- Elasticsearch Data Modeling
- Elasticsearch Data Processing
- Elasticsearch from Dev to Production
- Elasticsearch Cluster Deployment
- Elasticsearch Nodes and Index Management
- Elasticsearch Advanced Tips and Tricks

Module 3

Elasticsearch from Dev to Production

elastic

Topics

- Securing Elasticsearch
- Development vs. Production Mode
- Scaling Elasticsearch
- Server Configuration Best Practices

Elasticsearch From Dev to Production

Lesson 1

Securing Elasticsearch

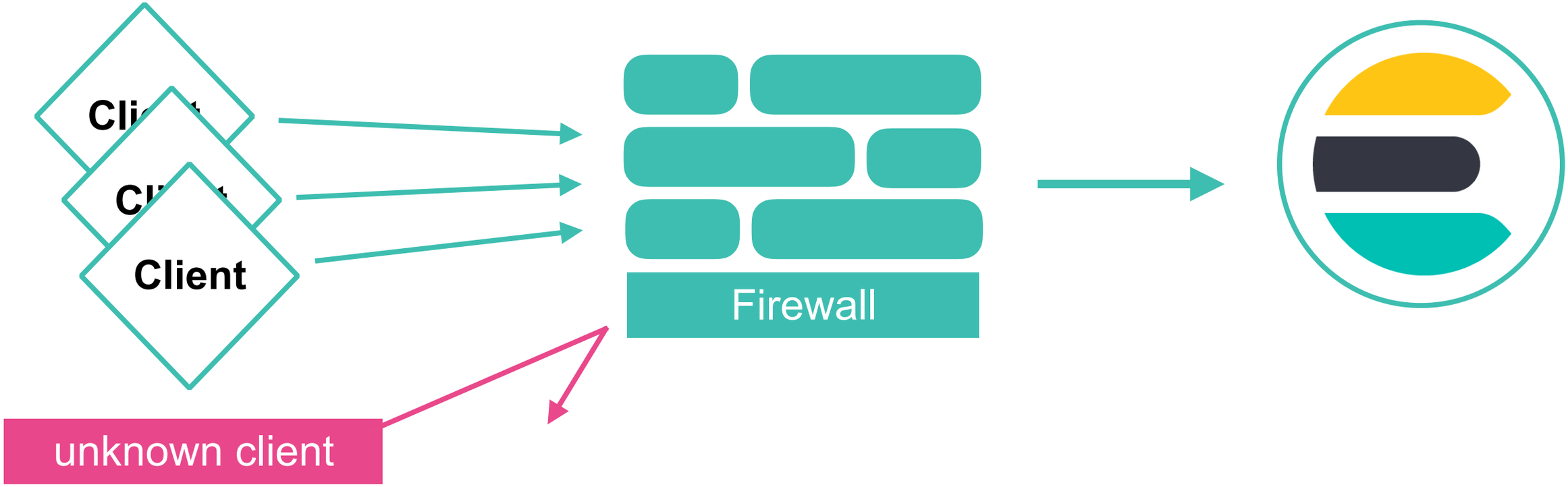
elastic

Security Considerations

- Make sure to secure your cluster before going live:
  - firewalls
  - reverse proxy
  - *Elastic Security*

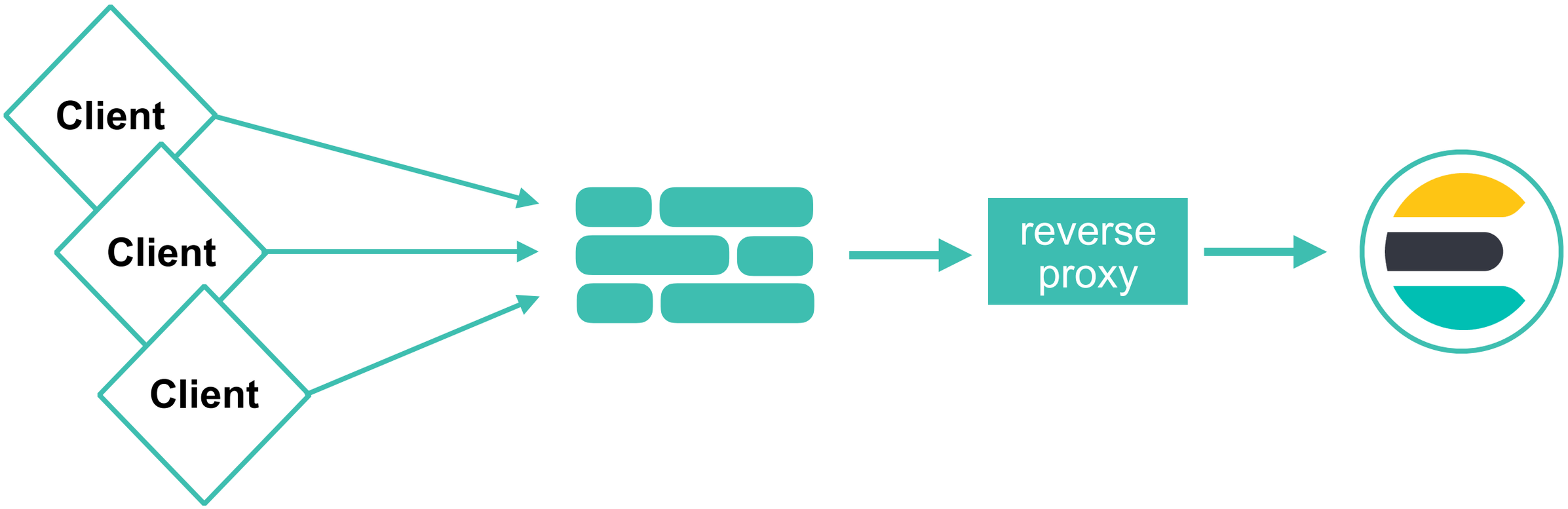
Firewall

- Add ***firewall rules*** with IP filtering to only allow access from the machines running your application
  - **never expose port 9200 http publicly on the internet**
  - restrict access of ports to only systems necessary
  - be careful, don't block port 9300 between the nodes



Reverse Proxy

- You can put Elasticsearch behind a ***reverse proxy***
  - gain basic authentication and authorization with OSS version
  - Elastic security (Basic license) is recommended instead



Elastic Security

- The Elasticsearch Basic subscription (free) includes **core security features** such as:
  - encrypted communications
  - role-based access control
  - and more...
- It enables the community to encrypt network traffic, create users and define roles to protect their data
  - https://www.elastic.co/blog/security-for-elasticsearch-is-now-free

Configuring Elastic Security

- Enable Elastic security feature on every node (**elasticsearch.yml**)

```
xpack.security.enabled: true
```

- Configure Transport Layer Security (TLS/SSL) for internode-communication
- If you plan to run Elasticsearch in a Federal Information Processing Standard (FIPS) 140-2 enabled JVM

```
xpack.security.fips_mode.enabled: true
```

- Restart Elasticsearch on each node of your cluster
- Set the passwords for all built-in users

```
./elasticsearch/bin/elasticsearch-setup-passwords interactive
```

Configuring Transport Layer Security

- Generate a certificate and private key for each node in your cluster

```
./elasticsearch/bin/elasticsearch-certutil cert -out config/elastic-certificates.p12
```

- Copy the PKCS#12 keystore to each node in your cluster
- Enable TLS communication on each node in your cluster (**elasticsearch.yml**)

```
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.keystore.path: elastic-certificates.p12
xpack.security.transport.ssl.truststore.path: elastic-certificates.p12
```

Configuring Kibana After Enabling Security

- Configure Kibana to use the appropriate built-in user (**kibana.yml**)

```
elasticsearch.username: "kibana"
elasticsearch.password: "kibanapassword"
```

- You can also use Kibana keystore
  - create a keystore to avoid having passwords in clear text in the configuration file

```
bin/kibana-keystore create
```

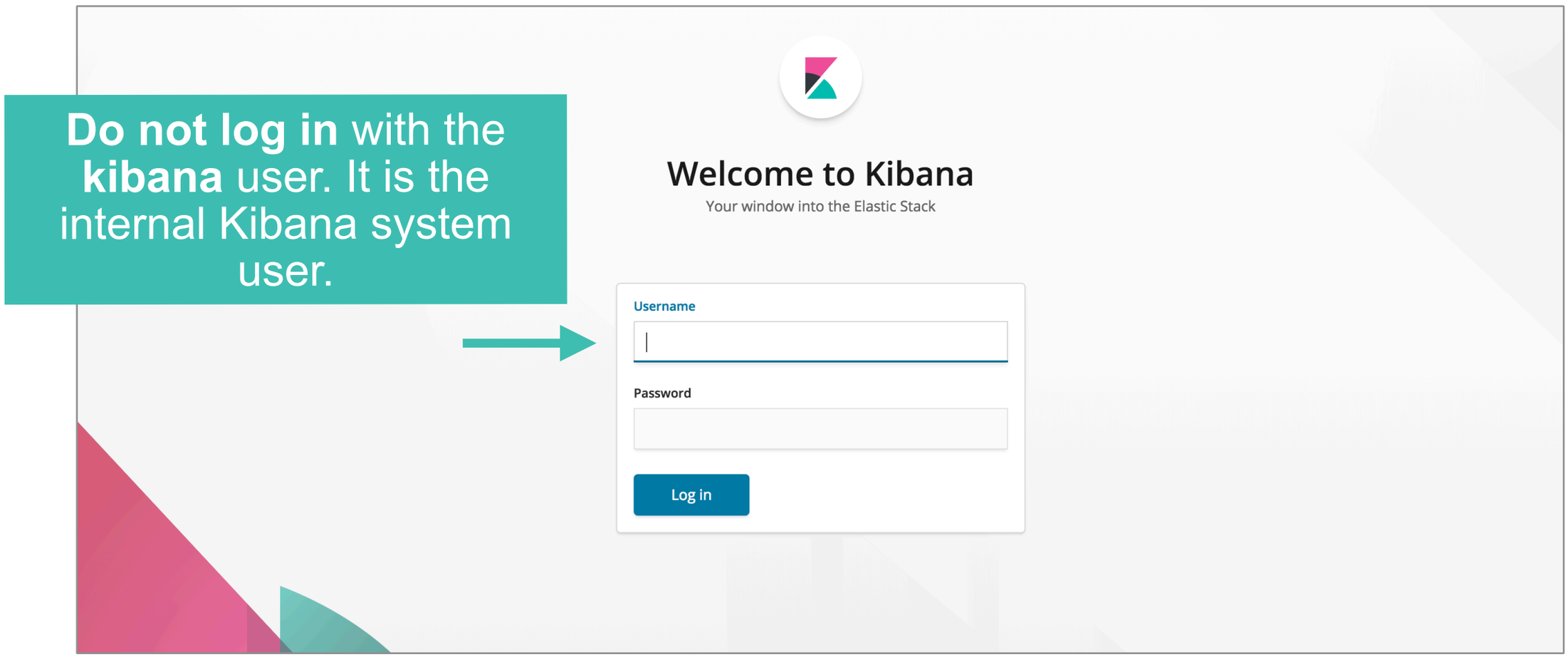
  - add sensitive information to your keystore

```
bin/kibana-keystore add elasticsearch.username
kibana
bin/kibana-keystore add elasticsearch.password
kibanapassword
```

  - those parameters don't need to be defined in **kibana.yml** anymore

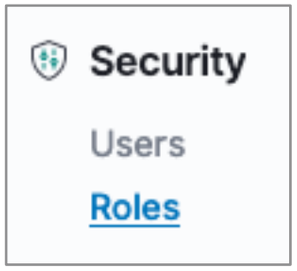
After Configuring Elastic Security

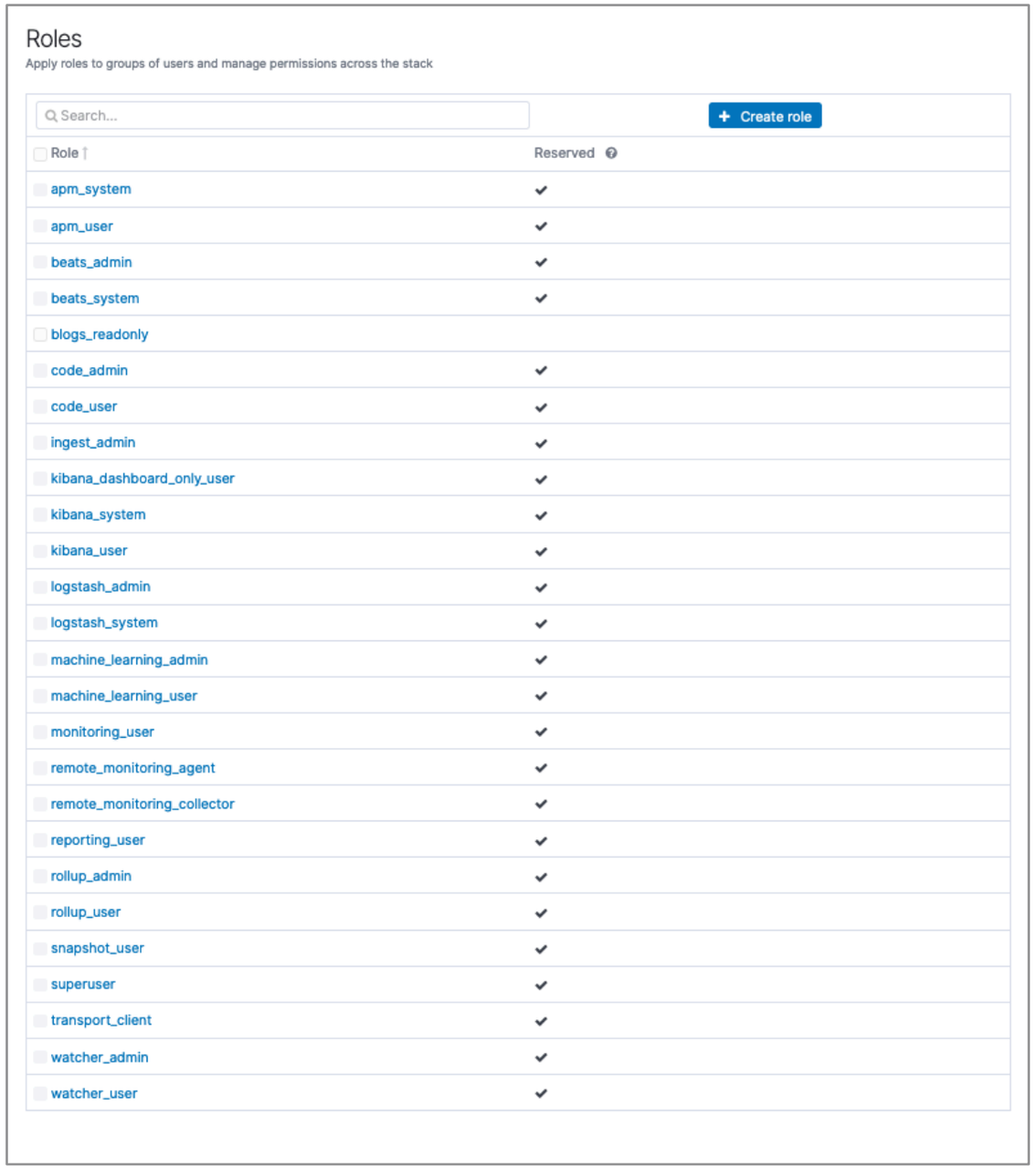
- Login with user and password that you defined before



Defining Users and Roles

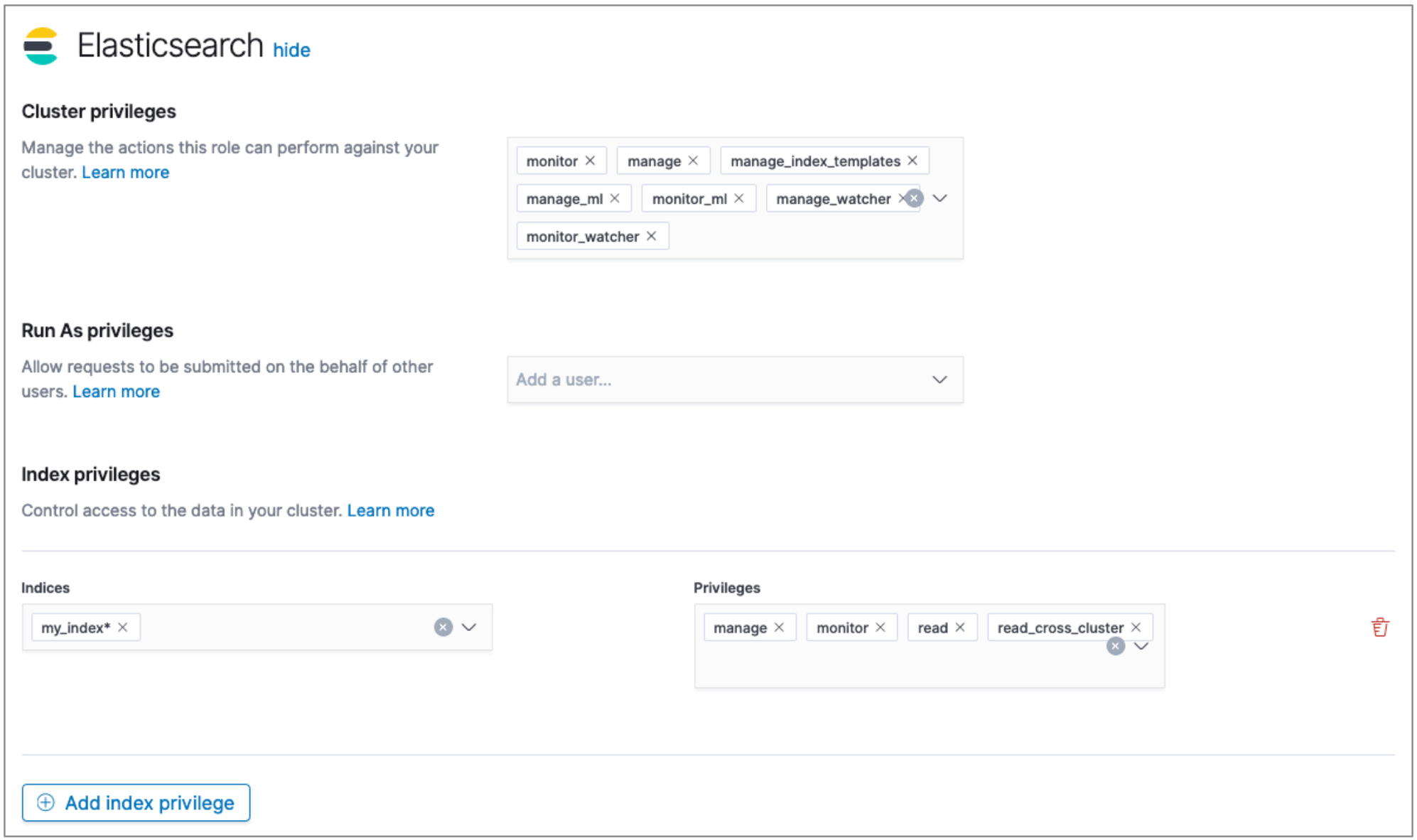
- You can use Kibana to set up roles and users to control access to Elasticsearch





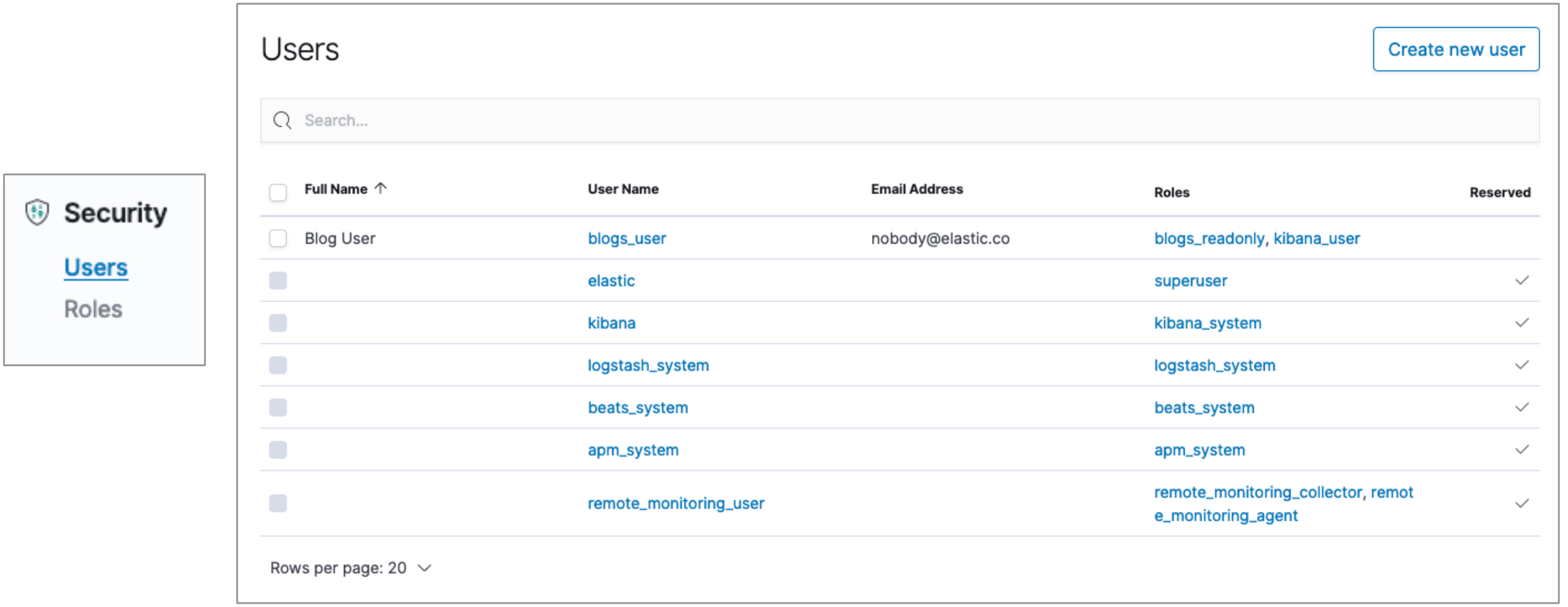
Defining Roles and Privileges

- Roles and privileges can be defined at a very fine-grained level



Defining Roles and Privileges to Users

- Add users with role-based access control



Elasticsearch From Dev to Production

Lesson 1

Review - Securing Elasticsearch

elastic

Summary

- You can secure your cluster with firewalls, reverse proxy, and Elastic Security
- ***Elastic Security*** provides a complete solution for securing Elasticsearch
- If **Elastic Security** is enabled, unless you have a trial license, you must configure SSL/TLS for internode-communication
- **Make sure to secure your cluster!**

Quiz

1. Explain three options to secure your cluster.
2. **True or False.** An Elasticsearch cluster comes with security enabled by default.
3. **True or False.** It is required to enable SSL/TLS communication when Elastic security is enabled on the basic license.

Elasticsearch From Dev to Production

Lesson 1

Lab - Securing Elasticsearch

elastic

Lesson 2

Development vs. Production Mode

elastic

When You Go To Production

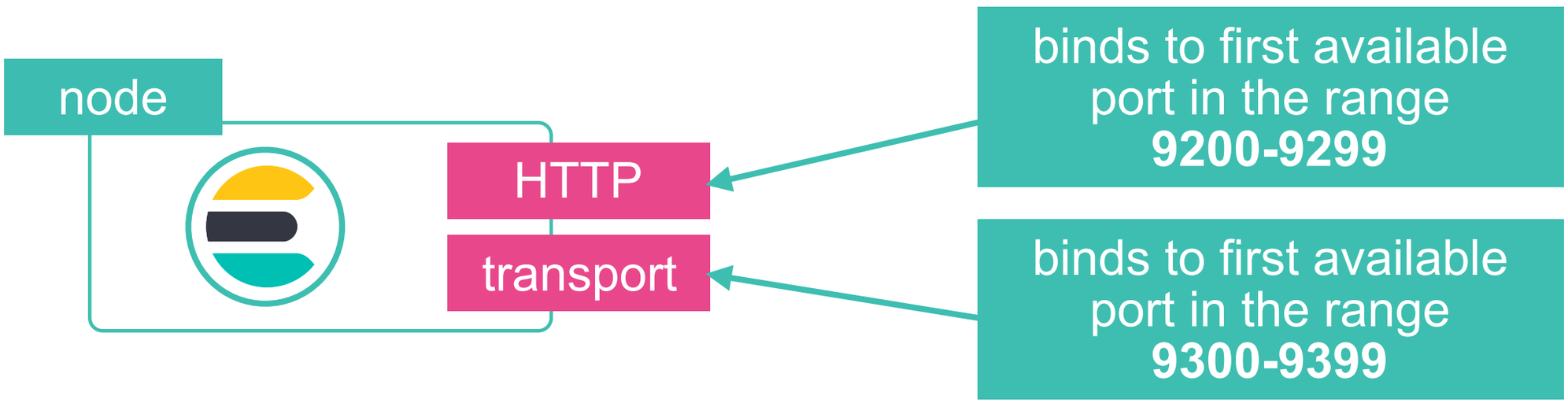
- You might want to care about
  - development vs. production mode
  - caching
  - Linux packages

Development vs. Production Mode

elastic

HTTP vs. Transport

- There are two important network communication mechanisms in Elasticsearch to understand
  - **HTTP**: address and port to bind to for HTTP communication, which is how the Elasticsearch REST APIs are exposed
  - **transport**: used for internal communication between nodes within the cluster
- The defaults are fine for downloading and playing with Elasticsearch, but not useful for production systems
  - bind to **localhost** by default



Development vs. Production Mode

- Every Elasticsearch instance is either in development mode or production mode
  - ***development mode***: if it does ***not*** bind transport to an external interface (the default)
  - ***production mode***: if it does bind transport to an external interface

*"I am just a local cluster used for **development**."*

my_dev_cluster

```
http.port: 9200
http.host: localhost

transport.tcp.port: 9300
transport.bind_host: localhost
```

*"I am running in a **production** environment."*

my_prod_cluster

```
http.port: 9200
http.host: 192.168.1.21

transport.tcp.port: 9300
transport.bind_host: 192.168.1.21
```

Bootstrap Checks

- Elasticsearch has ***bootstrap checks*** upon startup
  - they inspect a variety of Elasticsearch and system settings,
  - and compare them to values that are safe for the operation of Elasticsearch
- Bootstrap checks behave differently depending on the mode
  - **development mode**: any bootstrap checks that fail appear as ***warnings*** in the Elasticsearch log
  - **production mode**: any bootstrap checks that fail will cause Elasticsearch to ***refuse to start***
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/bootstrap-checks.html

Bootstrap Checks

- A node in ***production mode*** must pass all of the checks, or the node will not start
  - the bootstrap checks fit into two categories

JVM Checks
☑ heap size
☑ memory lock
☑ server JVM
☑ not use serial collector
☑ OnError and OnOutOfMemoryError
☑ not use early-access snapshots
☑ not use G1GC collector
☑ disable swapping

Linux Checks
☑ file descriptor
☑ maximum number of threads
☑ max file size
☑ maximum size virtual memory
☑ maximum map count
☑ system call filter
☑ java security permissions
☑ discovery configuration

Caching

elastic

Node Query Cache

- ***Caches search results*** for non-scoring queries
  - uses Least Recently Used (LRU) eviction policy
  - per node, shared by all the shards on that node
- ***Use filters*** whenever possible:

```
GET blogs/_search
{
  "query": {
    "range": {
      "publish_date": {
        "gte": “2016-01-01”,
        "lte": “2018-12-31”
      }
    }
  }
}
```

Cannot be cached

```
GET blogs/_search
{
  "query": {
    "bool": {
      "filter": {
        "range": {
          "publish_date": {
            "gte": 2016-01-01,
            "lte": 2018-12-31
          }
} } } } }
```

Can be cached

Shard Request Cache

- Caches shard-level ***aggregation results, counts*** and ***suggestions*** by default
  - uses Least Recently Used (LRU) eviction policy
  - per node, shared by all the shards on that node
  - uses the JSON request body as the cache key
- Only if the data has not changed and **size** is set to **0**

```
GET logs_server*/_search
{
  "aggs": {
    "logs_by_day": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "day"
      }
    }
  }
}
```

Is **not** cached

```
GET logs_server*/_search
{
  "size": 0,
  "aggs": {
    "logs_by_day": {
      "date_histogram": {
        "field": "@timestamp",
        "interval": "day"
      }
    }
  }
}
```

Is cached

Shard Request Cache

- Can also be used to cache ***hits***
- Use the **request_cache** parameter to force caching:

```
GET blogs/_search?request_cache=true
{
  "query": {
    "match": {
      "content": "filebeat"
    }
  },
  "aggs": {
    "top_categories": {
      "terms": {
        "field": "category.keyword",
        "size": 10
      }
    }
  }
}
```

caches even when size > 0

Linux Packages

elastic

Linux Packages

- The tarball package is good for testing and development
- On a production environment you should take advantage of your package management system
  - automatic dependency verification
  - better directory infrastructure
  - binaries and libraries on standard locations
  - configuration and log files on standard locations
  - easier to update through package management systems
  - easier to start and stop through daemons
  - automatic startup on machine reboot

Configuration Management

- It becomes difficult to manage your Elasticsearch settings as you grow your cluster
  - it is really hard to edit many configuration files by hand without making a mistake
- It is a good idea to use a configuration management tool
  - for example: Puppet, Chef, Ansible
- Configuration management tools help make your cluster consistent by automating the process of config changes
  - when you install Elasticsearch through Linux packages it is a bit easier to automate your configuration process
  - all files are in standard locations and an update should not break your automation process

Elasticsearch From Dev to Production

Lesson 2

Review - Development vs. Production Mode

elastic

Summary

- A node in ***production mode*** must pass a series of checks, or the node will not start
- A **query** in a filter context can be ***cached*** by Elasticsearch to improve performance
- Installing Elasticsearch with Linux packages makes it easier to manage your cluster configurations

Quiz

1. **True or False.** In production mode, if a bootstrap check fails then the node will not start.
2. **True or False.** Elasticsearch caches any query to improve performance.
3. Why would you use Linux packages to install Elasticsearch?

Elasticsearch From Dev to Production

Lesson 2

Lab - Development vs. Production Mode

elastic

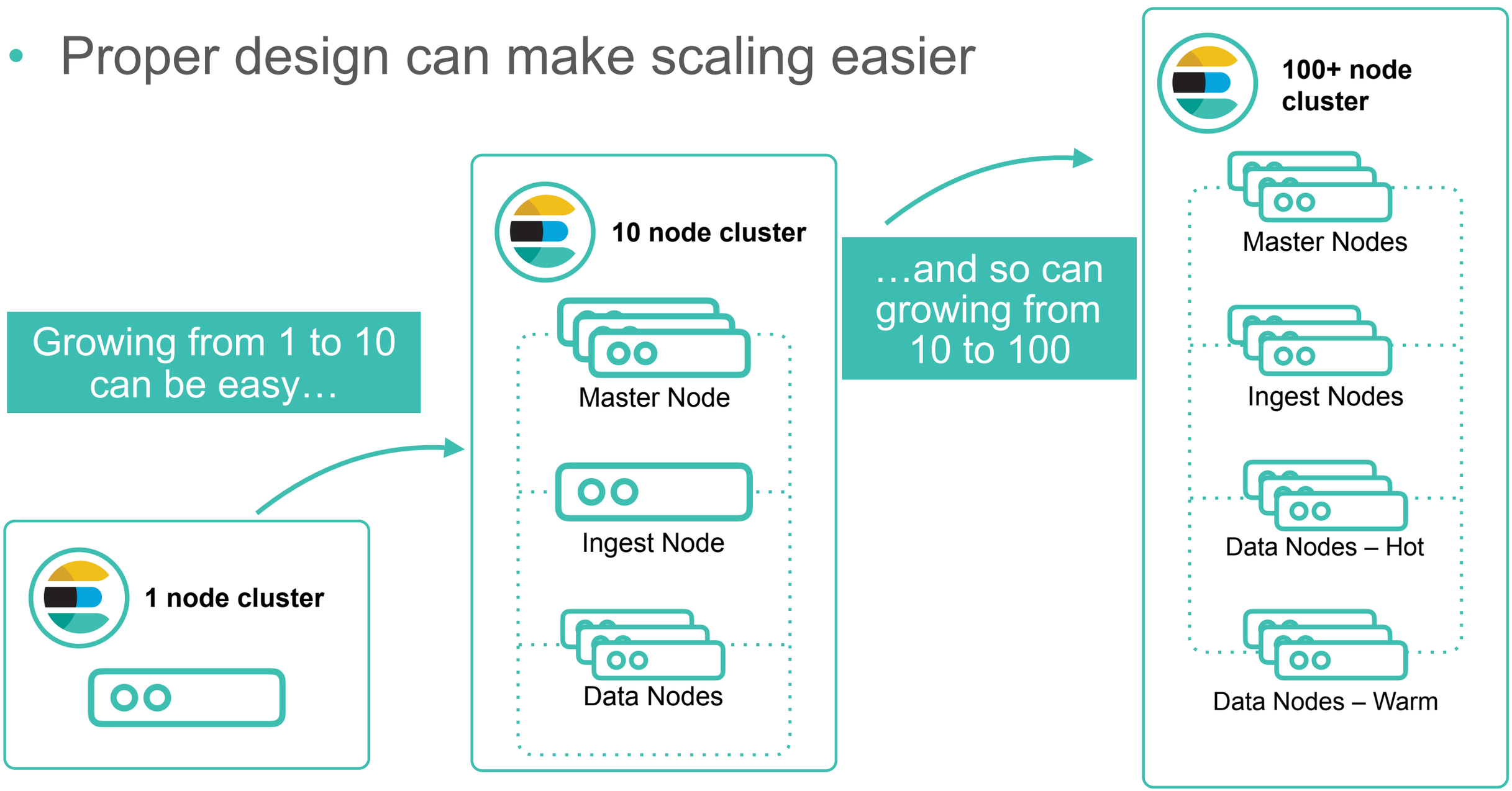
Lesson 3

Scaling Elasticsearch

elastic

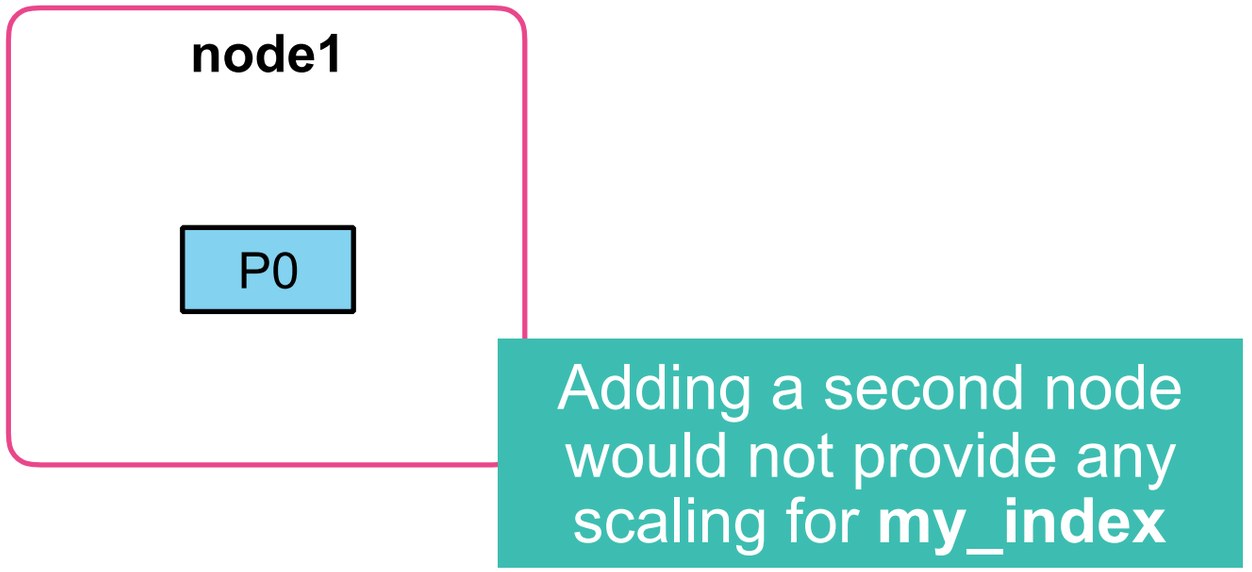
Designing for Scale

- Elasticsearch is built to scale
  - and the default settings can take you a long way
- Proper design can make scaling easier



One Shard…

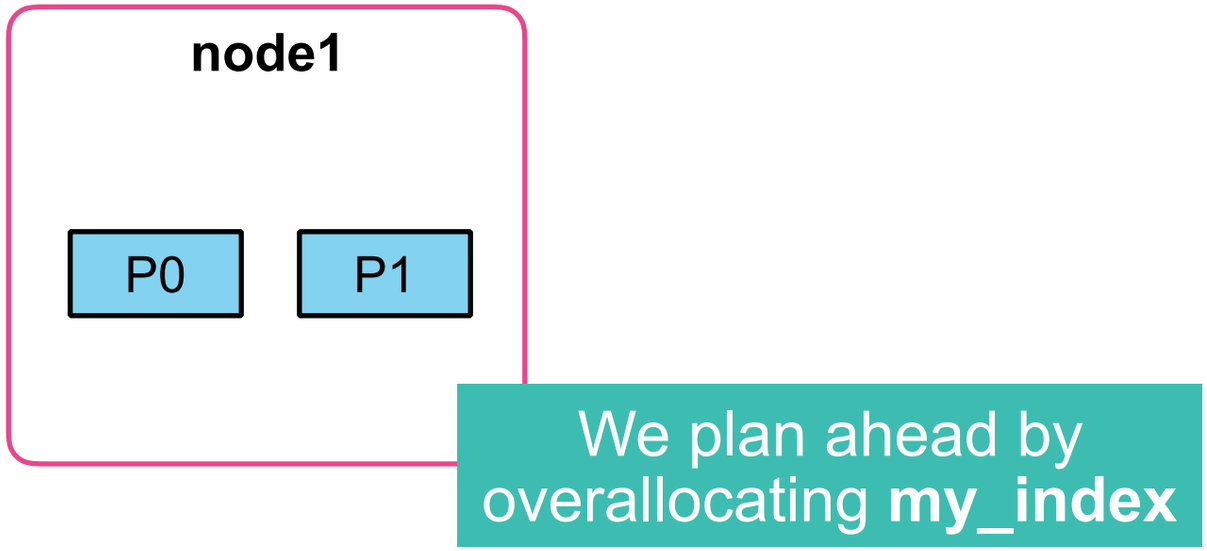
- …does not scale very well:




```
PUT my_index
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0
  }
}
```

Two Shards…

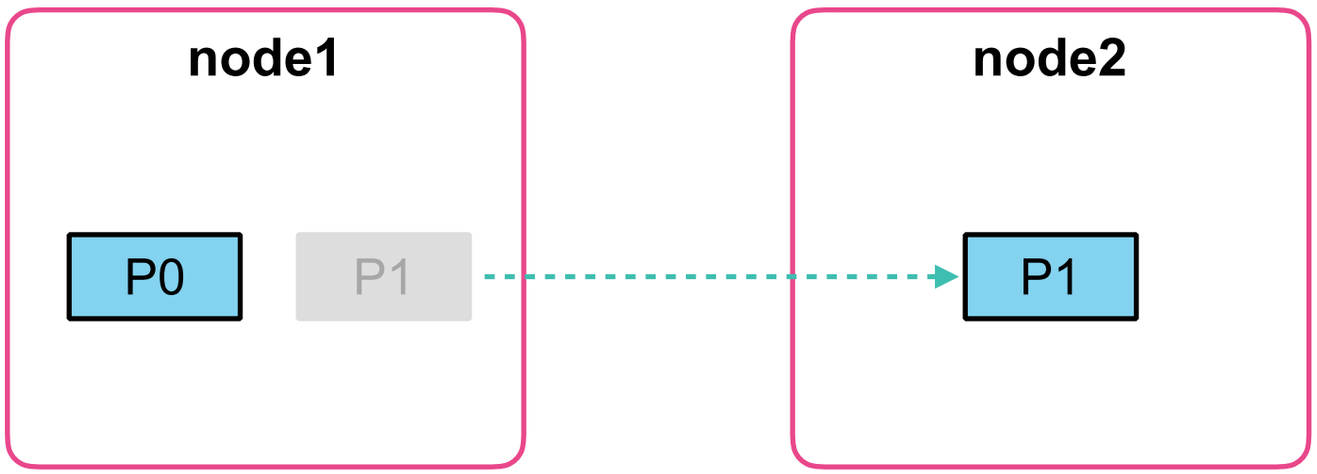
- …can scale if we add a node:




```
PUT my_index
{
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 0
  }
}
```

Balancing of Shards

- Elasticsearch automatically balances the shards:
  - node1 is now responsible for half the amount of data
  - write throughput has doubled (twice the disk IO available)
  - the memory pressure on node1 is less than before
  - when searches are executed, we now use the resources of both node1 and node2

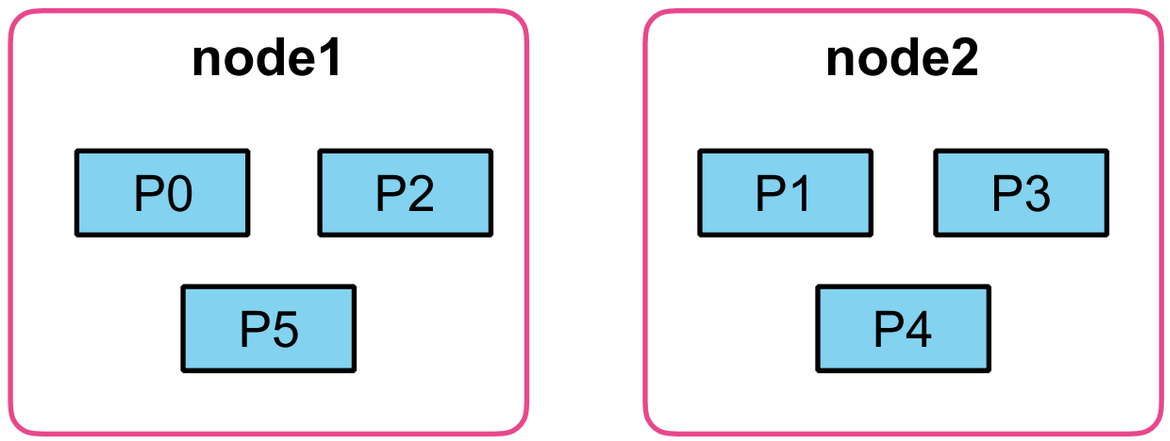


Shard Overallocation

- If you are expecting your cluster to grow, then it is good to plan for that by overallocating shards:
  - ***#shards > #nodes***
- Shards can move between nodes quickly as the cluster grows
  - and there is no downtime during shard relocation

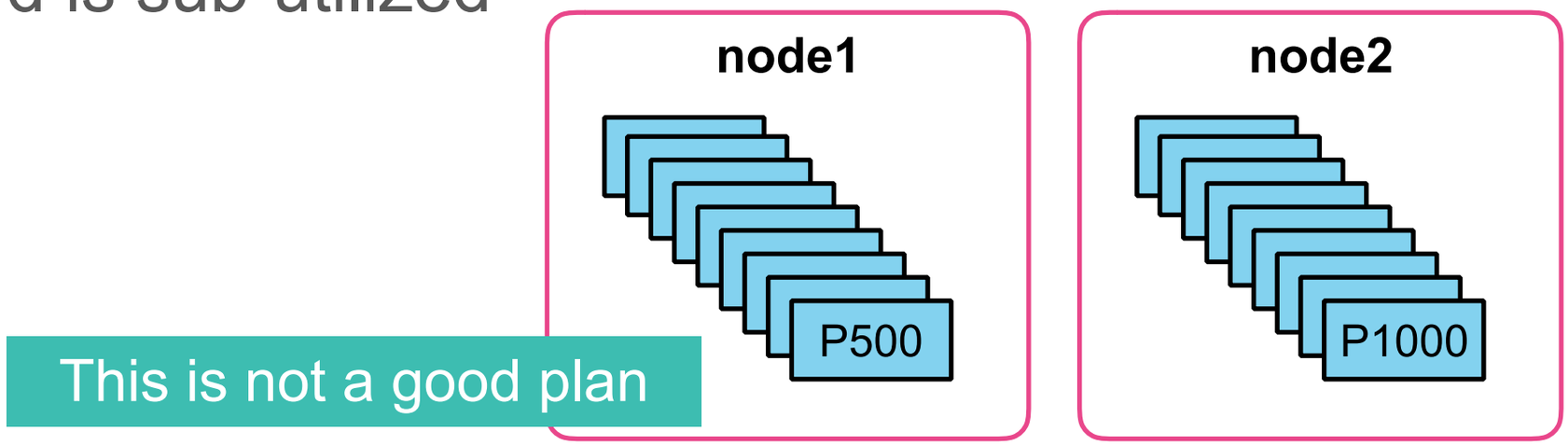
- Note: this strategy works for static, but not time-series data
  - for time-series data, you will have multiple indices with few shards

```
PUT my_index
{
  "settings": {
    "number_of_shards": 6,
    "number_of_replicas": 0
  }
}
```



“Too Much” Overallocation

- A little overallocation is good
- A “kagillion” shards is not:
  - each shard comes at a cost (Lucene indices, file descriptors, memory, CPU)
  - also, a search request needs to hit every shard in the index
- A shard typically can hold at least 10s of gigabytes
  - depends on the use case
  - a 1GB shard is sub-utilized



This is not a good plan

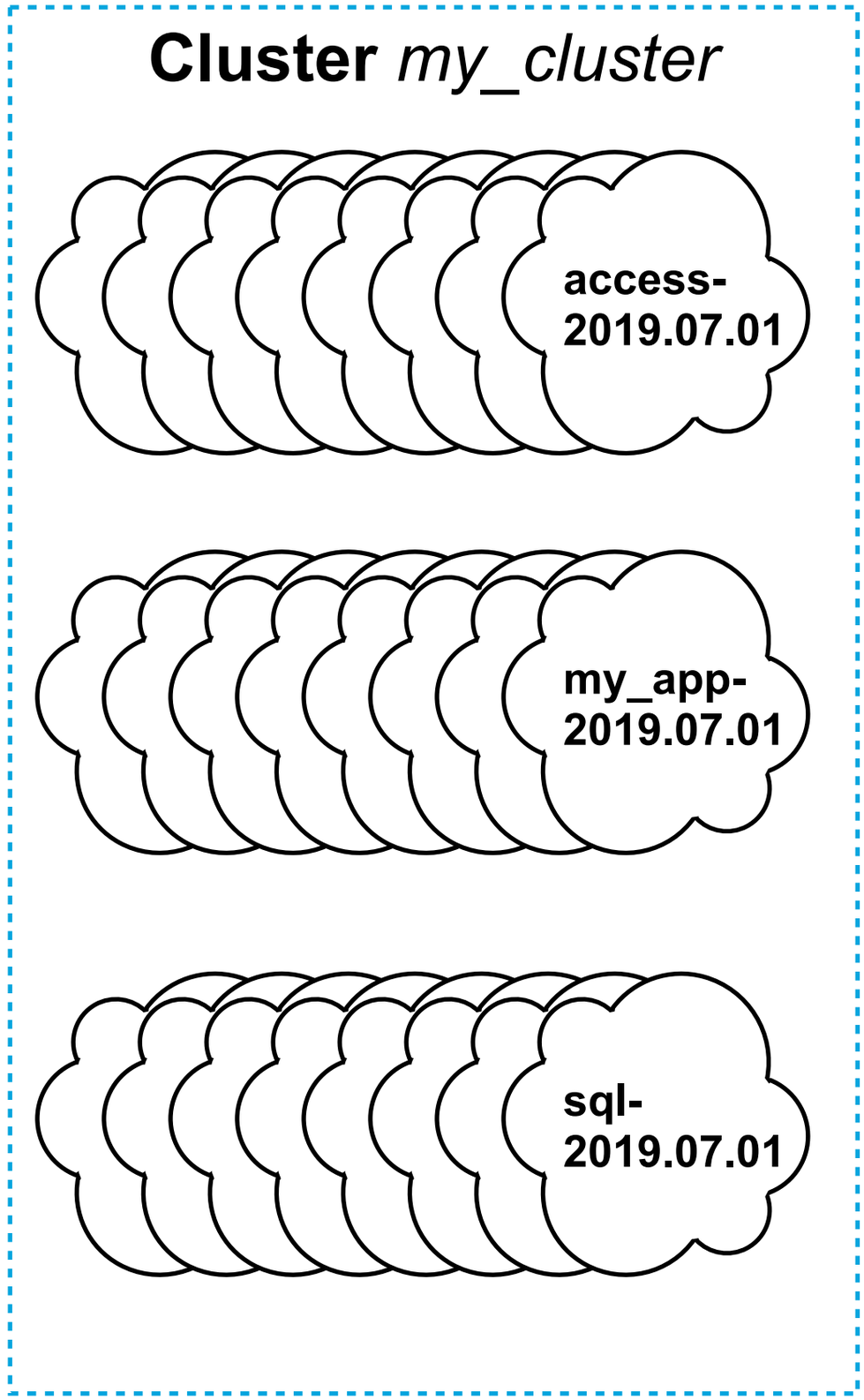
Do not Overshard

- Business Requirements
  - 1GB per day
  - 6 months retention
  - **~180GB**

we could easily have this data in 10 shards

- Common Scenario
  - 3 different logs
  - 1 index per day each
  - 5 shards (default before 7.0)
  - 6 months retention
  - **~2700 shards**

too many shards for no good reason!

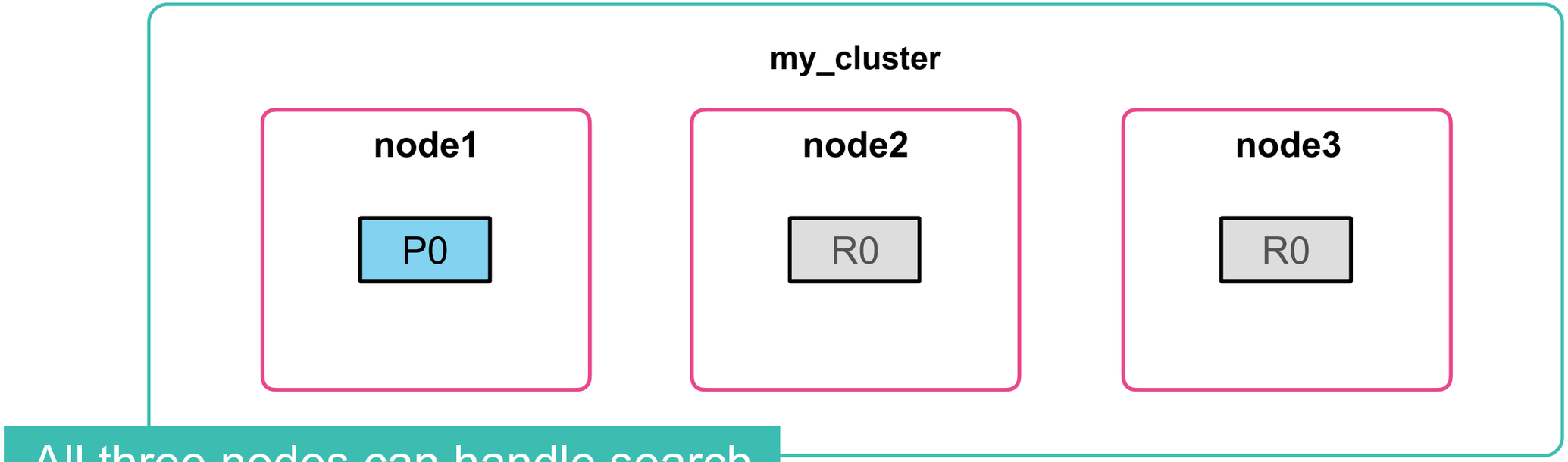


Scaling for Reads

elastic

Scaling for Reads

- Searches and aggregations scale with replicas
- For example, one primary and as many replicas as you have additional nodes
  - use **auto_expand_replicas** setting to change the number of replicas automatically, as you add/remove nodes



All three nodes can handle search requests in parallel, as they each have a full copy of the data

Optimizing for Read Throughput

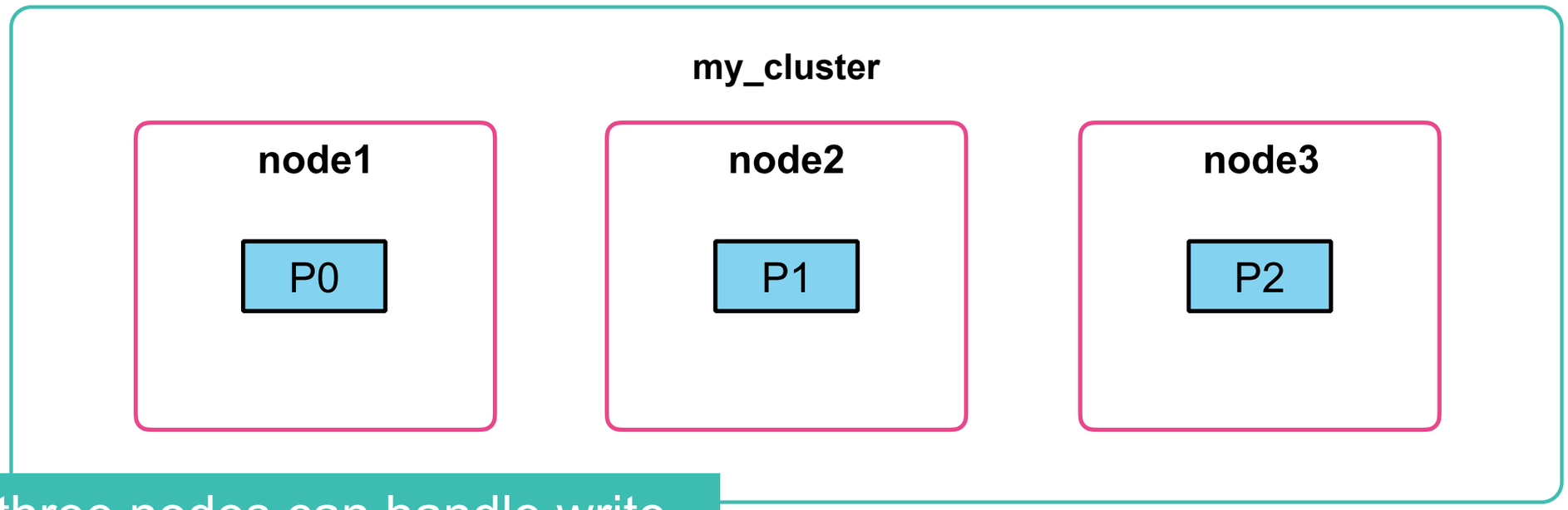
- Avoid **nested types** and the **join** datatype
  - create flat, denormalized documents
- Query the smallest number of fields
  - consider **copy_to** over **multi_match**
- Map identifiers as **keyword** instead of as a number
  - term queries on keyword fields are very fast
- Force-merge read-only indices
- Limit the scope of aggregations (upcoming topic!)
- Use filters, as they are cacheable

Scaling for Writes

elastic

Scaling for Writes

- Write throughput scales by increasing number of primaries
- Configuring more primary shards allows Elasticsearch to "fan out" the writes, so each shard is doing less work
- Maximize throughput by using disks on all machines
- When an index is done with writes, you can shrink it



All three nodes can handle write requests in parallel

Optimizing for Write Throughput

- Use **_bulk** API to minimize the overhead of HTTP requests
- Parallelize your write requests
- Disable refreshing every second
  - set **index.refresh_interval** to **-1** for very large writes (then back to default when finished indexing)
  - set **index.refresh_interval** to **30s** to increase indexing speed but affect search as little as possible (e.g. logs use case)
- Disable replicas, then re-enable after very large writes
  - every document also needs to be written to every replica
- Use auto-generated IDs
  - Elasticsearch won't check whether a doc ID already exists

Lesson 3

Review - Scaling Elasticsearch

elastic

Summary

- If you are expecting your cluster to grow, then it is good to plan for that by ***overallocating*** shards
- A little overallocation is good. A “kagillion” shards is not
- You can scale the read workload of your cluster by adding more nodes and ***increasing the number of replicas*** of your indices
- You can scale the write workload of your cluster by adding more nodes and ***increasing the number of primaries*** of your indices

Quiz

1. If you have a two node cluster, why would you ever create an index with more than two primary shards?
2. **True** or **False**. To maximize read throughput you need to divide your data over as many primaries as possible
3. **True** or **False**. Using auto-generated IDs is more efficient than providing your own IDs

Elasticsearch Advanced Tips and Tricks

Lesson 3

Lab - Scaling Elasticsearch

elastic

Elasticsearch Advanced Tips and Tricks

Lesson 4

Server Configuration Best Practices

elastic

Server Configuration Best Practices

- Hardware Best Practices
- JVM Settings

Hardware

Best Practices

elastic

Networking Best Practices

- Avoid running over WAN links between data centers
  - not officially supported by Elastic
- Try to have zero (or very few) hops between nodes
- If you have multiple network cards
  - separate **transport** and **http** traffic (transport on faster NIC)
    - bind to different network interfaces
    - use separate firewall rules for each kind of traffic
- Use long-lived HTTP connections
  - client libraries support this
  - or use a proxy/load-balancer

Storage Best Practices

- Prefer solid state disks (SSDs)
  - segments are immutable
  - so the *write amplification factor* approaches one and is a non-issue
- Local disk is king!
  - in other words, avoid NFS, SMB, AWS EFS, or Azure filesystem
- Elasticsearch does not need redundant storage
  - replicas = software provided HA
  - local disks are better than **SAN**
  - **RAID1/5/10** is not necessary

Storage Best Practices

- If you have multiple disks in the same server
  - you can set **RAID 0** or **path.data**
- **RAID 0**
  - splits ("stripes") data evenly across two or more disks
  - perfect distribution of the data across the disks
  - However, if you lose one disk, you lose the data on all disks
- **path.data**
  - allows you to distribute your index across multiple disks
  - potential to have an unbalanced distribution (all files belonging to a shard will be stored on the same data path)
  - if you lose one disk, the shards on all other disks are preserved

may generate node level watermark issues if disks have different sizes

Storage Best Practices

- SSDs: Use **noop** or deadline scheduler in the OS
  - for more detail, see: https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html#_disks

```
echo noop > /sys/block/{DEVICE}/queue/scheduler
```

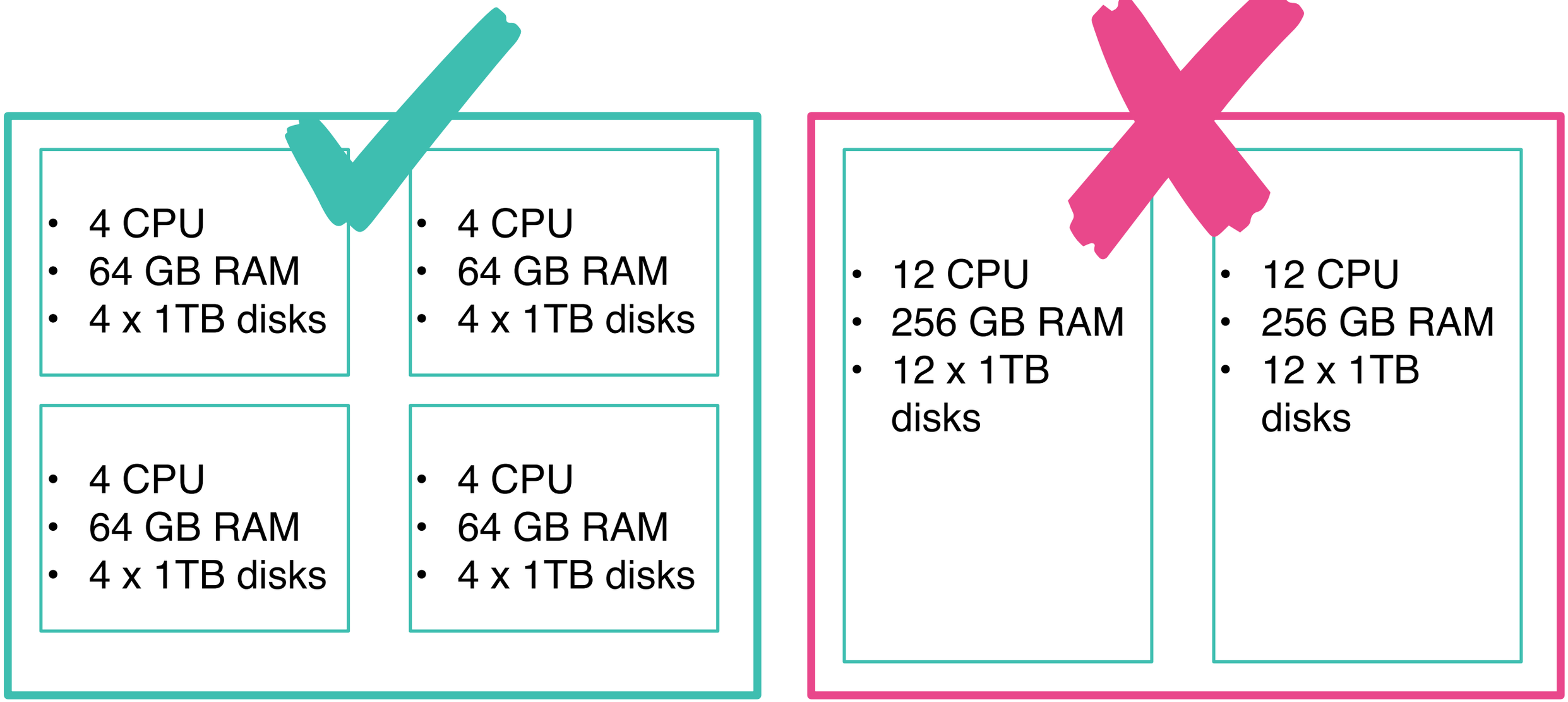
- SSDs: Trim your disks periodically:
  - https://www.elastic.co/blog/is-your-elasticsearch-trimmed
- Spinning disks are OK for ***warm*** nodes
  - but don't forget to disable concurrent merges

```
index.merge.scheduler.max_thread_count: 1
```

Hardware Selection

- Elasticsearch was designed to run on commodity hardware
- In general, choose ***large*** machines *over* ***x-large*** machines
  - loss of a large node has a greater impact
  - ***prefer*** 6 machines with 4-8cpu x 64gb x 4 1tb drives
  - ***avoid*** 2 machines with 12cpu x 256gb x 12 1tb drives
- Avoid running multiple nodes on one server
  - one Elasticsearch instance can fully consume a machine
  - resource contention + multiple JVMs can be problematic
- Larger machines can be helpful as ***warm*** or ***cold*** nodes
  - configure shard allocation filtering

Hardware Selection



Cloud Strategies

- On Cloud, use the ***discovery plugin***
  - because IPs can change frequently, the plugin dynamically configures the discovery seeds
- Span the cluster across more than one AZ
  - use shard awareness and forced awareness
- Prefer ephemeral storage over network storage
  - as noted earlier, local SSD-backed storage is preferred
- Snapshot to cloud storage with the **repository plugins**
- Avoid instances marked with low networking performance
  - network latency will kill performance

Throttles

- Elasticsearch has relocation and recovery throttles
  - it ensures that these tasks do not have a negative impact
  - you can speed up the process by throttling up
- **Recovery**
  - for faster recovery, temporarily increase the number of concurrent recoveries

```
PUT _cluster/settings
{
  "transient": {
    "cluster.routing.allocation.node_concurrent_recoveries": 2
  }
}
```

- **Relocation**
  - for faster rebalancing of shards, increase

```
"cluster.routing.allocation.cluster_concurrent_rebalance" : 2
```

JVM Settings

elastic

JVM Configuration

- Since Elasticsearch 6.0, only 64-bit JVMs are supported
- You can configure the Java Virtual Machine (JVM) in two ways:
  - the **config/jvm.options** file (preferred)

```
-Xms30g
-Xmx30g
```

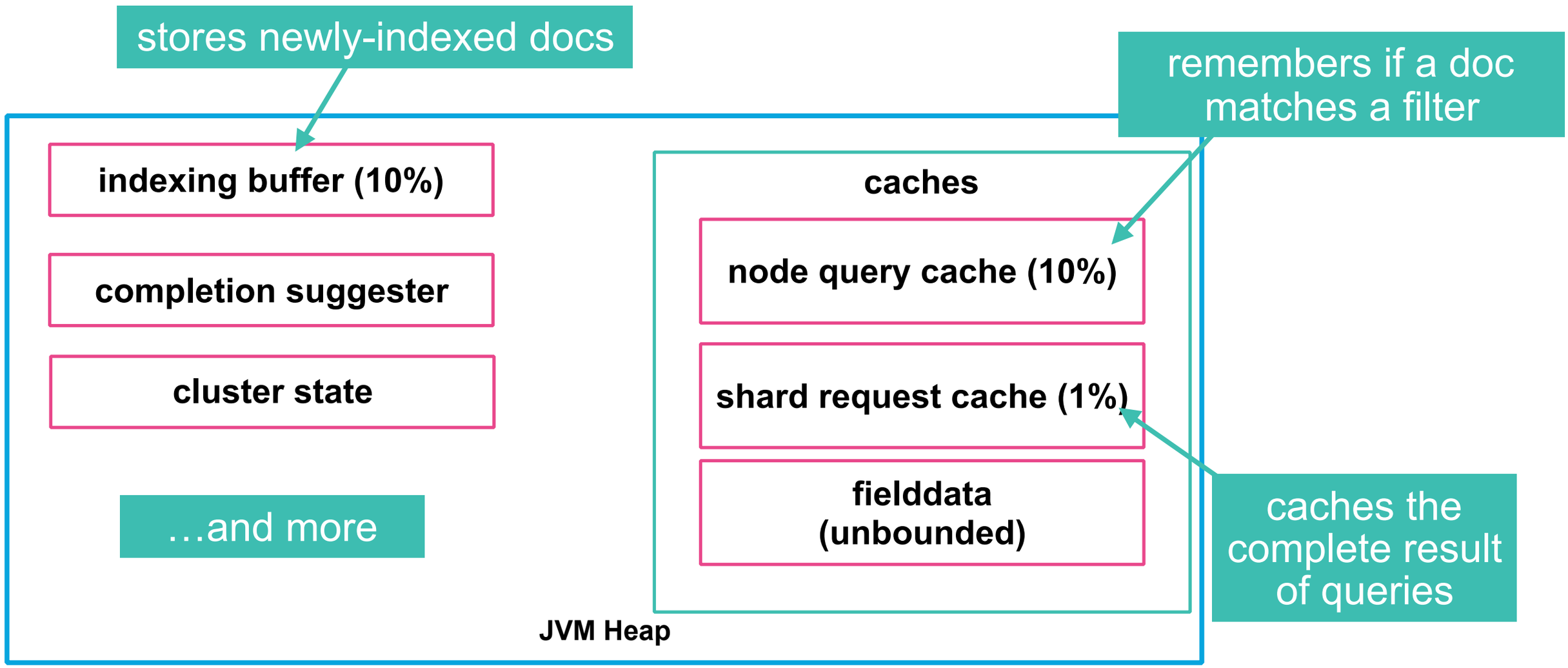
  - setting the **ES_JAVA_OPTS** environment variable

```
ES_JAVA_OPTS="-Xms30g -Xmx30g" bin/elasticsearch
```

- Elasticsearch has very good JVM defaults
  - avoid and be careful when changing them

What Goes on the Heap?

- Some of the major usage of the heap by Elasticsearch includes:



JVM Heap Size

- By default, the JVM heap size is 1 GB
  - likely not high enough for production
  - you can change it using **Xms** (min heap) and **Xmx** (max heap)

```
ES_JAVA_OPTS="-Xms8g -Xmx8g" ./bin/elasticsearch
```

- Some guidelines for configuring the heap size
  - set **Xms** and **Xmx** to the same size (bootstrap check)
  - set **Xmx** to no more than 50% of your physical RAM
  - do not exceed more than 30GB of memory (to not exceed the compressed ordinary object pointers limit)
- Leave as much memory to the filesystem cache as possible
- https://www.elastic.co/blog/a-heap-of-trouble

Production JVM Settings

- JDKs have two modes of a JVM: client and server
  - server JVM is required in production mode
- Configure the JVM to disable swapping
  - by requesting the JVM to lock the heap in memory through mlockall (Unix) or virtual lock (Windows)

Lesson 4

Review - Server Configuration Best Practices

elastic

Summary

- For best performance, choose ***SSD over spinning disks***
- Local disks are preferred - the software provides HA
- In general, choose ***large*** machines over ***x-large*** machines
- Elasticsearch has very good JVM defaults, so avoid changing them or be careful
- When configuring the JVM heap, leave as much memory to the filesystem cache as possible

Quiz

1. **True or False.** SAN storage is preferred over local disks to provide high availability of data.
2. **True or False.** It is a good idea to separate the **transport** and **HTTP** traffic over different network interfaces.
3. **True or False.** It is a good idea to change the JVM defaults to accommodate your needs.