  
Use Helm charts/Operators, Persistent Volumes, StatefulSets, Service discovery, readiness/liveness probes, anti-affinity for nodes, configure resource limits.

14. Disk Failure Recovery:  
Mark node offline, replace disk, restore from snapshot if lost, let cluster rebalance. Ensure replication factor prevents data loss.

15. Sharding/Replication Tuning:  
Start with more (smaller) primary shards for parallelism, 2+ replicas for redundancy, monitor shard sizes (20-40GB ideal), rebalance regularly.

## **Module 3: Logstash – Advanced Data Ingestion & Parsing**

16. Distributed, Unreliable Ingest:  
Enable persistent queues, use filebeat as shipper for at-least-once, back-pressure with Lumberjack/Beats, monitor input/output rates.

17. Persistent vs Dead Letter Queues:  
Persistent queues buffer unprocessed events (reliability); enable in settings. DLQs catch events that failed output (debug/troubleshoot); configure path and monitor.

18. Plugin-caused Spikes:  
Check pipeline stats, isolate plugin via disabling/testing. Refactor logic, upgrade plugin, adjust filter worker threads, mock test input for bottlenecks.

19. Logstash+Filebeat Arch:  
Deploy Filebeat on edge nodes, lightweight parsing + ship to message queue/Logstash, then parse/enrich/index. Use backpressure, persistent/replica indices.

20. Multi-format Ingest:  
Use multiple input blocks, conditionals to decode/parse, use tags, increase code/test complexity. Excessive complexity = harder to maintain.

21. Field Normalization:  
Use mutate filter for renames, convert datatypes, conditional logic (if ... else ...), use grok for parsing, test with sample data.

22. Plugin Startup Failure:  
Review logs, check version compatibility, rollback plugin, isolate bad config, set monitor/alert, use pipeline-level version control.

23. Logstash Pipeline Latency:  
Use Monitoring API, check event durations per stage, narrow filters, index slow stage, optimize regex, increase CPUs, test on isolated data.

24. Delivery Semantics:  
Enable persistent queues (for at-least-once), add idempotency on output. For exactly-once: use Kafka as input/output plus unique IDs, but Logstash is not strictly exactly-once.

25. Plugin Upgrades:  
Test in staging, review compatibility documents, automate testing (unit/integration), utilize pipeline versioning, monitor errors during rollout.

## **Module 4: Elasticsearch – Scaling, Performance, and Operations**

26. Mapping Explosion Mitigation:  
Disable dynamic mapping: set "dynamic":"strict", set field limits (index.mapping.total\_fields.limit), normalize field names, monitor mappings via APIs.

27. Circuit Breaker Exceptions:  
Occur when requests might exhaust node memory. Lower breaker limits, optimize query size/aggregations, add nodes, monitor heap usage, prevent heavy aggregations.

28. Backup/Disaster Recovery:  
Automated snapshots (S3/NFS repo), offsite storage, frequent snapshot scheduling, test restores, hot-warm architectures for fast resume, document runbooks.

29. High Cardinality Optimization:  
Use doc values, increase fielddata, optimize field types, pre-aggregate data, use composite aggregations, tune hardware (memory/disk), avoid complex regex, consider rollups.

30. ILM for Unpredictable Retention:  
Enable ILM with flexible policies (hot-warm-cold), allow for extension of life, use APIs for index state transitions, monitor and adjust as needed.

31. Stale Search Results Diagnosis:  
Index refresh interval too high, not frequent enough. Tune refresh\_interval, monitor search latency vs. freshness, adjust replication settings if lag occurs.

32. Zero Downtime Reindexing:  
Create new index, dual-write or alias swap pattern, reindex-while-ingest, backfill old data, monitor catch-up, atomic alias switch.

33. Encryption at Rest:  
Use OS-level (dm-crypt/LUKS) or Elasticsearch X-Pack native encryption. Operationally: impacts disk I/O, increases recovery complexity, may complicate snapshots.

34. Surges Protection:  
ILM to swap out older indices, slow down rate limiters at source, increase input node count, implement circuit breaker and bulk queue limits.

## **Module 5: Kibana – Security, Visualizations, Dashboards**

35. Slow Dashboards Debug:  
Isolate slow visualizations, inspect queries in Dev Tools, use profile API, optimize mapping and filters, improve index, reduce dashboard panel count, aggregate less.

36. RBAC in Kibana:  
Enable Spaces/roles (X-Pack/Elastic Stack security), assign privileges to users/groups, restrict dashboards via Spaces, test with non-admin accounts.

37. Parameterized Visualizations:  
Use saved searches, Kibana Lens, dashboard variables, controls, and filters. Template dashboards, embed with query parameters.

38. Real-time Infra Dashboards:  
Receive pipeline data (metrics/logs), create time-series panels, use Kibana Alerts, route alert actions, show quick-to-read status, color-code, minimize latency.

39. Custom Plugins for Visualization:  
Use Kibana plugin generator, develop React-based plugins, set up build/test pipeline, deploy via tarball/zip/plugin manager, automate via CI/CD pipelines.

40. Secure Kibana Exposure:  
Proxy (e.g., nginx) in front of Kibana, hide direct Elasticsearch endpoints, enforce HTTPS, enable authentication, IP restriction, use API keys.

41. Dashboard Auditability:  
Enable X-Pack audit logging, monitor saved object changes, use external version control (export configs), if needed use webhooks or backup scripts.

42. Advanced Visualization Integration:  
Use Canvas for custom layouts, Lens for flexible analysis, utilize external sources via plugins or integrations (e.g., Vega for external JSON).

## **Module 6: Troubleshooting, Upgrades, and Advanced Operations**

43. Blue-green/Canary Deployment Strategy:  
Deploy new stack in parallel, use traffic splitting/load balancer or alias swap, monitor traffic/errors, rollback on failures, automate deployment pipeline.

44. JVM OOM Diagnosis/Remediation:  
Analyze heap dumps, GC logs, check for mapping explosion, oversized aggregations. Short-term: restart/shard reallocation; long-term: refactor queries, tune heap.

45. Kibana-Elasticsearch Partition Symptoms:  
Dashboards fail to load, no data, errors in console. Fix: restore network, check node connections, failover to healthy endpoints, analyze logs.

46. Slow Ingest/Delayed Dashboards at Peak:  
Review ingest pipelines, node stats, index queue stats, check I/O, memory during peak, look for query hot spots, recommend scaling, ILM, or resource increases.

47. Index Becomes Read-Only:  
Triggered by exceeding disk watermark (cluster.routing.allocation.disk.watermark). Free disk, reset index.blocks.read\_only\_allow\_delete to false, monitor usage.

48. Logstash CI/CD Pipelines w/ Secrets:  
Store configs in VCS, use CI/CD pipelines to lint/test/rollout, handle secrets via Vault/Secret Manager/env variables, restrict pipeline permissions.

49. Production-like Test Data:  
Use index snapshot/restore to dedicated cluster, mask data, restrict access, replicate mapping/structure only, use sample downscaling tools.

50. Automate ELK Config/Mapping Backup:  
Use Elasticsearch and Kibana APIs to export mappings, templates, pipelines, dashboards via scheduled scripts; store in VCS or S3; automate with cron or CI/CD.

Tab 4

## **Elasticsearch: Commands & Configuration**

1. What command would you use to list all indices in your Elasticsearch cluster?
2. How would you use the \_cat/nodes API to view node roles and health in the cluster?
3. Demonstrate how to create a new index with a custom mapping using a cURL command.
4. Show the command to update the number\_of\_replicas for an existing index without restarting the cluster.
5. Which command changes a cluster from read-only mode back to read-write?
6. How do you apply a persistent cluster setting via the Elasticsearch REST API?
7. Update JVM heap size for Elasticsearch and verify the change.
8. Demonstrate how to add a new node to an existing cluster and validate successful joining.
9. What configuration in elasticsearch.yml sets the minimum number of master-eligible nodes?
10. How would you enable encryption in transit between Elasticsearch nodes?
11. Using API, create a user with read privileges on an index and test access.
12. Provide the cURL command to delete all documents from an index but preserve the index itself.
13. How would you configure cross-cluster search in Elasticsearch?
14. Using a command, refresh an index to make recent operations searchable immediately.
15. Demonstrate how to trigger a snapshot backup of selected indices using the API.
16. Show how to install a plugin (e.g., ingest-geoip) into Elasticsearch.
17. What command retrieves current shard allocation for a given index?
18. List the steps to tune index refresh intervals through both elasticsearch.yml and API.
19. How would you temporarily disable allocation of shards in the cluster?
20. How do you check and change the log level of Elasticsearch dynamically?

## **Kibana: Configuration & Command-Line Tasks**

1. Which command starts Kibana from the command line and sets a custom configuration path?
2. Show a snippet of kibana.yml to enable SSL on the Kibana server.
3. Describe how to configure Kibana to connect to a remote Elasticsearch instance.
4. Using a configuration file, enable role-based access control for dashboard access.
5. How would you configure Spaces in Kibana to provide isolated work areas?
6. Write the command to generate or update the encryption keys for securing session data in Kibana.
7. How do you configure Kibana to listen on all network interfaces instead of just localhost?
8. Show the steps to enable reporting features in Kibana.
9. Demonstrate how to configure logging level to debug in kibana.yml.
10. How would you provision and test a new connector from Kibana to an external alerting system?

## **Logstash: Commands, Pipelines & Configuration**

1. What command line is used to test a Logstash pipeline configuration file without actually running the pipeline?
2. Show how to reload a Logstash configuration automatically when files change.
3. Demonstrate creating a persistent queue for reliable ingestion using logstash.yml.
4. What command checks installed input/output/filter plugins in Logstash?
5. Describe editing pipelines.yml to add two parallel pipeline definitions.
6. Use a command to manually trigger Logstash with a specific configuration file for troubleshooting.
7. How do you set environment variables for secret management in Logstash pipeline configs?
8. Show a configuration snippet to output data to both Elasticsearch and a file in Logstash.
9. Provide the command for installing a new filter plugin in Logstash.
10. Demonstrate how to tail Logstash logs in real time for troubleshooting pipeline errors.

## **Advanced Administration & Real-World Troubleshooting**

1. Given a node out of disk, use commands to relocate shards and unblock writes on indices.
2. Use Kibana Dev Tools to issue a multi-index search query via the Console.
3. Rotate Elasticsearch access credentials via command and configuration.
4. Configure Logstash pipelines with conditionals that partition input data by hostname.
5. Apply a transient cluster setting in Elasticsearch; show how to revert it back.
6. List the steps to enable detailed audit logging in Elasticsearch via configuration.
7. Script a scheduled index deletion policy using Index Lifecycle Management (ILM).
8. Provide an example of configuring and verifying API key authentication for log shippers.
9. How would you automate exporting dashboards from Kibana for backup purposes?
10. Demonstrate a full-stack configuration change: Add a new log field in Logstash, map it in Elasticsearch, and render it on a Kibana dashboard.

Tab 5

## **Elasticsearch**

1. List all indices in the cluster:

text

curl -XGET "localhost:9200/\_cat/indices?v"

2. View nodes, roles, and health:

text

curl -XGET "localhost:9200/\_cat/nodes?v"

3. Create a new index with custom mapping:

text

curl -XPUT "localhost:9200/my-index" -H 'Content-Type: application/json' -d '{

"mappings": {

"properties": {

"timestamp": {"type": "date"},

"user": {"type": "keyword"}

}

}

}'

4. Update number\_of\_replicas for an index:

text

curl -XPUT "localhost:9200/my-index/\_settings" -H 'Content-Type: application/json' -d '{"number\_of\_replicas": 2}'

5. Make the cluster read-write:

text

curl -XPUT "localhost:9200/\_all/\_settings" -H 'Content-Type: application/json' -d '{"index.blocks.read\_only\_allow\_delete": null}'

6. Apply a persistent cluster setting (example: max shards):

text

curl -XPUT "localhost:9200/\_cluster/settings" -H 'Content-Type: application/json' -d '{

"persistent": {

"cluster.max\_shards\_per\_node": 1000

}

}'

7. Change Elasticsearch JVM heap size:

* Edit jvm.options file (commonly at /etc/elasticsearch/jvm.options):  
  Change lines to:
* text

-Xms8g

-Xmx8g

* Restart Elasticsearch and check heap with:
* text

curl -XGET "localhost:9200/\_nodes/jvm?pretty"

8. Add a new node to the cluster:

* Install Elasticsearch, set the same cluster.name in elasticsearch.yml,  
  set discovery.seed\_hosts and cluster.initial\_master\_nodes (for new clusters).
* Start the node and verify with \_cat/nodes.

9. Set minimum master-eligible nodes:

text

discovery.zen.minimum\_master\_nodes: 2

(For older versions; new versions use voting configuration.)

10. Enable encryption in transit:

* Set these in elasticsearch.yml:
* text

xpack.security.transport.ssl.enabled: true

xpack.security.transport.ssl.keystore.path: certs/your\_keystore.p12

xpack.security.transport.ssl.truststore.path: certs/your\_truststore.p12

11. Create a user with read privileges:

text

curl -XPOST "localhost:9200/\_security/user/myreader" -H "Content-Type: application/json" -d '{

"password" : "mypassword",

"roles" : [ "read\_role" ],

"full\_name" : "Read Only User"

}'

Create the role beforehand using the role API.

12. Delete all documents but keep index:

text

curl -XPOST "localhost:9200/my-index/\_delete\_by\_query" -H 'Content-Type: application/json' -d '{"query": {"match\_all": {}}}'

13. Cross-cluster search configuration:

* Add cluster.remote.<remote-cluster-name>.seeds: [ "otherhost:9300" ] to elasticsearch.yml on querying cluster.

14. Refresh an index:

text

curl -XPOST "localhost:9200/my-index/\_refresh"

15. Trigger a snapshot backup:

text

curl -XPUT "localhost:9200/\_snapshot/my\_backup/snapshot\_1?wait\_for\_completion=true" -H 'Content-Type: application/json' -d '{

"indices": "index1,index2"

}'

16. Install an Elasticsearch plugin:

text

sudo bin/elasticsearch-plugin install ingest-geoip

Restart required.

17. Retrieve shard allocation for an index:

text

curl -XGET "localhost:9200/\_cat/shards/my-index?v"

18. Tune index refresh interval:

* API:
* text

curl -XPUT "localhost:9200/my-index/\_settings" -H "Content-Type: application/json" -d '{"index.refresh\_interval": "60s"}'

* Or set in index template/config.

19. Temporarily disable shard allocation:

text

curl -XPUT "localhost:9200/\_cluster/settings" -H "Content-Type: application/json" -d '{

"transient": {

"cluster.routing.allocation.enable": "none"

}

}'

20. Change log level dynamically:

text

curl -XPUT "localhost:9200/\_cluster/settings" -H 'Content-Type: application/json' -d '{

"transient": {"logger.org.elasticsearch.transport": "DEBUG"}

}'

## **Kibana**

21. Start with custom config path:

text

bin/kibana --config /etc/kibana/custom-kibana.yml

22. Sample SSL section in kibana.yml:

text

server.ssl.enabled: true

server.ssl.certificate: /path/to/cert.crt

server.ssl.key: /path/to/cert.key

23. Configure Kibana for remote Elasticsearch:

text

elasticsearch.hosts: ["https://remote-eshost:9200"]

24. Enable RBAC for dashboards:

* Set xpack.security.enabled: true in both Elasticsearch and Kibana.
* Use Kibana Spaces, grant roles via UI or the API.

25. Configure Spaces in Kibana:

* In Kibana UI, go to "Stack Management" → "Spaces" → "Create Space", assign permissions.

26. Generate/update Kibana encryption keys:

text

bin/kibana-encryption-keys generate

Add generated keys to kibana.yml under xpack.encryptedSavedObjects.encryptionKey and related settings.

27. Configure Kibana to listen on all interfaces:

text

server.host: "0.0.0.0"

28. Enable Kibana reporting:

text

xpack.reporting.enabled: true

And ensure chromium dependencies are installed.

29. Set logging to debug:

text

logging.root.level: debug

30. Provision external alert connector:

* In "Stack Management" → "Connectors", "Create Connector" and enter the config (e.g., Slack webhook). Test using "Test connector" in UI.

## **Logstash**

31. Test a pipeline config without running it:

text

bin/logstash -t -f /etc/logstash/conf.d/pipeline.conf

32. Enable config reload on change:  
Set in logstash.yml:

text

config.reload.automatic: true

33. Set up a persistent queue:

text

queue.type: persisted

Add to logstash.yml

34. List installed plugins:

text

bin/logstash-plugin list --verbose

35. Add to pipelines.yml:

text

- pipeline.id: pipeline1

path.config: "/etc/logstash/conf.d/pipeline1.conf"

- pipeline.id: pipeline2

path.config: "/etc/logstash/conf.d/pipeline2.conf"

36. Run Logstash manually with a config file:

text

bin/logstash -f /tmp/test.conf --config.test\_and\_exit

37. Use environment variables for secrets:

text

password => "${LOGSTASH\_MY\_PASSWORD:defaultpassword}"

38. Output to Elasticsearch and file:

text

output {

elasticsearch { hosts => ["localhost:9200"] index => "myindex" }

file { path => "/tmp/output.log" }

}

39. Install a new filter plugin:

text

bin/logstash-plugin install logstash-filter-translate

40. Tail Logstash logs:

text

tail -f /var/log/logstash/logstash-plain.log

## **Advanced Administration & Troubleshooting**

41. Relocate shards, unblock writes:

* Move shards:
* text

curl -XPOST 'localhost:9200/\_cluster/reroute' -H 'Content-Type: application/json' -d '{ "commands": [ { "move": { "index": "my-index", "shard": 0, "from\_node": "node1", "to\_node": "node2" } } ] }'

* Unblock writes:
* text

curl -XPUT "localhost:9200/my-index/\_settings" -H 'Content-Type: application/json' -d '{"index.blocks.read\_only\_allow\_delete": null}'

42. Multi-index search in Kibana Dev Tools:

json

GET index1,index2/\_search

{

"query": { "match\_all": {} }

}

43. Rotate Elasticsearch access credentials:

* Create new user/key with POST \_security/user, update consuming configs, then delete old.

44. Conditionals by hostname in Logstash:

text

filter {

if [host] == "web1" {

mutate { add\_field => { "source\_group" => "frontend" } }

} else if [host] == "db1" {

mutate { add\_field => { "source\_group" => "backend" } }

}

}

45. Apply/revert transient cluster setting:

* Apply:
* text

curl -XPUT "localhost:9200/\_cluster/settings" -H 'Content-Type: application/json' -d '{ "transient": { "indices.recovery.max\_bytes\_per\_sec": "40mb" } }'

* Revert:
* text

curl -XPUT "localhost:9200/\_cluster/settings" -H 'Content-Type: application/json' -d '{ "transient": { "indices.recovery.max\_bytes\_per\_sec": null } }'

46. Enable Elasticsearch audit logging:

* In elasticsearch.yml:
* text

xpack.security.audit.enabled: true

xpack.security.audit.outputs: [ index, logfile ]

47. Scripted ILM policy for deletion:

json

PUT \_ilm/policy/delete-after-7d

{

"policy": {

"phases": {

"hot": { "actions": {} },

"delete": { "min\_age": "7d", "actions": { "delete": {} } }

}

}

}

48. Configure API key authentication for log shippers:

* Create an API key:
* text

curl -XPOST "localhost:9200/\_security/api\_key" -H 'Content-Type: application/json' -d '{"name":"filebeat\_key","role\_descriptors":{...}}'

* Set in the shipper config:
* text

output.elasticsearch:

api\_key: "<id>:<api\_key>"

49. Automate Kibana dashboard export for backups:

text

curl -XPOST "localhost:5601/api/saved\_objects/\_export" -H "kbn-xsrf: true" -H "Content-Type: application/json" -d '{"type":["dashboard"]}' > backup\_dashboards.ndjson

50. End-to-end: new field pipeline

* Logstash:  
  Add to filter:
* text

mutate { add\_field => { "environment" => "prod" } }

* Elasticsearch:  
  Ensure mapping for "environment": {"type": "keyword"}.
* Kibana:  
  Refresh field list, add to index pattern, and use it in a dashboard visualization