ChatGPT

Ubuntu 22.04 Üzerinde Tek Sunucu Agentless SIEM Yığını Kurulum ve İşletim Kılavuzu

Giriş

Bu teknik eğitim kılavuzu, **Ubuntu 22.04** üzerinde tek bir sunucuda çalışan, **agentless (Elastic Agent/ Fleet kullanılmayan)** bir **Elastic SIEM** yığınının tam kurulumunu ve işletimini adım adım açıklamaktadır. Hedef kitle, Elastic Stack ve SIEM konularında deneyimli mühendisler ve SRE'lerdir. Bu nedenle kılavuz, detaylı yapılandırma örnekleri, tasarım kararlarının gerekçeleri, olası hata senaryoları ve çözümleriyle derinlemesine bir başvuru niteliğindedir. Elastic Stack'in Elasticsearch, Kibana ve Logstash bileşenlerini güvenli şekilde kuracak; TLS ile şifrelenmiş iletişim, X-Pack güvenlik özellikleri (kimlik doğrulama, yetkilendirme), SIEM kural motoru ve tespit (detection) yetenekleri gibi konuları ele alacağız. Ayrıca **Elastic resmi dokümantasyonu ile birebir uyumlu** bilgiler sunulacak ve ilgili bölümlerde kaynak bağlantıları ile referanslar belirtilecektir.

Kılavuz boyunca tüm örnekler ve konfigurasyonlar `elk.local` ana makine adına ve `127.0.0.1` IP adresine göre verilmiştir. Bu sayede laboratuvar ortamında veya tek bir sunucuda test yaparken doğrudan örnekleri uygulayabilirsiniz. Tüm şifreli iletişim için kendi oluşturacağımız bir sertifika otoritesine (CA) ait sertifikaları kullanacağız. Görseller yerine, anlaşılması güç akışları açıklamak için ASCII diyagramlar kullanılmıştır. Her bölüm, yapılan adımların **"Neden"** yapıldığını açıklayan notlar, **uygulama komutları**, **doğrulama yöntemleri** ve gerekirse **geri alma/geri dönme** adımlarıyla desteklenmiştir. Kılavuzun sonunda, öğrendiklerinizi pekiştirmeniz için laboratuvar egzersizleri de sunulmuştur.

Önemli Not: Elastic Stack 8.x sürümü ile birlikte güvenlik (X-Pack Security) özellikleri kurulumdan itibaren etkin gelmektedir. İlk kurulum sırasında Elasticsearch otomatik olarak TLS sertifikaları oluşturur, güvenlik ayarlarını uygulayarak **Elastic kullanıcı şifresi** belirler ve Kibana'nın bağlanması için **enrollment token (kayıt belirteci)** üretir [1] [2]. Bu kılavuzda, hem bu otomatik adımları anlamanızı sağlayacak açıklamalar yapılacak, hem de öğrenme amacıyla her adımın manuel olarak nasıl yapılacağı gösterilecektir. Böylece otomasyon arka planında neler olduğunu kavrayabilecek ve gerektiğinde elle müdahale edebileceksiniz.

Mimari Genel Bakış ve Bileşenler

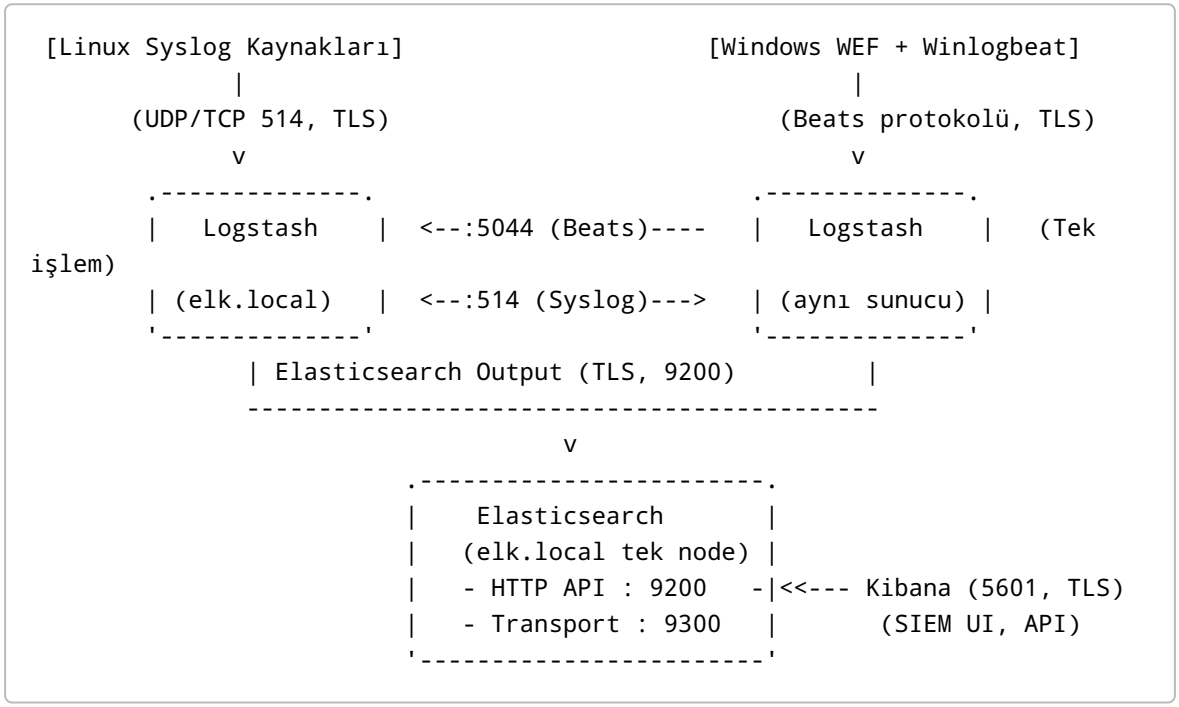
Tek sunucu SIEM mimarisinde tüm temel Elastic bileşenleri aynı makinede çalışır. Bu bileşenler ve görevleri şöyledir:

- **Elasticsearch:** Log verilerini depolayan arama ve analiz motoru. SIEM tarafından üretilen alarm verilerini de indeksler.
- **Kibana:** Elasticsearch'ten verileri çekerek görselleştirme, SIEM arayüzü ve **Detection Engine** (tespit motoru) gibi uygulamaları sunan web arayüzü. Kullanıcılar tarayıcı üzerinden Kibana'ya erişerek arama yapar, panoları görüntüler, alarmları yönetir.
- **Logstash:** Harici kaynaklardan (ör. sunucu logları, network cihazlarından syslog, Windows etkinlikleri vb.) gelen logları alıp filtreleyerek Elasticsearch'e ileten ardışık düzen motoru.

Agentless mimaride Logstash genellikle ağ üzerinden logları dinler (ör. Syslog, Beats protokolü vb.) ve ön işleme yapar.

- **Beats (Opsiyonel):** Agentless kavramı Elastic Agent/Fleet kullanılmamasını ifade eder. Ancak **Filebeat, Winlogbeat** gibi hafif Beats ajanları isteğe bağlı olarak kullanılabilir. Bu kılavuzda, Beats ajanları doğrudan Logstash'e gönderim yapacak şekilde ele alınacaktır (Fleet yönetimi olmadan).

Aşağıdaki ASCII diyagramı, tek sunucu SIEM yığınının temel veri akışını ve bileşenlerin iletişimini göstermektedir:



Yukarıdaki diyagramda, **Linux sunucular** ve **ağ cihazları** loglarını *syslog protokolü* ile Logstash'e gönderir. **Windows sunucuları** ise *Windows Event Forwarding (WEF)* mekanizmasıyla olayları merkezi bir Windows kolektörüne iletir; kolektör üzerindeki **Winlogbeat** ajanı bu logları Logstash'e aktarır. Logstash, aldığı verileri önceden tanımlanmış işlemlerden (pipeline) geçirerek **TLS üzerinden** Elasticsearch'ün 9200 numaralı HTTP arayüzüne iletir. **Kibana**, 5601 portunda çalışır ve hem kullanıcıların web tarayıcısı ile HTTPS iletişimini sağlar, hem de arka planda Elasticsearch'ün API'larına istek göndererek veri çeker veya SIEM dedeksiyon kurallarını çalıştırır.

Bu mimaride güvenlik kritik önemdedir. Tek sunucu olsa bile, iletişim kanalları TLS ile şifrelenmeli, bileşenler arası kimlik doğrulama etkin olmalıdır [3] [4]. Bu nedenle Elasticsearch'ün hem **transport (9300)** hem de **HTTP (9200)** katmanlarında TLS aktif olacak; Kibana ve Logstash da Elasticsearch'e güvenli şekilde bağlanacak şekilde yapılandırılacaktır. Aynı şekilde Kibana'nın web arayüzü de TLS ile sunulacaktır. Bütün bileşenler için gerekli sertifikaları biz üretecek ve yapılandıracağız.

Ayrıca, **Elastic Common Schema (ECS)** denilen ortak şema standardını izleyeceğiz. ECS, Elastic'in tüm log kaynakları için önerdiği alan adlandırma şemasıdır. Örneğin loglarda kaynak IP adresi her zaman `source.ip` alanında, kullanıcı adı `user.name` alanında, vs. tutulur. SIEM **tespit kuralları** ECS alanlarını kullanarak çalıştığından, Logstash ile işlenen tüm logların bu şemaya uygun olması gerekir. Beats gibi Elastic'in kendi ajanları zaten ECS formatında veri gönderir. Syslog gibi kaynaklardan gelen ham verileri de ECS'ye uyacak şekilde parse edeceğiz.

Devam eden bölümlerde önce her bileşenin kurulumu ve güvenlik yapılandırması ele alınacak, sonra verilerin işlenmesi ve SIEM özelliklerinin kullanımı incelenecektir.

> **Neden:** Tek bir sunucuda tüm bileşenleri toplamak, küçük ortamlar veya eğitim amaçlı laboratuvarlar için idealdir. Ancak üretimde Elasticsearch genellikle ayrı bir küme olarak dağıtılır, Logstash ve Kibana da ayrı sunucularda ölçeklendirilir [5]. Bu kılavuzdaki tek-host kurulumu, konseptlerin anlaşılması ve hızlı bir POC (Proof of Concept) için tasarlanmıştır. Yine de güvenlik ayarları ve mimari, daha büyük dağıtımlara kolaylıkla uyarlanabilir şekilde ele alınmıştır.

Elasticsearch Kurulumu ve X-Pack Güvenlik Yapılandırması

Elastic Stack'in kalbinde yer alan **Elasticsearch**, verileri indeksleyen ve sorgulayan arama motorudur. Bu bölümde Elasticsearch 8.x'in Ubuntu 22.04 ortamına kurulumu, gerekli sistem ayarlarının yapılması ve **X-Pack Security** özelliklerinin (TLS, kimlik doğrulama, izinler) etkinleştirilmesini adım adım işleyeceğiz. Ayrıca **PKCS#12 ve PEM** sertifika formatları arasındaki farklara değinip, **enrollment token** kullanımını açıklayacağız.

Elasticsearch Paketinin Kurulumu (DEB)

Ubuntu tabanlı sistemlerde Elasticsearch, Elastic'in sağladığı APT deposu kullanılarak kolayca kurulabilir. Aşağıdaki adımlar, resmi Elastic dokümantasyonunda belirtildiği şekildedir [6] [7]:

1. **Elastic APT deposunu ekleyin:** Elastic'in imzalama anahtarını sisteme ekleyip depo kaynağını tanımlayın. Öncelikle GPG anahtarını indirelim ve ekleyelim:

   ```
   sudo apt-get install apt-transport-https ca-certificates curl gnupg
   curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo
   gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
   ```

   Ardından depo tanımını oluşturun:

   ```
   echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] \
   https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /
   etc/apt/sources.list.d/elastic-8.x.list
   ```

   > **Not:** `add-apt-repository` yerine yukarıdaki echo yöntemi kullanılmıştır. Bazı sistemlerde `add-apt-repository` komutu yoktur veya yanlışlıkla `deb-src` kaynağı ekleyip hataya yol açabilir [8]. Verilen yöntem, depo bilgisini temiz şekilde ayrı bir dosyada oluşturmaktadır.

2. **Elasticsearch'i kurun:**

   ```
   sudo apt-get update && sudo apt-get install elasticsearch
   ```

   Bu komut, Elasticsearch'ü varsayılan ayarlarla `/etc/elasticsearch` konfigürasyon diziniyle ve bir systemd servisi olarak kuracaktır.

3. **Sistem ayarları ve servis başlatma:** Kurulumdan sonra Elasticsearch servisini henüz başlatmayın. Önce bazı sistem ayarlarının kontrol edilmesi gerekir. Elasticsearch, bellek kilitleme, açık dosya tanımlayıcı sayısı, sanal bellek gibi konularda öneriler sunar (bootstrap checks). Özellikle `vm.max_map_count` değeri en az `262144` olmalıdır (genellikle Ubuntu'da bu değer zaten uygun gelir). Ayrıca swap kullanımını devre dışı bırakmak veya `bootstrap.memory_lock=true` ile kilitlemek önerilir, ancak tek düğüm lab ortamında swap kapatılması zorunlu değildir. Yine de performans için swap'i kapatabilirsiniz:

   ```
   sudo swapoff -a
   ```

   Artık servisi başlatabilirsiniz:

   ```
   sudo systemctl daemon-reload
   sudo systemctl enable elasticsearch.service
   sudo systemctl start elasticsearch.service
   ```

   Elasticsearch ilk kez başlatılırken güvenlik otomatik yapılandırması yapacağı için biraz zaman alabilir.

4. **İlk başlangıçta otomatik güvenlik yapılandırması:** Elasticsearch 8 ilk defa başlatıldığında, eğer siz farklı bir ayar yapmadıysanız, aşağıdaki işlemleri **otomatik olarak** gerçekleştirir [9] [10]:

5. Transport (9300) ve HTTP (9200) katmanları için TLS sertifikaları ve anahtarları oluşturur [1]. (Bu sertifikalar `/etc/elasticsearch/certs/` dizinine kaydedilir; kendi imzaladığı bir CA oluşturur.)
6. **X-Pack Security** ayarlarını `elasticsearch.yml` dosyasına uygular (örn. `xpack.security.enabled`, TLS ayarları vb. ilgili satırları ekler).
7. `elastic` süper kullanıcı için rastgele bir parolayı bellekte üretir ve **kurulum çıktısında** gösterir (ancak bu parola systemd hizmetiyle arka planda başladığı için loglara yazılmaz).
8. Kibana'nın Elasticsearch'e güvenli bağlanabilmesi için tek seferlik bir **enrollment token** oluşturur [1].

Bu süreç tamamlandığında (genellikle birkaç saniye ile 1 dakika arası sürer), Elasticsearch node'u 9200 portundan dinlemeye hazır olacaktır. Bu noktada auto-config çıktısını görmek için servis loglarını kontrol etmelisiniz.

1. **Elastic kullanıcı parolasını ayarlama:** Eğer Elasticsearch bir terminal içinde çalıştırılsaydı (örn. `bin/elasticsearch` ile), ilk başlatmada `elastic` kullanıcısının geçici parolası konsola yazılırdı. Ancak systemd ile başlattığımız için bunu görmedik. Bu nedenle kendi parolamızı belirlememiz gerekiyor. Elastic, 8.x ile birlikte `elasticsearch-setup-passwords` aracını yerine artık `elasticsearch-reset-password` aracını kullanmamızı öneriyor [11]. Aşağıdaki komutu çalıştırın:

   ```
   sudo /usr/share/elasticsearch/bin/elasticsearch-reset-password -u
   elastic
   ```

Bu komut, `elastic` kullanıcısının mevcut şifresini sıfırlayıp yeni bir şifre belirlemenizi ister. Eğer etkileşimli modda kendi belirleyeceğiniz parolayı girmek isterseniz `-i` bayrağını kullanabilirsiniz:

```
sudo /usr/share/elasticsearch/bin/elasticsearch-reset-password -i -u
elastic
```

Yeni süper kullanıcı parolanızı **güvenli bir yerde saklayın** (ör. geçici olarak bir çevre değişkenine atayabilirsiniz:

```
export ELASTIC_PASSWORD="<yeni-şifreniz>"
```

Böylece sonraki komutlarda kullanmak kolay olur).

2. **Kurulumu doğrulama:** Elasticsearch servisi aktif mi ve güvenli modda mı çalışıyor kontrol edelim. Aşağıdaki komut ile Elasticseach'in 9200 portuna **HTTPS** üzerinden istek yapın:

```
curl --cacert /etc/elasticsearch/certs/http_ca.crt -u elastic:
$ELASTIC_PASSWORD https://127.0.0.1:9200/
```

Bu komut, Elasticsearch'in otomatik oluşturduğu **HTTP katmanı CA sertifikası** ile 127.0.0.1'e istek yapar, `elastic` kullanıcı adı ve belirlediğiniz şifreyi kullanarak kimlik doğrular. Eğer çıktı olarak JSON biçiminde küme bilgisi (name, cluster_uuid, versiyon vb.) alırsanız kurulum başarılı demektir:

```
{
  "name" : "elk.local",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "....",
  "version" : {
    "number" : "8.x.x",
    ...
  },
  "tagline" : "You Know, for Search"
}
```

Çıktıyı başarıyla aldıysanız, Elasticsearch artık TLS ile iletişim kuruyor ve şifre korumalı. Bu noktada **kullanıcı doğrulaması ve TLS aktif** olduğu için, **Kibana veya diğer istemciler bağlanırken** bu güvenlik bilgilerini kullanmak zorundalar.

Doğrulama: Elasticsearch servisi çalışmazsa `sudo journalctl -u elasticsearch -f` komutu ile anlık loglarına bakabilirsiniz. Kurulum sonrası beliren hatalar genellikle konfigürasyon veya sistem ayar eksikliklerine dayanır. Örneğin `max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144]` gibi bir hata, bahsettiğimiz kernel parametresinin yetersiz olduğunu belirtir. Bu durumda `sudo sysctl -w vm.max_map_count=262144` ile geçici çözüm uygulanabilir (kalıcı için `/etc/sysctl.conf`'a ekleyin) [12]. Başka bir sık karşılaşılan

hata, bellek kilitlemenin yapılamamasıdır (`memory locking requested for elasticsearch process but memory is not locked`). Bu uyarı, `bootstrap.memory_lock` ayarını etkinleştirdiyseniz swap'in kapalı olmasını önerir. Tek node dev ortamında hayati değildir, swap'i devre dışı bırakarak uyarıyı giderebilirsiniz.

Geri Alma: Elasticsearch'e TLS ve güvenlik ayarlarını etkinleştirdikten sonra eğer istemciler bağlanamıyor veya kritik bir sorun oluştuysa, geçici olarak **güvenliği devre dışı bırakıp** Elasticsearch'i tekrar başlatabilirsiniz. Bunun için `/etc/elasticsearch/elasticsearch.yml` içinde `xpack.security.enabled: false` satırını ekleyip servisi restart edin. Ancak bu, tüm kimlik doğrulamayı kapatacağı için **yalnızca acil durumlarda** tavsiye edilir. Alternatif olarak, sorunu gidermeye çalışırken 9200'ü geçici olarak lokal erişime açmak için firewall ayarı yapabilir veya config'de `network.host: 127.0.0.1` bırakarak dış dünyadan erişimi kısıtlayabilirsiniz. Sorun çözüldükten sonra mutlaka güvenlik tekrar etkinleştirilmeli ve doğru ayarlar uygulanmalıdır.

Elasticsearch Konfigürasyonu: `elasticsearch.yml` ve Güvenlik Ayarları

Kurulum tamamlandıktan sonra, Elasticsearch'ün yapılandırma dosyası olan `/etc/elasticsearch/elasticsearch.yml` üzerinde çeşitli ayarları gözden geçireceğiz. Debian paketi kurulumu ile birlikte gelen varsayılan konfigürasyon dosyası, çoğu satırı yorumlu olarak içerir ve bu satırların gerektiğinde açılması beklenir. Tek düğümlü bir kurulum yaptığımız için bazı kritik ayarları kontrol edelim:

- **cluster.name ve node.name:** Varsayılan cluster ismi "elasticsearch" gelir. Tek node'da değiştirmenize gerek yoktur ancak izleme ve yönetim kolaylığı için `cluster.name: elk-siem-demo` gibi bir isim verebilirsiniz. `node.name` ise sunucunun hostname'ine otomatik olarak set edilir (bizim durumda `elk.local`). Bu yeterlidir.

- **network.host:** Otomatik güvenlik yapılandırması aktif olduğu için Elasticsearch HTTP katmanı eğer **güvenlik açık** ve tek düğüm moddaysa default olarak sadece `localhost`'tan dinler. Bizim kurulumda **TLS açık olduğu** için ve cluster join ayarları `discovery.type: single-node` durumunda, `network.host` büyük ihtimalle `0.0.0.0` olarak set edilmiştir (ilk başlangıçta auto-config bunu yapar). Bunu `elasticsearch.yml` içinde kontrol edin. Dışarıya servis vermeyecekseniz veya sadece localhost erişimi isterseniz `network.host: 127.0.0.1` ayarı yapabilirsiniz. Ancak Kibana aynı hostta olacağından localhost kullanması sorun yaratmaz. (Kibana da localhost üzerinden bağlanabilir.)

- **discovery.type:** Tek düğüm olduğu için kesinlikle `discovery.type: single-node` ayarı yapılmalıdır. Aksi halde Elasticsearch bir cluster kurmaya çalışacak ve başka master node bulamadığı için seçim yapamayarak faaliyete geçmeyecektir. Auto-config süreci sırasında bu ayarın eklenmiş olması gerekir, yoksa kendimiz ekleyelim.

- **Güvenlik ve TLS ile ilgili ayarlar:** Otomatik kurulum sırasında **TLS sertifikaları oluşturulup konfigürasyona eklendi** dedik. Bu, `elasticsearch.yml` içine çeşitli `xpack.security...` satırlarının eklendiği anlamına gelir. Örneğin (gerçek dosyadan bir örnek):

```
xpack.security.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.keystore.path: /etc/elasticsearch/certs/elastic-
stack-ca.p12
```

```
xpack.security.transport.ssl.truststore.path: /etc/elasticsearch/certs/
elastic-stack-ca.p12
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.keystore.path: /etc/elasticsearch/certs/http.p12
xpack.security.http.ssl.truststore.path: /etc/elasticsearch/certs/http.p12
```

Bu ayarlar örnek olarak verilmiştir. Gerçek dosyada farklı isimlerde .p12 dosyaları referans edilebilir. Temel olarak: - **transport.ssl** ayarları cluster içi haberleşmenin TLS ile ve karşılıklı sertifika doğrulamasıyla gerçekleşeceğini söyler. Tüm düğümlerin (bizde tek düğüm) aynı CA tarafından imzalanmış sertifikalara sahip olması beklenir. `verification_mode: certificate` ayarı, karşılıklı doğrulamada sadece sertifikanın geçerli bir CA'den olduğunun kontrol edildiğini, isim doğrulaması yapılmadığını belirtir. Tek düğümlü cluster'da bu sorun değildir. - **http.ssl** ayarları, istemci (Kibana, kullanıcı vb.) ile iletişimi şifreler. .p12 uzantılı dosyalar, PKCS#12 konteyner formatındadır ve içinde sertifika + özel anahtar + CA sertifikası bulunur. Bu dosyalar Elastic'in kendi `elasticsearch-certutil` aracıyla oluşturulmuştur.

PKCS#12 yerine PEM formatı da kullanılabilir. Örneğin eğer `.pem` uzantılı sertifika ve `.key` dosyalarınız olsaydı, ayarlar şu şekilde olacaktı:

```
xpack.security.http.ssl.certificate: /etc/elasticsearch/certs/elk.local.crt
xpack.security.http.ssl.key: /etc/elasticsearch/certs/elk.local.key
xpack.security.http.ssl.certificate_authorities: [ "/etc/elasticsearch/certs/
CA.crt" ]
```

Bu durumda truststore/keystore yerine doğrudan pem dosyaları kullanılırdı. **PKCS#12** (PFX olarak da bilinir), Java dünyasında sık kullanılan bir tümleşik formattır ve Elasticsearch Java tabanlı olduğu için bunu doğrudan okuyabilir. **PEM** formatı ise metin tabanlı Base64 kodlu sertifika/anahtar içerir ve daha insan okunur haldedir. Elastic ürünleri her ikisini de destekler, ancak Kibana (Node.js tabanlı) genelde PEM formatını kullanımını kolaylaştırır. Bu kılavuzda sertifikaları oluşturmada PKCS#12 kullanıp gerektiğinde PEM'e dönüştürmeyi de göstereceğiz.

- **İzinler ve diğer ayarlar:** Debian paket kurulumunda, konfigürasyon dizini `/etc/elasticsearch` ve altındaki dosyalar kök (root) kullanıcıya ait, grup ise `elasticsearch` şeklinde ayarlanmıştır [13]. Ayrıca `/etc/default/elasticsearch` adında bir dosya bulunur; burada Java heap bellek ayarları veya GC ayarları gibi environment değişkenleri tanımlanabilir (varsayılan Heap genelde 1/2 RAM olacak şekilde ayarlanır, küçük test sistemlerinde 1-2 GB bırakmanız yeterli). Bu dosyada `ES_PATH_CONF` vb. ayarlar da bulunur. Systemd ile servis çalıştığından, limit ayarları için de `/usr/lib/systemd/system/elasticsearch.service` içinde `LimitNOFILE=65535` gibi değerler zaten gelir. Yine de, Elastiksearch'in kendini başlatırken yaptığı **bootstrap checks** adı verilen kontrol listesini geçebildiğinden emin olun. (Komutu: `sudo -u elasticsearch /usr/share/elasticsearch/bin/elasticsearch -d -p pid &` ile debug modda çalıştırıp çıkışına bakmak bir yöntem olabilir, ancak servisi zaten başarılı başladıysa bunlara gerek kalmamıştır.)

Özetle, `elasticsearch.yml` dosyasını kurcalarken çok az değişiklik yapmamız gerekecek. Otomatik ayarlar işimizi gördü. Sadece tek düğüm mod, network host gibi kritik ayarlar doğru ise bırakın öyle kalsın.

Aşağıda örnek bir `elasticsearch.yml` ilgili kısımlarıyla verilmiştir:

```
cluster.name: elk-siem-cluster
node.name: elk.local
discovery.type: single-node
# Güvenlik ve TLS ayarları (auto-config ile eklenmiş olmalı):
xpack.security.enabled: true
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.keystore.path: /etc/elasticsearch/certs/elastic-
stack-ca.p12
xpack.security.transport.ssl.truststore.path: /etc/elasticsearch/certs/
elastic-stack-ca.p12
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.keystore.path: /etc/elasticsearch/certs/http.p12
xpack.security.http.ssl.truststore.path: /etc/elasticsearch/certs/http.p12
```

> **Neden:** Elasticsearch'i kurar kurmaz güvenlik ayarlarını yapmak, daha sonra oluşabilecek karmaşık sorunları engeller. Varsayılan (security kapalı) modda Elasticsearch 8.x zaten çalışmaz (güvenlik devre dışı bırakılmadıkça). Şifreleme ve kimlik doğrulama, SIEM gibi hassas log verilerinin işlendiği bir sistemde mutlak gerekliliktir. Tek bir düğümde çalışsak bile, sonraki aşamalarda Kibana ve Logstash'ın **Elasticsearch ile haberleşmesi** bu güvenlik parametrelerine uygun olmalıdır. Örneğin Kibana, Elastic'e bağlanırken mutlaka TLS üzerinden bağlanmalı ve geçerli bir kullanıcı adı/şifre kullanmalıdır; aksi halde bağlantı reddedilir.

elasticsearch-certutil ile TLS Sertifikası Oluşturma (Opsiyonel)

Elasticsearch'in ilk başlatmada kendi oluşturduğu sertifikalar yerine, biz kendimiz de bir **sertifika otoritesi (CA)** ve ona bağlı düğüm sertifikaları üretebiliriz. Bu, özellikle birden fazla sunucuya dağıtım yaparken veya sertifikaların süresini/alan adını özelleştirmek istediğimizde yararlıdır. Elastic, bu iş için `elasticsearch-certutil` adlı bir komut satırı aracı sağlar [14] [15].

`elasticsearch-certutil`, dört modda çalışabilir: `ca`, `cert`, `csr`, `http` [16].

- **CA modu:** Kendi kendine imzalayan yeni bir CA sertifikası ve anahtarı oluşturur. (Komut: `elasticsearch-certutil ca`). Bu çıktıyı .p12 veya .pem formatında alabilirsiniz. PKCS#12 istiyorsanız interaktif soruya boş şifre girerek geçebilirsiniz veya `--pem` parametresi ile .pem çıktı alabilirsiniz.
- **Cert modu:** Mevcut bir CA'den bir veya birden fazla node için sertifika/imzalama işlemi yapar. (Komut: `elasticsearch-certutil cert --ca <ca.p12>`). Bu mod, tek komutta birden fazla sertifika üretebilir, veya bir YAML girdi dosyası ile birden çok node'u tek seferde tanımlayabilirsiniz (silent mode).
- **CSR modu:** Sertifika imzalama isteği (CSR) oluşturur. Örneğin Kibana için CSR üretip şirketinizin iç CA'ine imzalatmak istiyorsanız işe yarar [17].
- **HTTP modu:** Elasticsearch HTTP arayüzü için özel bir sertifika paketi oluşturur. Bu mod aslında yukarıdaki modların etkileşimli bir kombinasyonu gibi çalışır; size sorular sorarak ihtiyacınıza göre bir paket hazırlar [15]. Örneğin CA yoksa bir CA ile birlikte, Kibana için de sertifika içerecek şekilde bir zip paketi çıkartabilir.

PKCS#12 vs PEM: `elasticsearch-certutil` çıktıları varsayılan olarak .p12 formatındadır, ancak `--pem` bayrağı ile çıktıyı PEM formatında alabilirsiniz [18] [19]. PKCS#12 formatı (genellikle `.p12` uzantılı), hem özel anahtarı hem sertifikayı (ve hatta CA sertifikasını) tek bir şifrelenmiş dosyada tutar. Bu dosyayı Elasticsearch ve Logstash doğrudan kullanabilir. PEM formatında ise genelde ayrı `.crt` (sertifika) ve `.key` (anahtar) dosyaları olur, bunları konfigürasyonda ayrı ayrı belirtmek gerekir. Kibana, Node.js tabanlı olduğu için .pem kullanımı yaygındır (veya `.pem` içeriğini .pem uzantılı bir keystore dosyası gibi de kabul edebilir). Logstash her iki formatı da destekler, ancak genellikle .pem ile kullanırız (özellikle input plugin'lerinde TLS ayarlarında `.pem` istenir).

Şimdi bir örnek yapalım: Kendi CA'mizi oluşturup, Elasticsearch ve Kibana için sertifika üretelim. Bu adımlar **opsiyonel**dir, çünkü halihazırda çalışan bir otomatik sertifikasyon mevcut. Ancak süreç öğrenimi için değerlidir:

1. **CA Oluşturma:** (Eğer hali hazırda `http_ca.crt` gibi bir dosyanız yoksa veya kendiniz yapmak istiyorsanız)

   ```
   sudo /usr/share/elasticsearch/bin/elasticsearch-certutil ca --out
   myElasticCA.p12 --pass ""
   ```

   Komut bize `myElasticCA.p12` adında bir CA oluşturarak verdi (şifresiz, --pass "" dedik). Bunu PEM tercih etseydik `--pem` eklerdik ve zip içinde `ca.crt` ve `ca.key` alırdık.

2. **Node Sertifikası Oluşturma:** Tek node için (transport+http ikisi için de kullanılabilir) bir sertifika üretelim:

   ```
   sudo /usr/share/elasticsearch/bin/elasticsearch-certutil cert --ca
   myElasticCA.p12 --ca-pass "" \
       --name elk.local --dns elk.local --ip 127.0.0.1 --out elk-
   bundle.p12 --pass ""
   ```

   Bu komut, CA olarak az önce ürettiğimiz myElasticCA.p12'yi kullanarak `elk.local` ismine ait bir sertifika paketi üretir. İçinde hem elk.local sertifikası hem de CA sertifikası bulunur. `--dns` ile sertifikaya eklenen Subject Alternative Name (SAN) değerini elk.local yaptık, `--ip` ile de 127.0.0.1 ekledik. Bu sayede istemciler IP veya elk.local ismiyle bağlandığında sertifika eşleşmesi sorun olmayacak. Çıktı `elk-bundle.p12` içine şifresiz kaydedildi.

   Not: Gerçekte, transport ve http için ayrı sertifikalar üretip farklı dosyalar kullanma olanağı var. Fakat tek node'da ikisi de aynı olabilir. Birden fazla node'lu cluster'da, transport sertifikaları tüm node'lar tarafından karşılıklı güvenileceğinden genelde aynı CA'dan imzalı olmalı. HTTP için ise ister aynı sertifikayı kullanırsınız, ister farklı (genelde farketmez, ama domain isimleri bakımından belki her node'un HTTP sertifikası kendi hostname'ini içerir).

1. **Dosyaları yerleştirme ve konfigürasyon:** Artık elimizdeki `myElasticCA.p12` (CA) ve `elk-bundle.p12` (node sert) dosyalarını /etc/elasticsearch/certs altına kopyalayalım:

```
sudo mkdir -p /etc/elasticsearch/certs
sudo cp myElasticCA.p12 elk-bundle.p12 /etc/elasticsearch/certs/
sudo chown root:elasticsearch /etc/elasticsearch/certs/*.p12
sudo chmod 640 /etc/elasticsearch/certs/*.p12
```

Ardından `elasticsearch.yml` içinde otomatik ayarlar yerine bunları gösterecek şekilde değişiklik yapalım:

```
xpack.security.transport.ssl.keystore.path: /etc/elasticsearch/certs/
elk-bundle.p12
xpack.security.transport.ssl.truststore.path: /etc/elasticsearch/certs/
elk-bundle.p12
xpack.security.http.ssl.keystore.path: /etc/elasticsearch/certs/elk-
bundle.p12
xpack.security.http.ssl.truststore.path: /etc/elasticsearch/certs/elk-
bundle.p12
```

Gördüğünüz gibi hem transport hem http aynı paketi kullanıyor. Bu pakette CA da gömülü olduğu için güvenmek için ayrı truststore vermeye de gerek yok aslında (ancak Elastic config dosyasında keystore ile truststore'u aynı dosyaya verebiliyoruz). Bu ayarlarla servisi restart ettiğinizde, artık Elasticsearch sizin oluşturduğunuz sertifikalarla çalışacaktır. Güvenlik açısından bir değişiklik yok – sadece kendi imzamızı kullanmış olduk.

2. **Kibana için sertifika:** Benzer şekilde Kibana'nın HTTPS sunabilmesi için bir sertifikaya ihtiyacı vardır. Bunu da aynı CA'den imzalayabiliriz ki tarayıcıya CA'yı tek seferde ekleyince Kibana ve Elasticsearch için ayrı ayrı uğraşmayalım. Bir Kibana sertifikası oluşturalım:

```
sudo /usr/share/elasticsearch/bin/elasticsearch-certutil cert --ca
myElasticCA.p12 --ca-pass "" \
    --name kibana.elk.local --dns kibana.elk.local --ip 127.0.0.1 --pem
--out kibana-cert.zip
```

Bu kez `--pem` kullandık ve çıktı bir zip dosyası (kibana-cert.zip) olarak geldi. İçini açarsak şöyle:

```
kibana.elk.local.crt   (Kibana sunucu sertifikası)
kibana.elk.local.key   (Kibana özel anahtarı)
ca.crt                 (CA sertifikası, tarayıcılar bu CA'yı tanımadığı
için bunu import edebilirsiniz gerekirse)
```

Bu dosyaları da Kibana'nın kullanacağı yere koyalım:

```
sudo mkdir -p /etc/kibana/certs
sudo cp kibana.elk.local.crt kibana.elk.local.key /etc/kibana/certs/
sudo cp myElasticCA.p12 /etc/kibana/certs/CA.p12
sudo cp ca.crt /etc/kibana/certs/CA.crt
```

```
sudo chown root:kibana /etc/kibana/certs/*
sudo chmod 640 /etc/kibana/certs/*
```

Kibana `.p12` formatını da destekler ancak orada genelde PEM gösterilir. Bu nedenle `.crt` ve `.key` dosyalarını verdik. CA.crt dosyasını da Elasticsearch'ün sertifikasına güvenmesi için kullanacağız.

Bu şekilde ürettiğimiz kendi sertifikalarımızı kullanmak, tüm sistemlerde tek bir CA yönetimi ile çalışmamızı sağlar. Birden fazla sunuculu (ör. ayrı Logstash sunucusu, ayrı Kibana sunucusu) ortamda bu şarttır. Tek sunucuda otomatik ayarları kullanmak da yeterlidir, ancak burada sertifika üretme bilgisini aktarmış olduk.

Neden: Özellikle kurumsal ortamlarda güvenlik politikaları gereği kendi CA'inizi kullanmanız istenebilir. Elastic Stack, kendi oluşturduğu geçici CA'yı kullanmış olsa bile siz bu sertifikaları her zaman değiştirebilirsiniz. **Elasticsearch-certutil** aracı, hataya açık OpenSSL komutlarıyla uğraşmadan, doğru formatta sertifikalar üretmeyi kolaylaştırır [20] [21]. PKCS#12 dosyalarının şifrelenmesi de mümkündür; ancak bu durumda Elasticsearch'e her açılışta keystore parolasını vermeniz gerekir veya keystore parolasını elasticsearch.keystore içinde tanımlayabilirsiniz. Genellikle test ve küçük ortamlarda parolasız P12 kullanmak pratik oluyor.

Doğrulama: Kendi sertifikalarımızı uyguladıktan sonra Elasticsearch'ü yeniden başlatıp `curl` komutu ile bağlantıyı test edelim. Bu sefer `--cacert` olarak kendi CA'mizin sertifikasını kullanacağız:

```
curl --cacert /etc/kibana/certs/CA.crt -u elastic:
$ELASTIC_PASSWORD https://elk.local:9200
```

Eğer DNS ayarınızda **elk.local** ismi 127.0.0.1'e çözülüyorsa (hosts dosyasına ekleyebilirsiniz), bu komut sorunsuz çalışacaktır. Sorun olursa (örn. "certificate issuer unknown" gibi bir hata) muhtemelen yanlış CA kullanmışsınızdır. `--cacert` parametresi ile doğru CA'yı (bizim oluşturduğumuz) gösterdiğinizden emin olun.

Ayrıca `openssl s_client -connect elk.local:9200 -CAfile /etc/kibana/certs/CA.crt` komutuyla da sertifika detaylarını ve zincir doğrulamasını görebilirsiniz. CN, SAN gibi değerlerin beklendiği gibi olup olmadığını inceleyin.

Geri Alma: Elasticsearch'e özel sertifika yüklerken bir hata yaparsanız (örneğin keystore dosyası yolu yanlış veya şifre sorması vs.), Elasticsearch başlatılamayabilir. Böyle bir durumda eski ayarlara dönmek (auto-config kullandığı ayarlara) bir seçenek veya `xpack.security.http.ssl.enabled: false` yaparak geçici olarak TLS'i kapatıp sistemi çalışır hale getirmek gerekebilir. TLS'i kapatmak, Kibana ve Logstash'ın da yeniden ayarlanmasını gerektirir, bu nedenle sorunu düzeltip TLS'i en kısa sürede geri açmalısınız. Bir diğer geri alma yöntemi, hatalı P12 veya PEM dosyaları yerine orijinal auto-generated dosyaları tekrar koymaktır (sildiyseniz, elasticsearch-certutil ile `http` modunda yenisini oluşturabilirsiniz).

Enrollment Token ile Elasticsearch'e Bileşen Ekleme

Enrollment token kavramı, Elastic 7.13+ sürümlerinde gelen bir kolaylıktır. Bu token, yeni bir Elasticsearch düğümünü var olan bir cluster'a eklerken veya yeni bir Kibana örneğini Elasticsearch cluster'ına bağlarken kullanılan zaman kısıtlı bir anahtardır. İçinde hedef cluster'ın CA sertifikası bilgisi ve kayıt sırasında kullanılacak geçici kimlik bilgileri bulunur [2].

Tek düğümlü kurulumumuzda, aslında enrollment token sadece Kibana'yı ilk kurarken işe yarayacak. Yukarıda kurulum adımlarında Elasticsearch ilk başlatmada bir token oluşturdu demiştik. Eğer Kibana, Elasticsearch ile aynı makinede kuruluysa ve erişim sorunu yoksa token kullanmadan da manuel konfigürasyon yapabiliriz. Ancak pratikte enrollment token şu akışta kullanılır:

- Elasticsearch ilk node kuruldu -> size bir **Kibana enrollment token** verdi.
- Kibana'yı kurup çalıştırdığınızda eğer Kibana henüz konfigurasyonu yapılmamışsa ilk açılışta bu token'ı sorar.
- Token'ı girdiğinizde Kibana otomatik olarak Elastic'e güvenli biçimde bağlanacak şekilde `kibana.yml` dosyasına ayarları yazar (ör. CA sertifikasını, kibana_system kullanıcı şifresini vs.) [2].

Ayrıca, cluster'a yeni bir Elasticsearch node ekleyeceğiniz zaman da: - Mevcut node'da `elasticsearch-create-enrollment-token -s node` komutuyla bir node token üretirsiniz [22]. - Yeni node'un config'ine bu token'ı verirsiniz veya ilk açılış parametresi olarak iletirsiniz; böylece güvenlik ayarlarını otomatik alarak join eder.

Bizim senaryomuzda **Kibana enrollment token** konusuna odaklanalım. Elasticsearch kurulum loglarından veya `journalctl -u elasticsearch` çıktısından muhtemelen bir satırda token'ı yakalayabilirsiniz. Format genelde uzun bir Base64 string gibi olur.

Eğer token'ı bulamadıysanız veya süresi geçtiyse (varsayılan 30 dakika geçerlidir [23]), kendiniz de oluşturabilirsiniz:

```
sudo /usr/share/elasticsearch/bin/elasticsearch-create-enrollment-token -s
kibana
```

Bu komut size konsolda tek seferlik bir kayıt token'ı basacaktır. Şimdi Kibana kurulumuna geçtiğimizde bunu kullanacağız.

Unutmayın, bu token hassas bilgidir; herhangi biri eline geçirirse 30 dakika içinde Kibana'nızı sizin cluster'a bağlanır gibi bağlamaya çalışabilir. Yine de bu sadece ilk kurulum için geçerlidir.

> **Neden:** Enrollment token mekanizması, **"detection-as-code"** gibi altyapıları kodla yönetme değil, *kurulum aşamasını kolaylaştırma* aracıdır. Özellikle çok düğümlü ve güvenlik açık bir cluster kurarken sertifika dağıtımı ve kullanıcı parolası derdiyle uğraşmamak için geliştirilmiştir. Token, gereken güvenlik ayarlarını Kibana'ya otomatik yazdığı için hataya yer kalmaz [2]. Biz uzmanlar için manual ayarlar basit görünse de, özellikle GUI ağırlıklı kullanıcılar veya hızlı kurulum isteyenler için bu pratik bir yoldur. Bu rehberde, biz hem token ile otomatik bağlanmayı göstereceğiz (Kibana bölümünde), hem de istenirse manuel ayar yolunu anlatacağız.

Kibana Kurulumu ve Yapılandırması

Elasticsearch kurulduğuna ve çalışır durumda olduğuna göre, sıradaki adım **Kibana** arayüzünü kurmaktır. Kibana, kullanıcıların görsel arayüzü kullanarak Elastic Stack ile etkileşime geçtiği bileşendir ve SIEM uygulamasını barındırır. Bu bölümde Kibana'nın Ubuntu 22.04 üzerine kurulumu, gerekli konfigürasyon ayarlarının yapılması, **Kibana keystore** kullanımını ve **systemd servis olarak yönetimini** ele alacağız. Ayrıca Kibana'nın Elasticsearch'e güvenli bağlanması için **enrollment token** ile otomatik yapılandırma ve manuel yapılandırma adımlarını da işleyeceğiz.

Kibana Paketinin Kurulumu (DEB)

Kibana da Elastic'in apt deposundan kurulabilir. Adımlar yine resmi dokümantasyonla uyumludur [24] :

1. **APT deposu ve GPG anahtarı:** Elasticsearch kurulumu sırasında zaten Elastic apt deposunu eklemiştik. Aynı depo Kibana paketini de içerir. (Eğer atlandıysa önceki adımlarda belirtildiği gibi GPG anahtarını ve depo satırını ekleyin).

2. **Kibana'yı kurun:**

   ```
   sudo apt-get update && sudo apt-get install kibana
   ```

   Bu komut Kibana'nın ilgili sürümünü indirip `/etc/kibana` konfigürasyon diziniyle birlikte yükleyecektir. Varsayılan olarak Kibana servisi kurulmuş ancak devre dışı durumda gelir (enable yapmamız gerekecek).

3. **Kibana'nın Elasticsearch'e Erişimi:** Kibana'nın sorunsuz çalışması için, öncelikle Elasticsearch'ün çalışır ve güvenlik kurulumunun tamamlanmış olması gerekir. Yukarıdaki adımlarda bunu yaptık. Kibana kurulduktan sonra, çalıştırmadan önce bir **enrollment token** kullanarak Elasticsearch cluster'a bağlayabiliriz. Bunu iki yöntemle yapabiliriz:

4. Kibana web arayüzü üzerinden (eğer tarayıcı erişiminiz varsa).
5. Komut satırı ile (sunucuda tarayıcı yoksa veya uzaktaysanız).

Kibana 8.x'te, servis ilk kez başlatıldığında loglarına bir link ve doğrulama kodu yazar. Ancak en kolay yöntem, komut satırında token'ı vermektir. **Kibana 8+ ile birlikte gelen yeni bir araç** var: `kibana-setup`. Bu aracı kullanarak token'ı girebiliriz.

Önce Kibana'yı servis olarak çalışacak şekilde etkinleştirelim:

```
sudo systemctl daemon-reload
sudo systemctl enable kibana.service
```

Bu, sistemi her açtığınızda Kibana'nın başlamasını da sağlar. Şimdi Kibana'yı henüz başlatmadan `kibana-setup` ile kayıt yapalım:

```
sudo /usr/share/kibana/bin/kibana-setup --enrollment-token "<sizdeki-kibana-enrollment-token>"
```

Bu komutu, Elasticsearch tarafında elde ettiğimiz token ile çalıştırdığımızda, Kibana şu işlemleri otomatik yapacaktır [25] [26] : - Elasticsearch cluster'ının CA sertifikasını kendi config dizinine (ör. `/etc/kibana` altına) kaydeder. - Kibana ile Elasticsearch arasındaki güvenli bağlantı ayarlarını (`elasticsearch.hosts`, `elasticsearch.ssl.certificate_authorities`, `elasticsearch.username`, `elasticsearch.password` gibi) `kibana.yml` dosyasına yazar [2] . - Kibana için gereken kibana_system kullanıcısının kimlik doğrulamasını sağlar. - Ek olarak, eğer Kibana'nın kendi TLS sertifikası yoksa (web arayüzü için), bu aşamada gene Kibana kurulumu sırasında bir geçici kendinden imzalı sertifika oluşturabilir veya sizden prompt ile bir şeyler isteyebilir (genellikle dev ortamı uyarısı verip kendi sertifikasını oluşturur).

Eğer `kibana-setup` başarılı olursa konsolda "Kibana sucecssfully configured" benzeri bir çıktı alırsınız.

1. **Kibana'yı başlatma:**

   ```
   sudo systemctl start kibana.service
   ```

   Kibana arka planda birkaç saniye içinde ayağa kalkacaktır. Durumunu kontrol etmek için:

   ```
   sudo systemctl status kibana.service
   ```

   komutunu kullanın. İlk defa kurulduğunda, Kibana logunda bir altı haneli doğrulama kodu görebilirsiniz (ör. `Go to http://localhost:5601/?code=123456 to get started` şeklinde). Bu, tarayıcıdan ilk bağlantıda token'ı girmediyseniz kullanacağınız koddur. Biz CLI'dan token verdiğimiz için muhtemelen buna gerek kalmadı.

2. **Kibana'ya tarayıcı ile erişim:** Artık makinenizde Kibana 5601 portunda çalışıyor olmalı. Tarayıcıdan `https://elk.local:5601` (veya IP kullanıyorsanız `https://127.0.0.1:5601`) adresine gidin. Eğer kendi oluşturduğunuz sertifikayı Kibana'ya vermediyseniz (Kibana şu an muhtemelen self-signed bir sertifika ile çalışıyor), tarayıcı güvenlik uyarısı verecektir. Devam edip siteye girin. Karşınıza "Welcome to Elastic" sayfası gelecektir. Burada `elastic` kullanıcı adı ve daha önce belirlediğiniz şifre ile giriş yapın.

Başarılı giriş yaptıysanız, Kibana arayüzünü ve içinde **Security (SIEM)** uygulamasını görebileceksiniz.

1. **Kibana'nın TLS ile çalıştırılması (Kibana Web HTTPS):** Varsayılan kurulumda Kibana, 5601 portunda **HTTP** olarak çalışır. Bunu da TLS ile güvenli yapmak önemlidir (özellikle üretimde). Kendi oluşturduğumuz `kibana.elk.local.crt` ve `.key` dosyalarını kullanarak Kibana'yı HTTPS yapalım. Bunun için `kibana.yml` dosyasına aşağıdaki ayarları ekleyin veya düzenleyin:

   ```
   server.host: "elk.local"
   server.port: 5601
   server.ssl.enabled: true
   server.ssl.certificate: /etc/kibana/certs/kibana.elk.local.crt
   server.ssl.key: /etc/kibana/certs/kibana.elk.local.key
   ```

   Bu, Kibana'nın kendi web sunucusunu TLS ile çalışacak şekilde yapılandırır [27] [28] . Değişiklikten sonra Kibana servisini yeniden başlatın:

```
sudo systemctl restart kibana
```

Artık tarayıcıdan `https://elk.local:5601` adresine gittiğinizde sertifika kibana.elk.local adına olduğu için DNS çözümlemesinin o isme uyması gerekir. hosts dosyanıza `127.0.0.1 elk.local kibana.elk.local` gibi bir ekleme yaparak tarayıcının bu isimleri 127.0.0.1'e yönlendirmesini sağlayın. Tarayıcıya CA'nızı güvenilir olarak eklediyseniz (veya Public CA kullanmış olsaydınız) artık güvenlik uyarısı da almazsınız.

Not: Bu ayarları yaparken `elasticsearch.hosts` veya `elasticsearch.ssl.certificateAuthorities` gibi Kibana'nın Elastic'e bağlanma ayarlarına dokunmadık. `kibana-setup` bunları zaten koymuştu. Örneğin `elasticsearch.hosts: ["https://127.0.0.1:9200"]` ve `elasticsearch.password` vb. keystore'dan çekilecek şekilde halledilmiş durumda. Keystore konusuna hemen değineceğiz.

1. **Kibana Keystore ile Hassas Ayarlar:** Kibana'nın Elasticsearch'e bağlanmak için kullandığı `elasticsearch.password` (kibana_system kullanıcısının şifresi) gibi değerler `kibana.yml` yerine güvenli bir depoda tutulabilir. **Kibana keystore**, tıpkı Elasticsearch keystore gibi, hassas ayarları saklayabildiğimiz bir şifreli dosyadır. Varsayılan olarak `/etc/kibana/kibana.keystore` şeklinde bulunur (ilk kurulumda oluşturulmamış olabilir). Bizim kurulumumuzda `kibana-setup` aracı kibana_system şifresini doğrudan keystore'a eklemiş olacak. Kontrol edelim:

```
sudo /usr/share/kibana/bin/kibana-keystore list
```

Eğer bir çıktı veriyorsa (muhtemelen `elasticsearch.password` öğesini görürsünüz), keystore zaten oluşmuş demektir. Kibana, konfigürasyonda eğer bir ayar yok ama keystore'da varsa onu kullanır [29]. Örneğin `kibana.yml` içinde `elasticsearch.password: ${elasticsearch.password}` gibi bir şey göremeyebilirsiniz; bu, Kibana'nın iç mekanizmayla keystore'daki `elasticsearch.password` anahtarını kullandığını gösterir.

Eğer elle yapmak isteseydik (enrollment token olmadan): - `kibana.keystore` oluştururduk: `./bin/kibana-keystore create` [29]. - Sonra kibana_system şifresini eklerdik: `./bin/kibana-keystore add elasticsearch.password` ve istenince şifreyi girerdik [30]. - `kibana.yml` içine de `elasticsearch.username: "kibana_system"` yazıp, `elasticsearch.password: "${elasticsearch.password}"` yazardık. Bu şekilde Kibana, configteki `${...}` ifadelerini environment veya keystore'dan tamamlar [31].

Kibana keystore'a başka hassas veriler de ekleyebilirsiniz: Örneğin xpack.encryptedSavedObjects.encryptionKey (Kibana'nın bazı özellikleri için rastgele uzun bir key), veya kurallar alarmı için SMTP şifresi, JSON web token vb. Tüm bunlar düz yazı yerine keystore'da tutulabilir.

1. **Kibana Varsayılan Ayarları ve Diğer İpuçları:**
2. Kibana, default olarak sadece localhost'tan erişilebilir (`server.host: "localhost"`). Dışarıdan erişim isterseniz `server.host: "0.0.0.0"` yapmanız gerekir [32]. Biz elk.local üzerinden aynı makinede kaldığımız için problem yok, ama farklı bir istemciden bağlanacaksanız bu ayara dikkat edin.
3. Kibana'nın dili İngilizce'dir; arayüzde dil desteği kısıtlıdır, ama biz teknik terimleri Türkçe anlatıyoruz, arayüzde göreceğiniz kısımları orijinal bırakacağız.

4. Systemd ile Kibana'nın loglarını takip etmek için: `journalctl -u kibana -f` kullanabilirsiniz. Başlangıçta özellikle Elastic'e bağlanma hatası olup olmadığını kontrol edin. Örneğin "Unable to retrieve version information from Elasticsearch nodes. Host is unreachable" gibi bir hata görürseniz Elasticsearch'e erişemiyor demektir (TLS ayarı veya yetki hatası olabilir).
5. Kibana'nın gelişmiş ayarları için `/etc/kibana/kibana.yml` içindeki örnekleri inceleyin veya resmi dokümandaki Kibana config referansına bakın. Çoğu ayar varsayılan uygun değerlerdedir.

Son durumda Kibana servisimiz çalışıyor, 5601 üzerinden TLS ile erişiyoruz, Elastic'e bağlantı kurulmuş, Kibana UI'dan giriş yapabiliyoruz. Sonraki adım verileri toplamaya başlayacak olan Logstash'ın kurulumu olacak.

> **Doğrulama:** Kibana arayüzüne giriş yaptıktan sonra sol menüde "Stack Management" > "Security" > "Users" sekmesine giderek Elasticsearch kullanıcılarını listeleyin. `elastic` ve bir dizi built-in kullanıcı göreceksiniz (kibana_system vs.). Bu liste geliyorsa Kibana, Elasticsearch'ten güvenli şekilde veri çekebiliyor demektir. Ayrıca Discover sayfasına gidip henüz veri olmadığından bir uyarı göreceksiniz, bu normal. Bir iki adım sonra verilerimizi gönderip burada görmeyi deneyeceğiz.
>
> **Geri Alma:** Kibana konfigürasyonunda yaptığınız değişiklikler sonucu Kibana başlamazsa (örn. TLS sertifika yolu hatalı, ya da elasticsearch bağlantısı hatalı), `sudo systemctl stop kibana` ile durdurun. Ardından `journalctl -u kibana -n 50` ile son 50 satır loga bakarak hatayı tespit edin. Gerekirse `kibana.yml` dosyasını eski haline getirip (veya yedeğiniz yoksa en azından son eklediklerinizi yoruma alıp) yeniden başlatın. Enrollment token ile kurulum yaptığınız durumda, eğer Kibana hiç açılmazsa token kullanımını atlayıp manuel ayar yoluna da dönebilirsiniz: `kibana.yml` içine gerekli `elasticsearch.hosts`, `elasticsearch.username` (`elastic` veya `kibana_system`), `elasticsearch.password` (geçici olarak düz yazı test için) ve `elasticsearch.ssl.certificateAuthorities` (CA path) yazarak Kibana'yı çalıştırmayı deneyebilirsiniz. Sorun çözüldüğünde bu düz parolayı keystore'a almayı unutmayın.
>
> Kibana TLS'ı geri almak isterseniz, `server.ssl.enabled: false` yapıp certificate ve key ayarlarını kaldırarak HTTP moduna dönebilirsiniz. Ancak kullanıcı parolaları yine de şifrelenmiş olarak gider (browser ile Kibana arası hariç), bu nedenle üretimde tüm trafiği TLS üzerinden tutmak en iyi pratiktir.

Kibana Keystore ve Ortam Değişkenleri ile Ayar Yönetimi

Kibana'nın önemli ayarlarının bir kısmından bahsettik. Bu alt bölümde, genel bir prensip olarak **hassas verilerin düz metin olarak config dosyalarında tutulmaması** konusunu vurgulayacağız. Elastic Stack bileşenlerinin her biri (Elasticsearch, Kibana, Logstash) kendine özgü bir *secure keystore* sistemine sahiptir:

- **Elasticsearch keystore:** `elasticsearch-keystore` komutuyla yönetilir ve Elasticsearch'ün `elasticsearch.yml` içindeki bazı ayarların (örn. cloud API key gibi) şifreli saklanmasına olanak tanır.
- **Kibana keystore:** `kibana-keystore` ile yönetilir. Genellikle Elastic'e bağlanma parolası veya üçüncü parti entegrasyon anahtarları için kullanılır.
- **Logstash keystore:** Bir sonraki bölümde ayrıntılandıracağız, pipeline içinde kullanılan parolaları saklamaya yarar.

Bu keystore'lar, disk üzerinde şifreli bir dosya oluşturup (gerekirse bu dosyayı da ayrıca bir parola ile koruyup) anahtar-değer çiftlerini içinde barındırır. Uygulama çalışırken bellekte çözülür, config dosyasında sadece referans geçilir. Örneğin Kibana için, `elasticsearch.password` değerini keystore'a koyduysanız config dosyasında bunu ya hiç yazmazsınız ya da `${ELASTICSEARCH_PASSWORD}` gibi bir referans bırakırsınız. Kibana bu ifadeyi okuduğunda önce kendi keystore'una bakar, bulamazsa ortam değişkenlerine bakar [31]. Elasticsearch keystore ise config dosyasında özellikle `secret.name: ${keystore:mysecret}` gibi bir sözdizimini destekler.

Ortam değişkenleri (Environment variables) de bir diğer yöntemdir. Kibana ve Logstash, configte `${VAR_NAME}` şeklinde yazılan yerleri ortam değişkenleriyle doldurabilir [31] [33]. Örneğin Kibana'da `elasticsearch.password: ${ES_PWD}` yazdıysanız ve Kibana servisinin ortamında `ES_PWD=abc123` tanımlıysa, Kibana bununla bağlanır. Ancak ortam değişkenleri genelde sistemde düz okunabilir durumdadır (örn. `ps` ile argument olarak gözüken yerlere yazılmaz ama /proc ortamında root tarafından görülebilir). Yine de container ortamlarında vs. sık kullanılır.

> **Neden:** *Configuration as Code* prensibiyle yönetilen sistemlerde, hassas verilerde şeffaflık istenmez. Örneğin bir git deposunda kibana.yml tutuluyorsa, içinde açıkta şifre bulunması güvenlik riski. Bu gibi durumlarda, şifreyi ayrı bir yol ile enjekte etmek gerekir. Kibana ve Logstash'ın keystore mekanizmaları tam da bu amaçla geliştirilmiştir [20] [21]. Ayrıca devops süreçlerinde CI/CD hatlarında, bu keystore'ları önceden oluşturup sunuculara dağıtabilir veya container imajına sadece vault'dan çekeceğiniz secret'ları ortam değişkeni ile set edebilirsiniz. **Özetle, keystore ve env kullanımı güvenlik ve otomasyon kolaylığı sağlar.**
>
> **Doğrulama:** Kibana keystore'daki bir değerin doğru yüklendiğini test etmek için basitçe Kibana'yı yeniden başlatıp fonksiyonelliğe bakmak gerekir. Örneğin `elasticsearch.password` keystore'dan çekilmiyorsa Kibana Elastic'e bağlanamaz ve logunda "authentication failed" görebilirsiniz. Bu durumda muhtemel hata, keystore'a yanlış kullanıcı şifresi girmektir. `kibana-keystore list` ile değer ismine bakın, doğru yerde kullanıldığına emin olun. Aynı şekilde Logstash'ta da birazdan göreceğimiz üzere, keystore'dan okunan bir credential için hata alırsanız Logstash pipeline o alanı boş geçmiş olabilir, bağlanma hatası oluşur. Böyle durumlarda keystore'daki veriyi silip yeniden eklemek veya doğru referansla kullanmak çözüm olacaktır.

Logstash Kurulumu ve Veri Pipelineları

Logstash, çeşitli kaynaklardan gelen log ve etkinlik verilerini toplayıp işleyerek Elasticsearch'e aktarmamızı sağlayan güçlü bir ardışık işlem (pipeline) motorudur. Agentless mimaride, logların büyük kısmı doğrudan ağ üzerinden (örneğin syslog protokolü, Beats protokolü vb.) Logstash'e gelir. Bu bölümde Logstash'ın kurulumu, güvenli iletişim için TLS ayarlarıyla giriş (input) tanımlanması, verilerin Elasticsearch'e gönderilmesi için çıkış (output) yapılandırması, **Logstash keystore** kullanımı ve **ECS uyumlu** örnek pipeline kuralları ele alınacaktır. Ayrıca Beats, Syslog ve Windows Event Forwarding (WEF) için ayrı ayrı pipeline örnekleri verilecektir.

Logstash Paketinin Kurulumu (DEB)

Benzer şekilde, Logstash da Elastic apt deposundan kurulabilir:

```
sudo apt-get update && sudo apt-get install logstash
```

Bu, Logstash'ı `/etc/logstash` konfigürasyon dizini ile birlikte yükleyecektir. Logstash, Java ile yazılmıştır ve bir systemd servisi olarak çalıştırılabilir. Kurulum sonrasında hemen servisi başlatmadan önce pipeline tanımlarımızı yapmak iyi bir fikirdir.

Logstash Temel Yapı ve Ayarlar

Logstash yapılandırması üç ana bileşenden oluşur: **input**, **filter**, **output**. Bu bileşenler `logstash.conf` ya da `/etc/logstash/conf.d/` altındaki birden fazla .conf dosyasında belirtilebilir. Her bileşenin bir veya birden fazla eklenti (plugin) kullanarak alt yapılandırmaları olur.

Örneğin basit bir yapı:

```
input {
  tcp {
    port => 5000
    codec => json
    ssl_enable => true
    ssl_cert => "/etc/logstash/certs/logstash.crt"
    ssl_key => "/etc/logstash/certs/logstash.key"
  }
}
filter {
  # ... (istediğimiz filtreler)
}
output {
  elasticsearch {
    hosts => ["https://127.0.0.1:9200"]
    index => "logs-%{[@metadata][beat]}-%{+YYYY.MM.dd}"
    user => "elastic"
    password => "${ES_PWD}"
    cacert => "/etc/logstash/certs/CA.crt"
  }
}
```

Yukarıdaki örneği adım adım açarsak: - **input/tcp:** TCP üzerinden 5000 portunu dinliyor, gelen verinin JSON formatında olduğunu varsayıyor ve TLS ile şifreleme yapıyor (`ssl_enable => true`). Bu, örneğin bir uygulamanın JSON loglarını doğrudan TCP üzerinden gönderdiği bir durum için olabilir. Burada TLS için Logstash'ın kendi sertifikasını (server sertifikası) ve özel anahtarını belirttik. `logstash.crt` ve `logstash.key` dosyalarını önceden oluşturmuş olmamız gerekir (bizim daha önceki CA ile imzalayarak yaratmamız mümkün). Bu sayede istemci taraf (log gönderen) Logstash'ın kimliğini doğrulayabilir. İstemciden de kimlik doğrulaması istenebilir, o durumda `ssl_verify_mode => "force_peer"` ve `ssl_cert` / `ssl_key` istemci tarafında da gerekir; fakat genelde Logstash input'larında sunucu kimliği yeterli olur. - **filter:** Filtre kısmı logları dönüştürmek, parse etmek, zenginleştirmek için kullanılır. Bu örnekte filter boş, ama ilerleyen alt başlıklarda ECS uyumu için neler yapacağımıza bakacağız. - **output/elasticsearch:** Elasticsearch'e çıkış yapıyoruz. `hosts` olarak TLS ile 127.0.0.1:9200 verildi. `index` ayarında, gelen verinin türüne göre bir indeks

adlandırması yapılmış (metadata'de beat ismi varsa gibi). Biz data stream kullanacaksak index yerine farklı bir yaklaşım olacak, bunu ayrıca ele alacağız. `user` ve `password` ile Elastic'e bağlanmak için kullanıcı kimlik bilgisi verdik. Burada **şiddetle tavsiye edilen**, `elastic` süper kullanıcısını *kullanmamaktır*. Bunun yerine sadece ingest yetkisi olan bir kullanıcı oluşturup onu kullanmalıyız. Fakat lab ortamında basitlik adına `elastic` kullandık, production için ayrı bir kullanıcı yaratma konusuna kısaca değiniriz. Parolayı da doğrudan yazmak yerine Logstash keystore'dan çekiyoruz (`${ES_PWD}` ifadesi). Logstash, config dosyası parse edilirken `${...}` gördüğünde, önce keystore'a bakar, bulamazsa environment variable'a bakar [34] [35]. Bir önceki bölümde biz CA'mizi kibana için kopyalarken bir de `/etc/logstash/certs/CA.crt` koymuştuk. `cacert` ayarı ile, Logstash'in Elasticsearch'e bağlanırken bu CA'ya güvenmesini sağladık. Aksi halde kendi imzaladığımız sertifika nedeniyle "certificate verify failed" hatası alır.

Şimdi bu temel yapıyı bizim kullanım senaryolarımıza uyduralım.

TLS ile Güvenli Log Alma (Beats ve Syslog İçin Input Ayarları)

1. Beats input (Filebeat/Winlogbeat): Beats, Elastic'in hafif ajan programlarıdır (Filebeat, Winlogbeat, Metricbeat vs.). Agentless mimaride "Elastic Agent" kullanılmıyor dedik, ama klasik Beats'ler doğrudan Logstash'e veya Elasticsearch'e veri gönderebilir. Burada senaryomuz: Uzak sistemlere Filebeat/Winlogbeat kurup onları Logstash'e yönlendirmek. Logstash'te bunun için özel bir **beats input plugin** vardır.

Beats input, Beats protokolünü (çoklu veri akışı ve doğrulama içerir) konuşur ve default portu 5044'tür. Şifreli iletişim için Beats, sunucu tarafı sertifikasını doğrulayabilir ve hatta client sertifikası ile kimlik sunabilir. Biz sunucu tarafı TLS yapacağız.

Logstash konfigürasyonumuza bir beats input ekleyelim:

```
input {
  beats {
    port => 5044
    host => "0.0.0.0"
    ssl => true
    ssl_certificate => "/etc/logstash/certs/logstash.crt"
    ssl_certificate_authorities => ["/etc/logstash/certs/CA.crt"]
    ssl_key => "/etc/logstash/certs/logstash.key"
    # ssl_verify_mode => "force_peer"  # (İstemciden de sertifika
isteyeceksek açılır)
  }
}
```

Açıklayalım: - `host => "0.0.0.0"` demek tüm arayüzleri dinle, isterseniz sadece elk.local ya da internal IP yazabilirdiniz. - `ssl => true` ve sertifika/anahtar parametreleri bizim Logstash'ın sunucu sertifikasını tanımlıyor. Bu sertifika muhtemelen elk.local adına olmalı ve CA.crt ile Beats tarafında güvenilecektir. - `ssl_certificate_authorities` parametresi ise **istemci sertifikası doğrulaması** yapacaksanız kullanılır (Logstash istemcinin sertifika otoritesini bilir ve ona göre istemcinin certini doğrular). Yukarıda satırı yorum yaptık çünkü Beats ajanlarının sertifikayla kimlik sunmasını şu an zorunlu kılmadık. İsterseniz Filebeat tarafında client sert. ayarlayıp burada da verify'i aktif edebilirsiniz, ama genelde fazla kullanımı yok – çoğu senaryoda Beats ile Logstash arasındaki TLS tek yönlüdür.

Bu ayarlarla Logstash 5044 üzerinde Beats beklemeye hazır olacaktır.

Beats tarafı ayarlar: Filebeat veya Winlogbeat konfigürasyonunda, Logstash hedefini belirtirken SSL'yi açmalı ve CA sertifikasını sunmalısınız. Örneğin Filebeat `output.logstash` bölümünde:

```
output.logstash:
  hosts: ["elk.local:5044"]
  ssl.certificate_authorities: ["/etc/filebeat/certs/CA.crt"]
```

Bu şekilde Filebeat, Logstash'ın sunucu sertifikasını bizim CA ile doğrular. Eğer istemci taraf kimlik doğrulaması yapacaksak Filebeat'e kendi sertifikasını da verdirtip Logstash'ta `ssl_verify_mode => "force_peer"` açmamız gerekirdi.

2. Syslog input: Birçok ağ cihazı veya Unix sistem, loglarını syslog protokolüyle UDP/514 veya TCP/514 üzerinden iletir. Logstash, syslog verisini almanın birkaç yoluna sahip: - `syslog` input plugin'i (udp ve tcp dinleyebilir, ve RFC3164 formatını parse edebilir). - `tcp` veya `udp` plugin + `syslog` codec kullanmak. - Dışarıda bir syslog toplayıcı (rsyslog, syslog-ng) kullanıp dosyaya yazmak, Filebeat ile okumak (agentless amacımıza ters gerçi). - WEF gibi kaynaklar için özel bir input yok, onlara ayrıca değineceğiz.

Logstash Syslog input plugin ne yazık ki doğrudan TLS desteği içermez (soru-cevaplarda çıktığı üzere) [36] [37]. Bu durumda, TLS ile syslog almak isterseniz iki seçenek var: - Syslog akışını stunnel, haproxy gibi harici bir TLS terminator ile Logstash'a iletmek. - Veya Logstash'ta `tcp` input kullanıp gelen baytları syslog formatında parse edecek bir filtre uygulamak.

Kolaylık açısından ikinci yolu seçebiliriz. Örneğin UDP syslog TLS desteklemediği için onu belki plain alabiliriz (iç LAN ise belki kabul edilebilir), ama TCP syslog TLS gerekli diyelim.

Basitçe:

```
input {
  tcp {
    port => 6514
    type => "syslog"
    ssl_enable => true
    ssl_cert => "/etc/logstash/certs/logstash.crt"
    ssl_key => "/etc/logstash/certs/logstash.key"
    ssl_verify => false
  }
  udp {
    port => 514
    type => "syslog"
  }
}
```

6514, TLS üzerinden syslog için IANA tarafından ayrılmış porttur. Yukarıda tcp input, 6514'ü TLS ile dinliyor, udp input 514'ü plain dinliyor. Her ikisine de `type => "syslog"` etiketi koyduk ki filter aşamasında bunları ayırt edelim.

Bu ham syslog mesajlarını parse etmek için filter tarafında hazır bir **syslog parser** kullanabiliriz: **grok** veya **dissect** ile kendimiz şablon yazabiliriz ya da Logstash'ın **syslog_pri** filtresini kullanabiliriz.

ECS uyumu için belki birden fazla aşama gerekecek, birazdan ayrıntılandırırız. Ama kabaca, bir syslog mesaj formatı:

```
<34>Oct 11 22:14:15 host process[123]: Message details...
```

gibidir. Bunu grok ile parçalamak istersek:

```
filter {
  grok {
    match => { "message" => "<%{INT:syslog.priority}>%
{SYSLOGTIMESTAMP:syslog.timestamp} %{HOSTNAME:syslog.host} %
{DATA:syslog.program}(?:\[%{INT:syslog.pid}\])?: %
{GREEDYDATA:syslog.message}" }
  }
  date {
    match => [ "[syslog][timestamp]", "MMM dd HH:mm:ss" ]
    remove_field => "[syslog][timestamp]"
  }
}
```

Bu örnek bir şablondur. grok ile syslog'un ön-eklerini parse ettik ve `syslog` isimli alan altında alt alanlara koyduk. Sonra date filter ile timestamp'ı gerçek zaman damgasına çevirdik. Bu alanları ECS'ye uydurmak için mapping yapabiliriz: - syslog.host aslında event'in orijinal kaynağı, ECS'de `host.hostname` veya `host.name` olarak konabilir. - syslog.program -> `process.name` ya da ECS'de `process.name` ifadesine (ya da `log.source.application` gibi bir alan yok ECS'de, process demek doğru). - syslog.pid -> `process.pid` - syslog.message -> esas log içeriği belki `message` alanına veya ECS'de `log.original` alanına (orijinal ham log için) konabilir.

Bu dönüşümleri yapmak için **mutate** filtresi kullanarak alanları yeniden adlandırabilir veya ekleyebiliriz:

```
filter {
  # ... (yukarıdaki grok ve date)
  mutate {
    rename => [ "[syslog][host]", "[host][hostname]" ]
    rename => [ "[syslog][program]", "[process][name]" ]
    rename => [ "[syslog][pid]", "[process][pid]" ]
    rename => [ "[syslog][message]", "message" ]
  }
  # event.original saklamak istersek orijinal message'ı kopyalayabiliriz.
  mutate {
    add_field => { "event.original" => "%{message}" }
  }
}
```

Yukarıda örnek olarak yaptık. ECS'de `host.name` alanı da kullanılıyor olabilir hostname için; burada hostname aldık, belki domain ya da IP yok, uygun bulduğumuz şekilde atadık.

3. Windows Event Forwarding (WEF) ve Winlogbeat: WEF, Windows ortamında istemci makinelerin (domain joined) belirli event'ları abonelik yoluyla bir kolektöre göndermesidir. Kolektör sunucu, bu forwarded event'ları kendi Event Viewer'ında "Forwarded Events" altında toplar. Buradan logların Elastic'e taşınması için genelde Windows tarafında bir ajan gerekir. **Winlogbeat**, Windows Event Log API üzerinden istenen logları okuyup iletebilen bir Beats aracıdır.

Agentless olarak burada belki WEF ile toplanan event'ları direkt Logstash'a syslog gibi atma opsiyonu yok (Windows Event'ları sysloga çevirmek ekstra bir yazılım gerektirir). En pratik çözüm: WEF kolektör makinesine Winlogbeat kurup, Logstash'a gönderim yapmak.

Dolayısıyla, WEF de aslında bir Beats senaryosu haline gelir. Winlogbeat içindeki `winlogbeat.yml` config'de:

```
winlogbeat.event_logs:
  - name: ForwardedEvents
output.logstash:
  hosts: ["elk.local:5044"]
  ssl.certificate_authorities: ["C:/ProgramData/Winlogbeat/ca.crt"]
```

gibi bir ayar, WEF ile gelen event'ları alıp Logstash'a yollar. Bizim Logstash pipeline'ımız Beats input'unu zaten hazırlamıştı. Winlogbeat gönderdiği veriyi ECS uyumlu olarak zaten JSON içerik şeklinde yollayacak (wineventlog alanlarını ECS'ye uygun şekilde). Bu nedenle Logstash'ta Beats için genelde *filtre yazmaya gerek kalmaz*, direk output'a göndeririz.

Örneğin Winlogbeat, her event için `@timestamp`, `host.name`, `winlog.channel`, `event.id`, `message` gibi alanları zaten ECS şemasında yazar. SIEM kural seti de Winlogbeat'ten gelen event'ları doğru yorumlayabilir. Dolayısıyla Logstash burada sadece iletim görevi görür.

Logstash Keystore Kullanımı (Elasticsearch Çıkışı için): Yukarıdaki pipeline config'lerinde fark ettiyseniz `user` ve `password` verdik. Parolayı Logstash config'ine düz yazmıyoruz, onun yerine `ES_PWD` adında bir değişken kullandık. Bunu ya ortam değişkeni ya da keystore ile sağlamalıyız. Daha güvenli olan keystore'dur.

Logstash keystore dosyasını oluşturalım ve elasticsearch parolasını ekleyelim:

```
sudo /usr/share/logstash/bin/logstash-keystore create
sudo /usr/share/logstash/bin/logstash-keystore add ES_PWD
```

İkinci komut sizden bir değer girmenizi isteyecek, `elastic` kullanıcısının şifresini girin (veya üretimde ayrı kullanıcı oluşturduysanız onun şifresini). Enter'ladığınızda keystore'a eklenecektir [38] .

Bu keystore'un Logstash tarafından okunabilmesi için doğru yerde olması gerekiyor (genelde /etc/logstash içindedir ve yaratırken path.settings doğru ise otomatik oraya koyar). Ayrıca `/etc/logstash/startup.options` içinde keystore şifresi parametresi vs. yoksa, keystore'u şifresiz bırakmak ya da environment'da LOGSTASH_KEYSTORE_PASS vermek gerekebilir. Biz şifresiz bıraktık (enter enter geçtik).

Artık `logstash.conf` içindeki `${ES_PWD}` ifadesi, Logstash tarafından keystore'dan çözülüp Elasticsearch'e bağlanmakta kullanılacaktır [39] [35].

Logstash'i Başlatma ve Servis Yapılandırması: Pipeline dosyalarımızı `/etc/logstash/conf.d/` içine yerleştirdik diyelim (ör. `01-inputs.conf`, `10-filters.conf`, `90-outputs.conf` gibi). Logstash varsayılan olarak bu dizindeki tüm .conf dosyalarını sıralı biçimde yükler. Alternatif olarak tek bir dosyada da tanımlayabilirsiniz.

`/etc/logstash/logstash.yml` içinde pipeline ile ilgili ayarlar vardır, ancak genelde oraya dokunmadan çalışır. (Örneğin `pipeline.workers`, `pipeline.batch.size` vs. ayarları var, default'ları yeterlidir.)

Logstash'i systemd ile başlatmak için:

```
sudo systemctl enable logstash
sudo systemctl start logstash
```

komutlarını kullanın. Anlık logları:

```
sudo journalctl -u logstash -f
```

ile takip edebilirsiniz. İlk çalışmada, Logstash pipeline derlemesi yapar, "Pipeline started" mesajı görürseniz her şey yolunda demektir.

Logstash, Beats ve Syslog portlarımızı dinlemeye başladıktan sonra, artık dış sistemlerimizi bu Logstash'a bağlayabiliriz demektir. Filebeat/Winlogbeat kurulu sistemleri 5044 TLS portuna yönlendirdik, onlar bağlanınca log akışı gelecektir. Aynı şekilde ağ cihazları/syslog kaynakları 514 veya 6514 portuna log yollamaya başladığında Logstash bunları alacak, parse edecek ve Elasticsearch'e gönderecektir.

Output tarafında **data_stream** kullanımına değinmedik. Bu, Elastic 7.10+ ile gelen modern indeksleme yaklaşımlarından biridir. Data streams, ILM (yaşam döngüsü) yönetimini kolaylaştırır ve indeks kalıplarını standardize eder. Biz klasik index pattern yaklaşımıyla örnek verdik (`logs-...-%{+YYYY.MM.dd}`). Ancak SIEM ürününde genelde data stream kullanılır (özellikle Fleet ile gelen entegrasyonlarda).

Logstash ile Data Stream kullanmak için, Elasticsearch output plugin'inde `data_stream => true` parametresini açmak gerekir [40]. Ayrıca `data_stream_type`, `data_stream_dataset`, `data_stream_namespace` gibi ayarlar yapmak gerekir (ve Elasticsearch tarafında bu data stream için bir index template mevcut olmalı). Örneğin:

```
output {
  elasticsearch {
    hosts => "https://127.0.0.1:9200"
    user => "elastic"
    password => "${ES_PWD}"
    cacert => "/etc/logstash/certs/CA.crt"
    data_stream => true
```

```
    data_stream_type => "logs"
    data_stream_dataset => "%{[@metadata][type]}"
    data_stream_namespace => "default"
  }
}
```

Bu yapı, Logstash'ın veriyi `logs-<dataset>-default` adlı bir data stream'e göndermesini sağlar. Dataset değerini biz genelde `[@metadata][type]` veya benzer bir meta alandan alabiliriz (örneğin Beats, kendi tipini meta olarak iletir). Data stream kullanıldığında Elasticsearch, index isimlerini kendi yönetir ve ILM rolloever'ları otomatik olur. Tabii bunun öncesinde ilgili `logs-*-default` için bir index template ve ILM policy tanımlı olmalıdır. Kılavuzun ilerleyen ILM bölümünde buna değineceğiz.

Prod ortamında data stream tercih edilir çünkü SIEM app, detection vb. hepsi artık data stream kullanan hazır entegrasyonlara göre optimize edilmiştir. Ancak manuel kurulumda klasik indexler de iş görecektir.

Özet olarak, Logstash artık verileri alıp Elasticsearch'e güvenli şekilde basabilecek durumda. Sırada, Elasticsearch tarafında bu gelen logların yönetimi (ILM, index templates) ve SIEM kurallarının oluşturulması var.

Doğrulama: Logstash çalıştıktan sonra, Filebeat ve Winlogbeat kurulu sistemlerdeki loglar Elasticsearch'e akmaya başlayacaktır. Bunu sınamak için Kibana'da **Discover** sekmesine gidin, veri görüntülenmiyorsa bir **Data View** oluşturmanız gerekebilir (ör. `logs-*` pattern'ine uyan bir data view). Ardından logların düşüp düşmediğini kontrol edin. Eğer veriler gelmiyorsa: - Logstash loglarında hata var mı bakın (özellikle Elasticsearch output ile ilgili authentication veya index izin hataları olabilir). - Beats tarafında `filebeat.exe -e` (Windows) ya da `sudo filebeat -e` (Linux) ile çalıştırıp hata mesajlarına bakın. Sertifika doğrulama hatası varsa, belki CA dosyasını yanlış belirttiniz veya Logstash'ın sertifikasını Beats'e trust ettirmediniz. Gerekirse `ssl.verification_mode: none` ile test amaçlı güvenliği esnetip sorunu izole edebilirsiniz. - Syslog gönderip göndermediğini cihazlar üzerinden takip edin. Netcat ile elle test yapabilirsiniz:

```
echo "<34>$(date '+%b %d %H:%M:
%S') testhost testprog[123]: Deneme log" | nc -u 127.0.0.1 514
```

Bu komutu Logstash sunucusunda çalıştırınca, Discover'da `Deneme log` içeren entry görmelisiniz. Görmüyorsanız ya Logstash udp input yoktur ya filter yanlış parse ediyordur.

Doğrulamanın bir diğer yöntemi de **Kibana SIEM Alarmlarını** test etmektir ama onu ilerleyen bölümde yapacağız.

Geri Alma: Logstash konfigürasyon değişiklikleri anında uygulanmaz; pipeline'ı değiştirdiğinizde servisi restart etmeniz gerekir. Eğer yeni config hata içeriyorsa Logstash başlamayabilir veya eski pipeline ile devam edebilir (config testini geçemezse). Bu durumda `sudo /usr/share/logstash/bin/logstash --config.test_and_exit -f /etc/logstash/conf.d` komutu ile config'lerin sentaks testini yapabilirsiniz. Hata mesajı size hangi satırda sorun olduğunu söyleyecektir.

Logstash input portları conflict edebilir; örneğin 514 portunu root olmadan dinleyemezsiniz (Logstash kullanıcısı açamaz). Çözüm: ya Logstash'i 514 için yetkiyle çalıştırmak (güvensiz) ya da 514 yerine 1514 gibi bir port kullanmak. Biz 514 UDP'yi root:logstash grubuyla CAP_NET_BIND servis capability ayarı olmadan açamayız. Böyle durumlarda portu yükseltmek en kolayıdır.

Son olarak, eğer Logstash ile uğraşmak istemeyenler için (tamamen agentless'tan vazgeçip), Beats'leri doğrudan Elasticsearch'e göndermek bir geri adım olabilir. Ancak o zaman log transform ve ECS'de tutarsızlık yaşanabilir. Bu nedenle sorun çıksa da Logstash hattını düzeltmeye çalışmak uzun vadede daha tutarlıdır.

Logstash Pipeline Örnekleri ve ECS Uyumlu Alanlar

Yukarıda Beats için filtre gerekmediğini, syslog için grok ile parse edip ECS alanlarına çevirmeyi örnekledik. Bu alt bölümde birkaç pratik örnek daha verelim:

- **Linux Syslog (auth.log veya messages):** Diyelim ki Filebeat yerine tüm Linux sistemleri loglarını merkezi syslog ile bize atıyor. Bu loglar içinde `/var/log/auth.log` satırları da olacak. ECS'de bunları classifiye etmek istersek, örneğin SSH başarısız login denemesi logu yakaladık diyelim, detection kuralı `event.category: authentication and event.outcome: failure` gibi bir filtre bekler. Bizim syslog düz string bunu barındırmaz. Fakat belki grok ile yakalarsak `Invalid password for user %{USER:user.name}` gibi bir desen, buna bir event.category = authentication atayabiliriz (mutate add_field ile). Bu tarz zenginleştirmeler, Logstash pipeline'ında custom olarak yapılabilir. ECS'nin tavsiye ettiği alanlar belgelidir, isterseniz her bir event türü için özel çaba harcanabilir. Bu, büyük emek gerektirebilir ama tam SIEM uyumu sağlar. Kısıtlı zamanda, sadece temel alanları eşleştirmekle yetinilebilir.

- **Windows Event ECS Uyum:** Winlogbeat verileri zaten ECS uyumlu gelir dedik. Örneğin Winlogbeat bir log gönderdiğinde `event.code: 4625` (Windows login başarısız), `event.category: authentication` ve `event.outcome: failure` gibi alanlar otomatik set olur. Dolayısıyla SIEM rule'ları (ki birazdan bakacağız) bu alanları kullanarak tetiklenir. Eğer Winlogbeat kullanmadan WEF loglarını düz bir metin olarak alsaydık çok zorlanırdık. (Örneğin XML formatlı eventları parse edip ECS'ye koymak epey uğraş). Bu yüzden Windows tarafında Beats kullanmak agentless konseptini biraz esnetse de mantıklıdır.

Özet: Logstash pipeline'larımız çalışır hale geldiğinde, ECS uyumuna dair yapabildiğimizin en iyisini yapmalıyız. Basit bir kontrol: Kibana Discover'da logları incelerken, kayda tıklayıp JSON görünümüne bakın. Eğer alan isimleri ECS'de tanımlı şekilde geliyorsa (host.name, process.name, log.source.address vs.) kendimizi tebrik edebiliriz. Eğer her şey message içinde tek bir string ise, SIEM katmanında pek işe yaramaz. Bu durumda parse filtrelerimizi geliştirmemiz gerekir.

Index Lifecycle Management (ILM) ve Index Template Ayarları

Log verileri sürekli biriktiği için, zamanla eski verilerin arşivlenmesi veya silinmesi gerekebilir. Elastic Stack, **Index Lifecycle Management (ILM)** özelliği ile indekslerin yaşam döngüsünü otomatik yönetmeyi sağlar. Ayrıca, log verilerinin indekslendiği şemayı ve ayarları önceden tanımlamak için **index template (indeks şablonu)** kullanırız. Bu bölümde, log verileri için bir ILM politikası oluşturacak ve `logs-*` adında indekslerimize uygulayacağız. Ardından, bu politikayı kullanan bir index template örneği göstereceğiz. Bu sayede yeni gelen loglar otomatik olarak bu şablona uygun indekslere yerleşecek.

ILM Politikası Oluşturma

Öncelikle bir yaşam döngüsü politikası tanımlayalım. Örneğin loglarımızın **30 gün sıcak (hot)** fazda kaldıktan sonra **soğuk (cold)** veya **silme (delete)** fazına geçmesini isteyebiliriz. Tek sunucu ortamında cold faz fiziksel ayrım yapmadığı için belki direkt silme uygularız. Diyelim ki verileri 90 gün saklamak istiyoruz ve her gün yeni bir indeks (veya data stream backing index) oluşsun.

Kibana arayüzünden ILM politikası oluşturmak için Stack Management > Index Lifecycle Policies kısmını kullanabilirsiniz. Biz burada API ile yapalım (Kibana Dev Tools > Console):

```
PUT /_ilm/policy/logs_policy
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "0ms",
        "actions": {
          "rollover": {
            "max_size": "50GB",
            "max_age": "30d"
          },
          "set_priority": {
            "priority": 100
          }
        }
      },
      "delete": {
        "min_age": "90d",
        "actions": {
          "delete": {}
        }
      }
    }
  }
}
```

Bu komut, Elasticsearch'e `logs_policy` adında bir ILM politikası kaydeder. Anlamı: - Indeks en geç 30 günde bir veya 50GB büyüklüğe ulaşınca rollover (yeni indeks aç) yap. - Hot fazda 0ms bekle yani hemen kullan. - 90 gün sonra (hot + belki warm vs. atlayıp) delete fazına geç ve sil.

Warm veya cold faz eklemedik. Tek node'da zaten tüm veriler hot'ta duracak, belki silmeden önce warm/cold ayrımı yapmaya gerek yok.

Index Template Tanımlama (logs-*)

Sırada index template var. Elasticsearch 8.x ile birlikte **component template** ve **index template** kavramları geldi. Biz tek bir şablon basitçe oluşturacağız. Bu şablon, `logs-*` ile başlayan her indeks veya data stream yaratılırken devreye girecek. İçinde: - Ayarlar (settings): ILM politikası ataması, gerektiğinde number_of_shards (1) vs. - Alan haritaları (mappings): ECS alanları için uygun tipler,

dynamic mapping ayarları vb. - Alias veya data stream tanımı: eğer data stream kullanıyorsak burada belirtilir.

Önce bir örnek verelim. Data stream kullanıyorsak index template şöyle olur:

```
PUT /_index_template/logs_template
{
  "index_patterns": ["logs-*"],
  "data_stream": { },
  "template": {
    "settings": {
      "index.lifecycle.name": "logs_policy",
      "index.lifecycle.rollover_alias": "logs-default"
    },
    "mappings": {
      "_source": { "enabled": true },
      "properties": {
        "@timestamp": { "type": "date" },
        "message": { "type": "text" },
        "host": {
          "properties": {
            "name": { "type": "keyword" }
          }
        },
        "process": {
          "properties": {
            "name": { "type": "keyword" },
            "pid": { "type": "long" }
          }
        },
        "user": {
          "properties": {
            "name": { "type": "keyword" }
          }
        },
        "event": {
          "properties": {
            "original": { "type": "text", "index": false },
            "category": { "type": "keyword" },
            "outcome": { "type": "keyword" }
          }
        }
        /* ... burada ihtiyaç duyulan tüm ECS alanlarını ekleyebiliriz ... */
      }
    },
    "aliases": {
      "logs-default": {
        "is_write_index": true
      }
    }
```

```
  }
}
```

Bu oldukça kapsamlı bir örnek. Parça parça bakarsak: - `index_patterns`: logs-* gelen her indeks ismine uyar. - `data_stream`: boş bir obje belirtmek, bu şablonun bir data stream için olduğunu gösterir. (Data stream olduğunda rollover alias otomatik data stream adı olur normalde, ama gene de alias ekledik.) - Settings içinde ILM politikamızı bağladık ve rollover alias verdik (`logs-default`). Bu alias data stream'in write alias'ı olacak. - Mappings'de en temel alanları tanımladık. Elasticsearch dynamic mapping ile kendisi de tip atayabilir, ama kritik alanlar için (özellikle `@timestamp`) biz belirttik. ECS alanları genelde keyword veya date vs. Burada gösterim amaçlı birkaç tanesini ekledik. Real world'de ECS'nin tam şemasını incorporate etmek istersek Elastic'in sağladığı ECS template'lerini kullanabiliriz (ör. Beats'in index template'leri hazır gelir). - Aliases bölümünde `logs-default` adında alias ekledik ve yeni yazılacak indeks olarak işaretledik. Data stream olmasa da rollover alias bu şekilde devreye girer. Data stream modunda buna belki gerek yok, ama zarar da vermez.

Bu template'i oluşturduktan sonra: - Eğer **data stream** modunda ingest yapıyorsak, bir data stream yaratmak lazım:

```
POST /_data_stream/logs-default
```

Bu, logs-default adında bir data stream ve arkasında ilk backing index (logs-default-000001) oluşturur. Bu index template'imiz uygulandığı için ILM vs. hazır olur. Logstash output'unda `data_stream.dataset => <birşey>` demiş olmalıydık. Biz template'i logs-* patternine yazdık ama data stream ismi logs-default ise belki dataset "default" olmalı. Bu detaylara dikkat etmek gerek.

- Eğer **klasik indeks** modunda kalıyorsak (her gün logs-2023.01.01 gibi açmak), template'i biraz farklı yazarız:
- `data_stream` kısmını koymayız.
- Alias kısmını gene koyarız (rollover alias).
- Indeks ilk oluşumunda alias'ı kendine atar, ILM rollover tetiklenince alias yeni indekse geçer. (Bu klasik rollover mekanizması.)

Mesela data stream kullanmadan:

```
"index_patterns": ["logs-*"],
"template": {
  "settings": {
    "index.lifecycle.name": "logs_policy",
    "index.lifecycle.rollover_alias": "logs-index"
  },
  ...
  "aliases": {
    "logs-index": { }
  }
}
```

İlk indeks ismini logs-000001 gibi verirseniz (alias ile aynı prefix olmalı ILM öyle ister), ILM çalışır. Bu biraz karmaşık gelebilir, ama nihai amaç: ILM, alias üzerinden aktif indeksi bilir ve rollover yapar.

Sonuç: Bu yapıyı kurduktan sonra, gelen loglar ya data stream'e gidiyordur (Logstash output ona göre ayarlıysa) ya da direkt index'e. Hangisi olsun karar verin, ikisi de olur. Data stream güncel yöntem olduğu için belki onu tercih edebilirsiniz.

Kibana tarafında, data view oluştururken logs-* patterni her ikisini de kapsar (data stream de sonuçta logs-default isminde alias), arka planda halleder.

> **Neden:** ILM kullanımı, log verisinin yönetimini otomatikleştirir ve disk tüketimini kontrol altında tutar [41]. Günlük indeks yaklaşımı klasik olsa da elle silme veya cron yazmak gerekirdi; ILM ile "90 gün sonra sil" demek yeterli. Data stream ise ILM ile entegre çalışarak veri girişi kesintisiz devam ederken arka planda indeks değişimini sorunsuz kılar. Bu mimari, Elastic Security (SIEM) uygulamasının beklediği standartları da karşılar çünkü resmi kural setleri genelde default data stream'lere göre ayarlanmıştır. Biz manuel ayar yaparken bu standartlara mümkün mertebe uymaya çalıştık.
>
> Template tanımlamak ise, log alanlarının tutarlı tiplerde olmasını sağlar. Örneğin host.name her yerde keyword ise arama yaparken sorun yaşamayız. ECS'ye uygun mapping, ileride makine öğrenimi veya kural yazarken büyük kolaylık sağlar.
>
> **Doğrulama:** Kibana'da Index Management bölümüne girerek logs-* indekslerini ve onların ILM durumlarını görebilirsiniz. Yeni gelen log ile belki logs-default-000001 indeksi oluşur (veya logs-2023.10.06 gibi). Bu indeksin "ILM policy" alanında logs_policy bağlı mı kontrol edin. Ayrıca Stack Management > Index Lifecycle Policies altında logs_policy'ye tıklayarak kaç indeks uygulandığını görebilirsiniz.
>
> Template'i test etmek için yeni bir indeks simüle edin:
>
> ```
> POST /logs-test/_doc
> { "@timestamp": "2025-10-06T12:00:00Z", "message": "test", "host":
> {"name": "testhost"} }
> ```
>
> Bu, logs-test adında bir indeks oluşturacak. Bizim template logs-*'ı kapsadığı için logs-test bu şablonu kullanacaktır. Sonrasında
>
> ```
> GET logs-test/_mapping
> ```
>
> çıktısında `@timestamp`'ın date tipinde olduğunu, ILM politikasının atandığını vb. görüyorsanız template başarılıdır.
>
> Bir de ILM rollover'ı test edelim:
>
> ```
> POST /logs-default/_rollover
> ```
>
> (ya da alias isminiz neyse). Bu komut elle rollover tetikler. Yanıt olarak "Rolled over" true gelirse ikinci indeks oluşmuştur. Index Management'ten bakın, logs-default-000002 oluştu mu? Oluştuysa alias 000002'ye geçti mi? Bu test, ILM'in ileride kendi tetikleyeceği işlemi doğrular.

Geri Alma: Yanlış bir ILM politikası uyguladıysanız (ör. min_age değerini yanlış koydunuz, indeks hemen silinmeye gitti vb.), paniğe kapılmayın. ILM, silme fazına geldiğinde gene de emniyet amaçlı `index.lifecycle.delete_phase_time` alanını indekse yazar, siz istemezseniz silemeyebilir. Ancak oldu da indeksler beklenenden önce kapandıysa (mesela sıcak fazda rollover habire tetiklendi) ILM politikasını düzenleyip etkilenen indekslere tekrar uygulatabilirsiniz.

Index template ile ilgili sorunlarda (mapping hatası vb.), ilgili indeksi silip (tabii veri kaybı olabilir test ortamı değilse dikkat), template'i düzelterek yeniden oluşturma yoluna gidebilirsiniz. Template değişikliği mevcut indekslere etki etmez, sadece yeni indeks yaratıldığında devreye girer. Dolayısıyla mapping hatası yaptığınızı üretimde fark ederseniz, yeni güncellenmiş template sonraki gün devreye girecektir; eski indeksleri `PUT mapping` ile güncelleyerek kısmen düzeltebilirsiniz ama her şeyi değil (bazı tipler değiştirilemez). Bu yüzden baştan planlama önemli.

Son çare olarak ILM'i devre dışı bırakıp manuel yönetim yapma seçeneği var ama tavsiye edilmez. ILM polices ve template'i silmek geri almanın bir yolu ama bunun yerine hataları düzeltmek daha iyi olur.

SIEM Kural Motoru: Tehdit Tespiti ve Alarm Yönetimi

Elasticsearch ve Kibana kurulumlarını tamamlayıp logları toplamaya başladıktan sonra, asıl SIEM değeri sağlayan kısma geldik: **Kibana SIEM Detection Engine** (Kibana güvenlik çözümünün tespit motoru). Bu motor, belirli kurallara göre logları sürekli tarayarak olası güvenlik olaylarını **alarm** (alert) olarak yükseltir. Bu bölümde, SIEM kural çeşitlerini, alarm döngüsünü (lifecycle), aksiyonları (connectors) ve bu kuralların API ile yönetimini inceleyeceğiz.

SIEM Kural Türleri ve Özellikleri

Elastic Security çözümünde oluşturabileceğiniz birkaç ana **kural türü** vardır [42]. Her biri farklı bir algılama metodolojisine hizmet eder:

- **KQL (Custom Query) Kuralı:** Belirli bir indeks veya indeks deseninde, belirli bir sorgu koşulunu (Kibana Query Language - KQL veya Lucene sorgusu) karşılayan belgeler arar [43]. Örneğin, `user.name: "root" and process.name: "bash"` gibi bir sorguyu her 5 dakikada bir çalıştırıp sonuç dönerse alarm üret diyebilirsiniz. Bu en genel ve esnek kural türüdür.
- **Threshold (Eşik) Kuralı:** Belirli bir alanda belli bir süre içinde *tekrarlanma sayısına* dayalı kuraldır. Örneğin, "aynı IP adresinden 5 dakika içinde 10'dan fazla başarısız giriş denemesi gelirse alarm üret" gibi [44]. Bu kural, KQL sorgusunun üzerine bir sayım şartı ekler. KQL ile de count yapamazsınız, o yüzden threshold özel bir türdür.
- **EQL (Event Correlation / Sequence) Kuralı:** Birden fazla olayı belirli bir düzen veya ilişki içinde arayan kuraldır. EQL, Elastic Query Language olarak da anılır (Event Query Language). Mesela "kısa süre içinde aynı kullanıcı önce admin grubuna eklendi sonra bir servis durdurulduysa" gibi iki olayı birleştiren bir senaryo yakalayabilir. Bu, ardışıl olay korelasyonları için idealdir.
- **Indicator Match (Göstergelerle Eşleşme) Kuralı:** Threat intelligence (tehdit istihbaratı) verisiyle sizin loglarınızı karşılaştırır. Örneğin harici bir IOC (indicator of compromise) listesinde IP adresleri var; sizin ağ loglarınızda bu IP'lerden biri görülürse alarm üret. Bu kural, Elastic Security'de threat intel index'leri ile sizin event index'lerinizi bir join yaparak eşleştirir diyebiliriz.
- **ML (Makine Öğrenmesi) Kuralı:** Elastic Stack içinde tanımlı Machine Learning anomaly job'larını kullanarak anomali skorları üzerinden alarm üretir. Örneğin "bir kullanıcı normalde hiç yapmadığı

bir işlemi yapmış" gibi patternleri ML tespit eder, kural bunları yakalar. (Not: ML tabanlı kurallar için **Platinum** lisans veya trial gerekir.)
- **Yeni Terms (New Terms) Kuralı:** Bu kural türü, belirli bir alanda daha önce görülmemiş yeni bir değerin ortaya çıkışını yakalar. Örneğin "ilk defa bir kullanıcı `login` event'ı üretti" gibi. Bu da 7.12 sonrası gelen bir özelliktir.

Makine öğrenmesi kuralları dışında yukarıdaki tüm kural tipleri ** ücretsiz ** (basic license) ile kullanılabilir. Bizim odaklanacağımız KQL (custom query), Threshold, EQL ve Indicator Match olacaktır.

Her kural oluştururken şunları belirlersiniz: - **İndeks veya Data View**: Hangi veri kaynaklarında arama yapılacağı (örn. `logs-*`). - **Filtre/Sorgu**: KQL/Lucene sorgusu veya EQL dizisi veya threshold koşulu vb. Her kural tipine göre form değişir. - **Zaman aralığı (Interval) ve Geriye Dönük Bakış (Look-back)**: Örneğin her 5 dakikada bir çalışsın ve son 5 dakikayı tarasın. Bu, kaçırmamak için genelde eşit veya biraz aşırı taranır (overlap olabilir). - **Üretilen alarmın durumu**: Kural çalışınca koşul sağlanırsa bir alarm dokümanı (signal) oluşturur. Bu alarm Kibana arayüzünde **Açık (Open)** durumda görünür. Bir analist bunu gördüğünde ya **Kapatır (Closed)** ya da **İnceleniyor (Acknowledged veya yeni versiyonda In Progress)** durumuna alır. Bu, *alert lifecycle* dediğimiz süreçtir. - **Alarm Birleştirme ve Bastırma (Deduplication & Suppression)**: Aynı kuralın aynı kaynağa dair çok fazla alarm üretmesini engellemek için mekanizmalar vardır. Örneğin threshold kuralında 10 başarısız deneme eşik, ama bir dakikada 100 deneme oldu diyelim, tek alarm tutup içinde count:100 bilgisini tutabilir. Ayrıca Elastic 8.5+ sürümünde gelen *alert suppression* ile belirli alanlara göre (örn. host.name) tekrarlı alarm oluşmasını belli süre engelleyebilirsiniz [45] .

- **Konnektörler ve Aksiyonlar (Connectors & Actions)**: Bir kural tetiklendiğinde otomatik bir aksiyon yaptırabilirsiniz. Örneğin email gönder, Slack mesajı gönder, bir webhook çağır, Jira kaydı aç, Elasticsearch index'ine yaz, veya entegrasyon varsa PagerDuty araması yap gibi. Bu aksiyonların çalışması için Kibana'da ilgili **Connector**'ların önceden yapılandırılması gerekir (örn. bir SMTP server tanımı, Slack webhook URL tanımı vs.). Temel aksiyonlar (index, webhook) genelde ücretsizdir, e-posta, Jira vb. çoğu **Gold/Platinum** lisans ister. Biz konsepti anlatacağız.

Bir kural oluşturulduktan sonra Kibana bir arka plan **Task** planlar ve her periyotta bu sorguyu çalıştırır. Bulduğu sonuçları `.alerts-security.alerts` benzeri özel bir indekse yazar (her alarm bir dokümandır). Bu dokümanlar "siem-signals" eski adıyla bilinir, ve SIEM app bunları gösterir.

Örnek Kural Oluşturma

1. **KQL Kural Örneği:** Örneğin Linux auth loglarından kök kullanıcıyla başarısız SSH giriş denemelerini tespit eden bir kural yazalım:
2. İndeks: `logs-linux-*` (farz edelim linux logları buraya gidiyor)
3. Sorgu: `event.category: authentication AND event.outcome: failure AND user.name: root`
4. Zaman aralığı: Her 5 dakikada bir, son 5 dakikayı tarasın.
5. Kural adı: "Root Failed SSH Logon"
6. Severity (Şiddet): Medium diyelim.
7. Risk Score: 50 verelim.
8. Üretilecek alarmın alanları: Kural tanımı bunları doldurur, ayrıca eşleşen her log olayının alanları alarm dokümanına kopyalanır.

Bu bir custom query rule olduğundan her bir eşleşen event için bir alarm üretilecektir. Son 5 dk'da 3 tane root failure olduysa, 3 alarm (veya ayara bağlı tek alarm içinde 3 count). Kaydedip etkinleştirince kural çalışır, eğer data varsa Alerts tab'inde Open olarak listelenir.

1. **Threshold Kural Örneği:** Aynı örneği threshold ile şöyle yapabilirdik:
2. Sorgu: `event.category: authentication AND event.outcome: failure`
3. Threshold field: `user.name`
4. Threshold value: 5 deneme
5. Zaman aralığı: 5 dakika

6. Yani: "Herhangi bir kullanıcı 5 dk içinde 5 defa failed auth yaparsa alarm". Bu kural alarm dokümanında hangi kullanıcı olduğu ve sayısı gibi bilgileri doldurur. Bir alarm = bir kullanıcı eşiği aşmış demek. Root özelinde demedik ama belki root dışı adminler de takibe değer.

7. **EQL Kural Örneği:** Örnek: Windows'da arka arkaya belirli iki event:

8. Olay 1: Event ID 4720 (Yeni kullanıcı oluşturuldu)
9. Olay 2: Event ID 4724 (Bir kullanıcı parolasını resetledi)

10. Koşul: Aynı Security ID (kimin tarafından yapıldı) ve 1 dakika içinde 4720 ardından 4724 gelirse. EQL sorgusu şöyle yazılır:

    ```
    sequence by user.id with maxspan=1m
      [ any where event.code == "4720" ]
      [ any where event.code == "4724" ]
    ```

    Bu Kibana EQL kurallar sayfasına yazılır. Böyle bir sequence kuralı, belki bir sysadmin bir hesap açıp hemen parola set ettiyse (usulüne uygun) alarm üretir, belki bir saldırgan art arda yaparsa alarm üretir. Sadece örnek. Bu kural bir "Event Correlation Rule" olarak tanımlanır Kibana'da.

11. **Indicator Match Kural Örneği:** Örneğin elimizde `threat-indicators` adında bir indeks var, içinde `ip` alanı tutuyor. Bizim de firewall loglarımız `source.ip` alanında IP'leri tutuyor. Kural ayarı:

12. İndeks: `logs-firewall-*`
13. Göstergeler indeksi: `threat-indicators`
14. Eşleştirilecek alanlar: kaynak IP <-> indicator IP
15. Kural mantığı: Eğer bir log satırındaki source.ip, threat intel indeksindeki herhangi bir ip ile eşleşirse alarm. Bunu Kibana'da Indicator Match rule seçerek dolduruyoruz. Bu kural herhangi bir sayı/time threshold'a bakmaz, feed'ler genelde IOC listeleri küçük tutulur. Her eşleşme bir alarm.

Kural oluştururken doldurduğumuz diğer alanlar: - **Severity:** Low, Medium, High, Critical şeklinde etiket. - **Risk Score:** 0-100 arası bir puan. Bunlar raporlama ve sıralama için anlamlı. - **Description, Mitre ATT&CK mapping vs.:** Kibana SIEM, kuralla ilgili Mitre tactic/technique seçmenize izin verir, ayrıca açıklama, önerilen yanıt adımları vb. yazabilirsiniz. Bu, alarm analizi sırasında analiste rehberlik eder.

Alarm Yaşam Döngüsü ve Analiz

Kural tetiklenince **Open Alerts** listesine bir giriş düşer. Analist bunu gördüğünde bakar: - Alarm detayında orijinal event bilgileri, kuralın adı, seviyesi, vs. hepsi yer alır. - Analist duruma göre araştırma yapıp bu alarmın gerçek bir olay mı yoksa yanlış pozitif mi olduğunu değerlendirir.

Ardından şu aksiyonlar alınabilir: - **Mark as In Progress (İnceleniyor):** Bu, ekip içinde "bu alarm ile ilgileniliyor" demektir. (8.x öncesi versiyonlarda "acknowledged" denirdi). - **Close (Kapat):** Alarm kapatılır. Bir not girilebilir, örn: "False positive, dev sunucuda test aktivitesi". - **Comment Ekleme:** Her alarm kaydına yorum eklenebilir. Bu, ileride bir Case (olay kaydı) oluşturulursa da taşınabilir.

Kapalı bir alarm, eğer aynı kural tekrar aynı olayı yakalarsa yeniden açılabilir (bazı dedup ayarlarına bağlı). Elastic 8.4 sonrasında "flapping" denilen bir durum eklendi: Kapatılmış ama tekrar gelen alarm reopen edilebilir veya yeni alarm mı oluşturulsun seçilebilir.

Alarmlar kendi indekslerinde tutulur dedik. Bu indeks ILM ile yönetilir, siz kapatınca dokümanda status field değişir ama doküman silinmez (sadece UI'da filtreyle görünmez olur).

Connectors/Aksiyonlar demiştik: Mesela bir High severity alarm oluşunca e-posta göndermeyi ayarladıysanız, Kibana o alarm verilerini e-postada belirttiğiniz şablona koyup yollar. E-posta connector'ünü Stack Management > Connectors bölümünden önceden kurmanız gerekir (SMTP host, port, user/pass vb). Slack connector için webhook URL vs girersiniz, sonra kuralda "Slack mesaj gönder" aksiyonu eklersiniz.

Desteklenen aksiyon tipleri arasında: - Email - Webhook (HTTP POST) - Slack, PagerDuty, ServiceNow, Jira, Teams vs. (çoğu platinum gerektirir) - Index (alarm verisini başka bir ES indeksine de yazar, audit için kullanışlı olabilir) - Kibana console log (deneme amaçlı basit çıktı, pek kullanılmaz)

Her aksiyon tanımında bir **throttle** süresi ayarlanabilir, yani aynı kural arka arkaya 10 alarm üretse bile e-postayı 1 kere gönder gibi.

Detection Engine Yönetimi ve Detection-as-Code

Kibana arayüzüyle kural oluşturmak kullanıcı dostudur. Ancak çok sayıda kuralı yönetmek, versiyonlamak veya SIEM kural setlerini dışarıdan alıp uygulamak için **Detection Engine API**'larını kullanabiliriz [46]. Elastic, tüm bu kural CRUD işlemleri için Kibana REST API uç noktaları sağlar:

Örneğin: - **Kural oluşturma (CREATE):** `POST /api/detection_engine/rules` endpoint'i JSON gövde ile kural bilgilerini alır ve yeni kural oluşturur. - **Listeleme (READ):** `GET /api/detection_engine/rules` ile kuralları çekebilir veya id ile tekil kural alabilirsiniz. - **Güncelleme (UPDATE):** `PUT /api/detection_engine/rules/<id>` gibi (tam endpoint dokümana bakmak lazım). - **Silme (DELETE):** `DELETE /api/detection_engine/rules?id=<id>`. - **İçe/Dışa Aktarma (Import/Export):** En kullanışlılarından, `POST /api/detection_engine/rules/_export` ve `_import` uçlarıdır. Export, NDJSON formatında seçili veya tüm kuralları dışarı verir. Import ile belki başka bir Kibana ortamında bu NDJSON'ı içeri alabilirsiniz [47] [48]. Bu, detection rules as code için kapı açar.

Elastic ayrıca kendi kural setini güncel tutmak için **prebuilt rules** konseptine sahip. Kibana ilk kurulumda internete erişebiliyorsa, Elastic public kural repository'sinden en son kural tanımlarını indirebilir. Bu **prebuilt** kurallar sizin ortamınıza uygun olanları enable etmeniz için sunulur. Örneğin

“Multiple Failed Login Attempts” diye hazır bir threshold kuralı var, siz datayı gönderdiyseniz onu açabilirsiniz. Prebuilt kuralların ID’leri sabittir, güncelleme gelirse Kibana size “prebuilt rules updates available” der, siz de update edersiniz (veya otomatik).

Detection-as-Code kavramı, SIEM kurallarının sürüm kontrolü (Git gibi) ile yönetilmesidir. Elastic bu amaçla open-source kural depoları ve araçları sağlıyor. Örneğin GitHub’daki `elastic/detection-rules` reposu, tüm kural setini içerir (toml formatlarında). Oradaki bir script ile kural geliştirip test edebilir, sonra Kibana API’ye push edebilirsiniz [49] . Kurallar JSON formatında NDJSON ile versiyonlanıp pipeline’lara konulabilir.

Bizim rehberimizde detayına inmeye gerek yok belki ama bilmekte fayda var: - Büyük ortamlarda, kural değişikliklerini, yeni kural eklemelerini kod review’dan geçirmek isterseniz bu yöntem kullanılır. - Kibana API’lerini çağırmak için bir servis hesabı (API key) oluşturup kullanabilirsiniz, böylece CI/CD pipeline’ınız `curl` ile kuralları deploy edebilir.

> **Neden:** SIEM kuralları, sürekli ayarlanıp iyileştirilmesi gereken şeylerdir. Elastic, hem kendi best practice kural setini sunarak kurumlara başlangıç noktası verir, hem de özelleştirmeye izin verir. Kural tiplerinin çeşitliliği sayesinde sadece basit IOC eşleşmeleri değil, davranışsal tespitler de yapılabilir [50] . Bu, olgun bir SOC ekibi için elzemdir.
>
> API ve otomasyon tarafı ise, özellikle çok sayıda farklı ortama (dev/test/prod Kibana gibi) kural taşımak veya açık kaynak kural setlerini uygulamak istediğinizde işe yarar. Detection-as-Code yaklaşımı, kuralların da tıpkı uygulama kodu gibi version control altında geliştirilmesini sağlar. Bu şekilde, örneğin MITRE ATT&CK TTP’lerine karşılık gelen kural setinizi bir repoda tutup, her yeni tehdit bilgisi eklendiğinde yeni kural ekleyip CI ile dağıtabilirsiniz.
>
> **Doğrulama:** Bir kural oluşturduğunuzda Kibana, onu arka planda hemen çalıştırır (veya schedule gelince çalıştırır). Test amaçlı kuralınızın hemen alarm üretmesini istiyorsanız, loglarda bir örnek olay oluşturabilirsiniz. Mesela threshold rule test etmek için bir dakikada 5 başarısız login yapıp loglara düşmesini sağlayın. Kibana Alerts tab’inde alarmı görmelisiniz.
>
> Ayrıca Kibana’nın **Rule Monitoring** sekmesinde kuralın son ne zaman çalıştığı, kaç doküman taradığı, error verip vermediği gibi metrikleri takip edebilirsiniz. Eğer bir kural sürekli error durumuna düşüyorsa (mesela yanlış index pattern verdiniz, index yok), bunu oradan fark etmek mümkün.
>
> Connector aksiyonu test etmek için de Kibana, Connector tanımlama ekranında “send test message” gibi bir buton sunar. Önemli: Kibana Connectors, Kibana sunucusunun o harici servise ulaşabildiğini varsayar. Örneğin e-posta için port 587’ye çıkışını firewall engelliyorsa mail göndermez. O yüzden test butonu ile doğrulayın.
>
> **Geri Alma:** Yanlış bir kural tanımladıysanız ve çok sayıda gereksiz alarm ürettiyse, öncelikle kuralı **durdurun (disable)** veya silin. Ardından oluşan alarmları Kibana arayüzünden toplu seçip "Close selected" diyerek kapatabilirsiniz. Alarmları silmek diye bir şey yok (indexten manuel silinebilir ama arayüzden yoktur), kapalı bırakmak yeterli.
>
> Eğer connector bir aksiyonu yanlış kişilere gönderdi vs, hemen rule actions kısmından o aksiyonu kaldırıp kaydedin veya kuralı disable edin. Connector’ı da Stack Management’ten

silmek gerekebilir. Unutmayın, bir kural her çalıştığında aksiyon tetiklenir. Örneğin Slack'e aynı mesajı defalarca atmasın diye throttle sürelerini iyi ayarlayın (kural ayarlarında "re-notify interval" şeklinde bulunur).

Kural yönetiminde bir problem de performans olabilir: Çok geniş bir zaman aralığını tarayan kural, Elasticsearch'e ağır query atarak cluster'ı yorabilir. Bunu engellemek için kuralları spesifik ve makul look-back aralıklı tutun. Yine de bir kural tüm cluster'ı yavaşlattıysa, disable edip sorgusunu optimize ederek tekrar yazmak gerekir.

Son olarak, kural setinizi sürekli güncel tutmayı unutmayın. Yeni tehditler çıktıkça Elastic'ten prebuilt rule güncellemelerini uygulayın veya kendi kurallarınızı ekleyin. Durağan bir kural seti, bir süre sonra yenilikleri yakalayamaz hale gelir.

Sürekli İzleme, Bakım ve Sorun Giderme

Sistemimiz çalışır hale geldikten sonra da yapmamız gereken işler bitmiyor. Bu bölümde, Elastic Stack bileşenlerinin **izlenmesi (monitoring)**, servislerin yönetimi, yedekleme ve sık karşılaşılan hataların ele alınması gibi konulara değineceğiz. Özellikle tek bir sunucuda her şey çalıştığı için sistem kaynaklarının takibi, logların döngüsü (rotation) ve servis sürekliliği kritik olacaktır. Ayrıca çeşitli **break/fix** senaryolarını birkaç örnekle ele alıp nasıl çözeceğimizi göstereceğiz.

Stack Monitoring (İzleme)

Elastic, kendi kendini izleme kabiliyeti sunar. **Stack Monitoring** özelliği ile Elasticsearch ve Kibana'nın performans metriklerini, Logstash pipeline istatistiklerini görebilirsiniz. Bunu aktifleştirmenin iki yolu var: - **Internal Monitoring Collection:** Elasticseach ve Kibana, metric verilerini kendi içlerinde toplayıp `.monitoring-*` indekslerine yazar. (Basic lisansta bile mevcuttur). Kibana'da Monitoring UI açıp aynı cluster'a bakabilirsiniz. - **Metricbeat ile Monitoring:** Elastic, internal methodu gelecekte kaldırıp Metricbeat kullanımını tavsiye ediyor. Yani her bileşenin metricbeat modulünü açıp metricbeat verilerini toplayıp bir monitoring cluster'ına göndermek.

Basitlik için belki internal yöntemi kullanabiliriz. Bunu açmak için Kibana'da Stack Management > Stack Monitoring > Enable Monitoring diyebilirsiniz. Ardından, aynı cluster'a mı yazsın başka cluster'a mı seçersiniz (development ortamda same cluster iş görür). Aktifleştikten sonra `.monitoring-es-*, .monitoring-kibana-*` vs. indexler oluşur.

Bunun sonucunda Kibana Monitoring UI'da: - Elasticsearch heap kullanımı, indexing hızı, arama sorgu süreleri vs. - Kibana response times, concurrency vs. - Logstash event in/out throughput, queue size vs. gibi birçok metrik görebilirsiniz.

Tek node'da belki aşırıya kaçabilir ama en azından ES heap ve disk doluluk takibi için iyidir.

Ayrıca, **DevOps** seviyesinde, CPU, RAM, disk, network gibi temel metric'leri de gözlemlemek gerekir (Elasticseach bunları x-pack ile kısmen raporlar ama tam değil). Burada tercihinize göre top/htop, netstat, iostat gibi araçlarla veya Metricbeat system modulü ile bu gözlemi yapabilirsiniz.

Alarming: Monitoring UI sadece gösterir, alarm kurmaz. Örneğin CPU %90'ı geçti alarm, veya disk %5 kaldı alarm istiyorsanız Metricbeat + Watcher (veya Kibana Rules - Kibana'da Metrics app ile threshold rule) kullanmanız lazım. Basic lisansta Watcher yok ama Kibana "Rules" kısmında Metrics threshold var.

Bunu inceleyebilirsiniz (bu, SIEM detection'dan farklı, Kibana alarmları kısmında yer alır ve CPU yüksek vs. tetikleyebilir).

Servis Yönetimi (systemd, otomatik başlatma, log rotation)

Systemd servis dosyaları sayesinde, makine yeniden başladığında Elasticsearch, Kibana, Logstash otomatik kalkacak şekilde ayarladık (enable komutlarıyla). Bu, sürekli çalışırlık için önemli.

Her servis loglarını `journalctl` ile takip ettik. Ancak bazıları /var/log dizinlerine de log yazar: - Elasticsearch logları: `/var/log/elasticsearch/elasticsearch.log` döner (başta journald'a da gider, 7.x ve 8.x paketleri journald yerine direct dosyaya yazıyor olabilir config'e göre). - Kibana logları: `/var/log/kibana/kibana.stdout` ve `.stderr` olabilir, ya da sadece journald. - Logstash log: `/var/log/logstash/logstash-plain.log` biriktirir.

Bu log dosyalarının **rotasyon** ayarlarına dikkat edin. Genelde /etc/logrotate.d içinde ayar dosyaları olur paketle gelen. Elasticsearch.log çok büyümesin diye rotasyonu var (daily ve 7 dosya tut default). Logstash-plain.log da öyle. Kibana journald'a yazıyorsa journald'ın kendi rotation'ı var.

Disk Doluluk: Tek sunucuda veri ve log aynı diskte olabilir. /var/lib/elasticsearch altındaki veri klasörü büyüdükçe disk dolar. ILM ile 90 günde silecek dedik. Yine de birim log hacmine göre disk planlamanızı yapın. Üretimde, ILM ile silinen indekslerin disk alanını gerçekten boşalttığını (ve snapshot alınıp alınmadığını) izlemek lazım. Disk % eşiğe gelirse, Elasticsearch yazmayı durdurur (eserly stop) ve cluster yeşilden sarıya döner. O yüzden disk usage metric takibi hayati.

Snapshot/Yedek: Prod ortamdaysa, logların yedeğini almak isteyebilirsiniz. Elasticseach Snapshot & Restore özelliği ile S3, NFS vb. ortamlara periyodik snapshot alabilir. Bu, veritabanının tutarlılığını korur. Bizim lab ortamında çok gerek yok, ama kurcalarken bir cluster state bozulması, index silinmesi vs. olursa snapshot yoksa veri gider.

Yaygın Hata Senaryoları ve Çözümleri (Break/Fix)

Son olarak, sık karşılaşılabilecek bazı hata durumlarını ve bunların çözüm yollarını özetleyelim:

- **Problem: Kibana Elastic'e Bağlanamıyor (Status Code 500 veya Kibana UI "Cannot connect to Elasticsearch" hatası).**
- *Olası Nedenler:* Elasticsearch kapalı, Elasticsearch TLS sertifikası Kibana'nın trust store'unda yok, elasticsearch.username/password yanlış, veya Kibana'nın elasticsearch.hosts ayarı hatalı.
- *Çözüm:*
    - Önce Elasticsearch'ün 9200'de çalıştığını ve `elastic` loginin çalıştığını doğrulayın (curl ile).
    - Kibana logunda genelde daha detay yazar. Örneğin "no living connections" tüm denemeler başarısız demek; "unable to verify the first certificate" sertifika sorunu demek.
    - Sertifika sorunuysa, Kibana config'te `elasticsearch.ssl.verificationMode: "certificate"` (yalnızca CA kontrolü yapsın, CN eşleşmesini es geçsin) gibi bir ayar ile belki düzelebilir. En iyisi CA'yı doğru verdiğinizden emin olun.
    - Kimlik hatasıysa, `elastic` şifresini kibana.keystore'a yanlış girmiş olabilirsiniz, tekrar ekleyin.

    - Elasticsearch URL hatalıysa, IP/host doğru mu bakın. IP yazıp sertifika CN mismatch olursa Kibana bağlanmaz. Bu durumda ya sertifikayı IP SAN ekleyerek yeniden üretin ya Kibana config'te `verificationMode: "none"` diyerek (sadece dev/test için kabul edilebilir) geçici çözüm sağlayın.

- **Problem: Elasticsearch Yüksek CPU veya RAM Kullanımı Nedeniyle Yavaş (Gelen logları işleyemiyor, aramalar yavaş).**

- *Olası Nedenler:* JVM heap boyutu yetersiz olabilir, gelen indeksleme yükü donanımdan fazla olabilir, ya da arka planda ağır sorgular (ör. çok geniş zaman diliminde SIEM kuralı) çalışıyor olabilir.

- *Çözüm:*

    - Heap boyutunu artırmayı düşünün: /etc/elasticsearch/jvm.options.d/ ile Xms/Xmx parametresini mevcut RAM'in ~%50'sine ayarlayabilirsiniz (makine 8GB ise 4GB verin). Servis restart gerekir.
    - Indeksleme iş yükünü hafifletin: Örneğin Logstash filtrelerinde gereksiz parse'ları kaldırın, ya da Beats'in gönderdiği event sayısını sınırlayın (çok detay logging varsa azaltın).
    - Sorgu optimizasyonu: SIEM kural sorguları eğer index mapping'ine uygun değilse (örn. wildcard arama) CPU yiyebilir. Kural sorgularınızı index planına göre optimize edin (mümkünse keyword alanlar üzerinde eşleşme yapın).
    - Coğrafi olarak tek disk ve belki zayıf I/O varsa, indexing buffer belki doluyor olabilir. ES logunda "writing new index_..." gibi log flush'ları çok görüyorsanız, belki disk I/O limitte. Bu durumda I/O iyileştirme (SSD'ye geçiş) gerekebilir.

- **Problem: Logstash Bağlantı Hatası (Elasticsearch output) – "Could not index event to Elasticsearch... 401 unauthorized / 403 forbidden".**

- *Olası Nedenler:* Logstash'in Elastic'e bağlanmak için kullandığı kullanıcı yetkisiz. Biz `elastic` kullandık, onunla 403 olmaz ama belki yanlış şifre 401 yapar. Eğer özel kullanıcı kullansaydık ve indexe yazma izni yoksa 403 olur.

- *Çözüm:*

    - Logstash'te keystore'a doğru şifre girdiğinizi teyit edin (yanlışsa 401 alırsınız).
    - Kullanıcı rol ayarlarını kontrol edin: diyelim `logstash_writer` diye bir user açtınız, buna `write` izni yoksa 403 çıkar. Çözüm: appropriate role ile yetkilendirin (örn. index priv: create_index, create_doc en az).
    - Pipeline'de index adı data stream ile uyumlu değilse de reddedilebilir (8.x'te normal indexe yazmak istiyorsanız "allow_ilm_indices" false vs. gibi!). Kontrol edin output plugin error detayını.

- **Problem: Logstash Pipeline Çalışmıyor – Hata ile Çıkıyor veya Hiç Veri Almıyor.**

- *Olası Nedenler:* Config syntax hatası olabilir (Logstash start olurken error verir). Veya input portu açamadı (özellikle 514/6514 gibi privileged portları). Veya firewall portları kapalı.

- *Çözüm:*

    - `journalctl -u logstash` e bakın. Syntax error varsa hat mesajından hangi dosya satır anlaşılır, düzeltin.

    - Port binding hatasıysa: 514/6514 için logstash keplerini root'a verip veremeyeceğinize bakın. `setcap 'cap_net_bind_service=+epi' /usr/share/logstash/jdk/bin/java` belki çözebilir (java process'ine low port yetkisi vermek), ama bu supported değil genelde. Kolay çözüm: 10514/16514 gibi portlar kullanın ve syslog göndericileri oraya yönlendirin.
    - Veri gelmiyorsa: Beats veya cihazlar Logstash'a ulaşıyor mu network'ü kontrol edin (telnet / openssl s_client ile 5044'e test).
    - Ayrıca Logstash pipeline çalışıyor ama filtre yanlış olabilir, ECS alanları gelmiyordur. Bu logic hataları için de debug mod açabilirsiniz. Logstash config'e `stdout { codec => rubydebug }` output ekleyip ne geldiğini görebilirsiniz (test için).

- **Problem: Disk Dolu – Elasticsearch İndeks Yazamıyor (Cluster Read-Only State).**

- *Belirti:* Elasticsearch loglarında `flood stage disk watermark [95%] exceeded` uyarısı görülür ve indexler read-only moda alınır.

- *Çözüm:*

    - Acil olarak disk alanı boşaltın: en kolay eski indeksleri silin (temporal veriler zaten). `DELETE logs-2023.07.*` gibi komutlarla 3 ay öncesini silin. Veya snapshot alabiliyorsanız alıp silin.
    - Silince birkaç dakika içinde ES otomatik read-only flag'ı kaldırır watermark altına inerse. Eğer kalmazsa, index ayarlarından `index.blocks.read_only_allow_delete` bayraklarını manuel false yapmanız gerekebilir (her indeks için).
    - Uzun vadede, disk kapasitesini artırın veya ILM retention'ı kısaltın.

- **Problem: Kibana SIEM Arayüzünde Prebuilt Kural Yükleyememe veya Hata.**

- *Olası Neden:* Kibana internet erişimi yoksa prebuilt gelmez. Veya versiyon uyumsuzluğu bir bug olabilir.

- *Çözüm:*

    - İnternet yoksa, offline paket import edilebilir (Elastic bir .zip ile prebuilt rules offline yükleme imkanı sunuyor mu bilmiyorum, belki detection-rules repo ile halledilebilir).
    - Versiyon bug ise upgrade yapmak gerekebilir. Minor updateler çıkıyor, Kibana->Stack Management->Detection Rules belki error veriyorsa forumlara bakmak lazım.

- **Problem: SIEM Alarm Üretmesi Gereken Durumlarda Üretilmiyor.**

- *Olası Nedenler:* Kural yanlış tanımlanmış (sorgu yanlış ya da index pattern yanlış), loglarda ilgili alanlar eksik (ECS alanı yok, kural match edemiyor), kural disable durumda kalmış, veya rule execution error alıyor.
- *Çözüm:*
    - Önce Discover'da ilgili olayı manuel aratarak kural koşulunun event'ta görünüp görünmediğine bakın. Belki event.outcome field gelmiyordur logdan (Biz ECS dedik ama parse etmediysek yoktur). Kuralı bu gerçeğe göre düzenleyin ya da pipeline'ı iyileştirin.
    - Kuralın son run bilgisini Rule Monitoring'den inceleyin. Belki "partial failure" diyor, tıklayın neyin hata olduğunu söyler. (Mapping conflict de olabilir)
    - Gerekirse kuralı debug modda API'den çalıştırabilirsiniz (Kibana dev tools: `POST api/detection_engine/rules/<id>/_simulate` gibi bir şey var mı bilmiyorum). En kolayı

kural query'sini Dev Tools Console'da `.alerts-security.alerts` endeksine query atarak test edebilirsiniz (ancak user friendly değil).

Bu sayılanlar dışında da pek çok senaryo olabilir, ancak temel yaklaşım: logları ve ayarları dikkatle inceleyip sorun giderme adımlarını sistematik uygulamaktır. Elastic forumları ve dökümantasyonu bu konuda geniş kaynak sunar.

Sonuç ve Lab Egzersizleri

Bu kılavuz boyunca, Ubuntu 22.04 üzerinde tek sunucu bir Elastic SIEM ortamını sıfırdan kurduk ve detaylı şekilde yapılandırdık. Tüm bileşenlerin güvenli (TLS & kimlik doğrulamalı) iletişim kurmasını sağladık, log verilerini ECS şemasına uygun toplamayı ve indekslemeyi başardık, SIEM tespit kurallarını devreye aldık. Gerçek dünyada bir SIEM implementasyonunu başarılı kılan unsurların çoğuna dokunduk: doğru sertifika yönetimi, konfigürasyonun kod ile otomasyonu, performans optimizasyonları ve izleme, hata durumlarında çözüm üretme gibi.

Artık öğrendiklerinizi pekiştirmek için bazı lab egzersizlerine hazırsınız:

Lab 1: TLS Sertifikası Kurulum ve Doğrulama

- **Görev:** Kendi sertifika otoritenizi oluşturun (ör. OpenSSL veya elasticsearch-certutil ile) ve Elastic Stack bileşenlerinizin hepsine bu CA tarafından imzalanmış sertifikalar dağıtın. Elasticsearch, Kibana ve Logstash'ın bu yeni sertifikalarla iletişim kurmasını sağlayın.
- **Adımlar:** CA sert ve key üret, Elasticsearch için node sert, Kibana için server sert, Logstash için server sert oluştur. Config dosyalarını uygun şekilde güncelle. Servisleri restart et.
- **Doğrulama:** Tüm servis loglarında TLS handshake hatası olmadığını teyit edin. Curl ile Elastic'e CA'nız ile bağlanın. Kibana arayüzüne tarayıcıdan bağlanırken kendi CA'nızı tarayıcıya güvendirerek hata almadan erişin.

Lab 2: Enrollment Token ile Kibana Kurulumu

- **Görev:** Kibana'yı tamamen sıfır (başka bir makine ya da konteyner olabilir) kurup Elasticsearch'e bağlamak için enrollment token kullanın.
- **Adımlar:** Yeni bir Kibana instance kurun. Elasticsearch'ten `elasticsearch-create-enrollment-token -s kibana` alıp Kibana'yı ilk başlatırken tarayıcı arayüzünden veya CLI'dan bu token ile kaydettirin.
- **Doğrulama:** Kibana'nın `kibana.yml` dosyasına otomatik eklenen ayarları inceleyin (CA sertifikası, kibana_system şifresi vs. gelmiş mi bakın). Kibana başarılı bağlanıp login ekranını gösteriyorsa token işi tamamdır.

Lab 3: Özel Bir SIEM Kuralı Yazma ve Test Etme

- **Görev:** Kendi ortamınızdan bir senaryo düşünün (örn. belirli bir kullanıcı ile ilgili şüpheli aktivite, veya belli bir IP'den gelen tarama). Bu senaryoya uygun bir detection rule tanımlayın.
- **Adımlar:** Örneğin, Windows için "Guest hesabıyla başarısız logon denemeleri" kuralı yazın. KQL sorgusunu belirleyin (`user.name: "Guest" and event.type: "authentication_failure"` gibi). Kibana'da kuralı oluşturup enable edin.
- **Doğrulama:** Test amacıyla birkaç log satırı üretin (Event ID'leri Kibana dev tools ile ekleyebilirsiniz ya da bir script ile). Kuralın tetiklenip tetiklenmediğini Alerts tablosunda görün. Gerekirse kuralı troubleshoot edin (zaman aralığını genişletmek, koşulu basitleştirmek vs.).

Lab 4: ILM Politikasını Sınama

- **Görev:** ILM ile indekslerin yaşam döngüsünü yönettiğimizden emin olmak için, 1 günlük retention'lı bir test politikası uygulayın ve rollover ile silme işlemlerini gözlemleyin.
- **Adımlar:** `test_policy` diye ILM tanımlayın: 1 dakika sonra rollover, 2 dakika sonra silme yapsın. Bunu `logs-test*` indeksine şablonla uygulayın. Ardından bir script ile her 10 saniyede logs-test indeksine bir doc yazan bir döngü başlatın (böylece veri akışı olsun).
- **Doğrulama:** Monitoring'den veya `_cat/indices` çıktısından, 1 dk sonra logs-test-000002 oluştuğunu, 2 dk sonra 000001'in silindiğini teyit edin. Bu hızlı döngü politikayı test etmek içindi, daha sonra eski policy'yi geri alın.

Bu egzersizler, rehberde anlatılan kavramları uygulamanıza yardımcı olacaktır. Gerçek bir ortam kurduğunuzda, buradaki adımları kendi ihtiyaçlarınıza göre uyarlamanız gerekecektir. Özellikle ölçek büyüdükçe (örn. birden fazla Elasticsearch node'u, birçok Beats agent'ı, yüksek hacimde log akışı) mimaride bazı değişiklikler gerekebilir (coğrafi dağıtık mimari, koordinasyon node'ları, load balancer vb.). Ancak temeller aynı kalır: Güvenlik, tutarlılık (ECS), otomasyon ve sürekli izleme.

Kaynakça: - Elastic resmi dokümantasyonu – Elasticsearch güvenlik ayarları [1] [2], Kibana konfigürasyon rehberi [31], Logstash keystore kullanımı [39] [35], SIEM kural oluşturma kılavuzu [43] vb. - Elastic forumları ve çözümler: https://discuss.elastic.co, benzer kurulum senaryoları. - ECS (Elastic Common Schema) reference: Hangi alan ne tipte olmalı, SIEM kurallarında ne kullanılıyor, bkz. Elastic ECS documentation.

Bu kılavuzdaki adımları takip ettiğinizde elinizde çalışır durumda bir SIEM platformu olacaktır. Bundan sonrası, bu platformu aktif biçimde kullanmaya, gelen alarmları doğru yorumlamaya ve gerektiğinde yeni dedeksiyon kuralları yazarak ortamınızı korumaya kalıyor. Başarılar ve iyi çalışmalar! [3] [4]

---

[1] [2] [6] [7] [8] [9] [11] [12] [13] [22] [23] Install Elasticsearch with a Debian package | Elastic Docs
https://www.elastic.co/docs/deploy-manage/deploy/self-managed/install-elasticsearch-with-debian-package

[3] [4] [5] TLS encryption for cluster communications | Elastic Docs
https://www.elastic.co/docs/deploy-manage/security/secure-cluster-communications

[10] [24] [32] Install Kibana with Debian package | Elastic Docs
https://www.elastic.co/docs/deploy-manage/deploy/self-managed/install-kibana-with-debian-package

[14] [15] [16] [18] [19] elasticsearch-certutil | Reference
https://www.elastic.co/docs/reference/elasticsearch/command-line-tools/certutil

[17] [27] [28] Set up HTTPS | Elastic Docs
https://www.elastic.co/docs/deploy-manage/security/set-up-basic-security-plus-https

[20] [21] [34] [35] [38] [39] Secrets keystore for secure settings | Logstash
https://www.elastic.co/docs/reference/logstash/keystore

[25] [26] Install Kibana on Windows | Elastic Docs
https://www.elastic.co/docs/deploy-manage/deploy/self-managed/install-kibana-on-windows

[29] [30] Minimal security setup | Elastic Docs
https://www.elastic.co/docs/deploy-manage/security/set-up-minimal-security

[31] Configure Kibana | Elastic Docs
https://www.elastic.co/docs/deploy-manage/deploy/self-managed/configure-kibana

33 Using environment variables | Logstash
https://www.elastic.co/docs/reference/logstash/environment-variables

36 Design options for ingesting syslog data - Logstash - Elastic Discuss
https://discuss.elastic.co/t/design-options-for-ingesting-syslog-data/228619

37 SSL/TLS on Logstash syslog input plugin without XPACK
https://stackoverflow.com/questions/60879130/ssl-tls-on-logstash-syslog-input-plugin-without-xpack-nginx-or-any-third-party

40 Set Up TLS To Encrypt Communications in the Elastic Stack
https://docs.appdynamics.com/appd/onprem/24.x/latest/en/events-service-deployment/set-up-tls-to-encrypt-communications-in-the-elastic-stack

41 Index lifecycle management tutorials | Elastic Docs
https://www.elastic.co/docs/manage-data/lifecycle/index-lifecycle-management/ilm-tutorials

42 43 45 50 Create a detection rule | Elastic Docs
https://www.elastic.co/docs/solutions/security/detect-and-alert/create-detection-rule

44 Create a detection rule | Kibana API documentation - Elastic
https://www.elastic.co/docs/api/doc/kibana/operation/operation-createrule

46 Security detections | Kibana API documentation - Elastic
https://www.elastic.co/docs/api/doc/kibana/group/endpoint-security-detections-api

47 Import detection rules | Kibana API documentation - Elastic
https://www.elastic.co/docs/api/doc/kibana/operation/operation-importrules

48 List all detection rules | Kibana API documentation - Elastic
https://www.elastic.co/docs/api/doc/kibana/operation/operation-findrules

49 Syncing Rules and Data from VCS to Elastic Security - DaC Reference
https://dac-reference.readthedocs.io/en/latest/core_component_syncing_rules_and_data_from_vcs_to_elastic_security.html