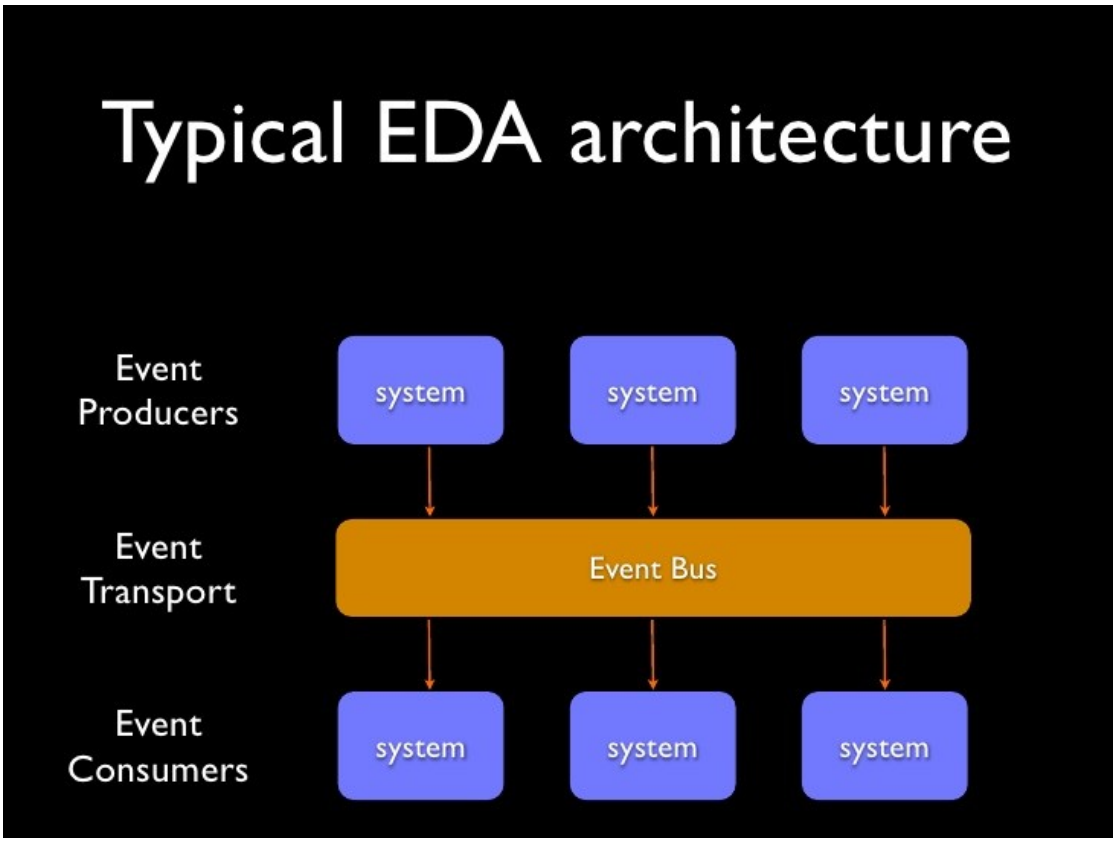
# Introduzione

Considerando che molti web server utilizzano un'architettura thread o basato sui processi, Nginx si distingue con una sofisticata architettura **event-driven** che consente di scalare centinaia di migliaia di connessioni simultanee su hardware moderno.

N.B. *Il termine **scalabilità**, nelle telecomunicazioni, nell'ingegneria del software, in informatica e in altre discipline, si riferisce, in termini generali, alla capacità di un sistema di "crescere" o diminuire di scala in funzione delle necessità e delle disponibilità. Un sistema che gode di questa proprietà viene detto **scalabile**. (cit. it.Wikipedia.org)*

---

# Event-Driven Architetture (EDA)




Un'**architettura event-driven** (EDA) è un framework che gestisce il comportamento attorno alla produzione, la rilevazione e il consumo di eventi, nonché le risposte che evocano. Un evento è una qualsiasi occorrenza identificabile che ha un significato per l'hardware o per il software.
Il creatore, che è la fonte dell'evento, sa che si è verificato l'evento.
I consumatori sono entità che hanno bisogno di conoscere è verificare l'evento; essi possono essere coinvolti nell'elaborazione dell'evento o possono semplicemente essere colpiti dall'evento stesso.
I consumatori di eventi in genere sottostanno a qualche tipo di middleware (event manager).
Quando il manager riceve la notifica di un evento da un creatore, inoltra l'evento a tutti i consumatori collegati.
Il vantaggio di un'architettura event-driven è che permette un gran numero di creatori e consumatori per lo scambio di informazioni di stato e di risposta in tempo quasi reale.

## Svantaggi con Arc. Multi-processo o Thread

Il modo più comune per  creare un'applicazione network e quello di assegnare per ogni thread o processo una connessione. Questa architettura e semplice da implementare, ma non è scalabile quando l'applicazione deve stabilire migliaia di connessioni simultaneamente.
L'uso di processi e di thread implica un consumo di risorse. Ognuno di essi utilizzano la memoria e altre risorse del sistema operativo, e hanno bisogno di  eseguire operazioni aggiuntive nell'attivare e la disattivazione dei core (denominata context switch).
La maggior parte dei moderni server è in grado di gestire centinaia di piccoli thread o processi attivi contemporaneamente, ma le prestazioni degradano seriamente una volta che la memoria si è esaurita o quando il carico di I/O provoca un grande volume di context switch.

N.B. *In informatica la **commutazione di contesto** (in inglese **context switch**) è un particolare stato del sistema operativo durante il quale avviene il cambiamento del processo correntemente in esecuzione su una delle CPU. (cit. it.Wikipedia.org)*

---

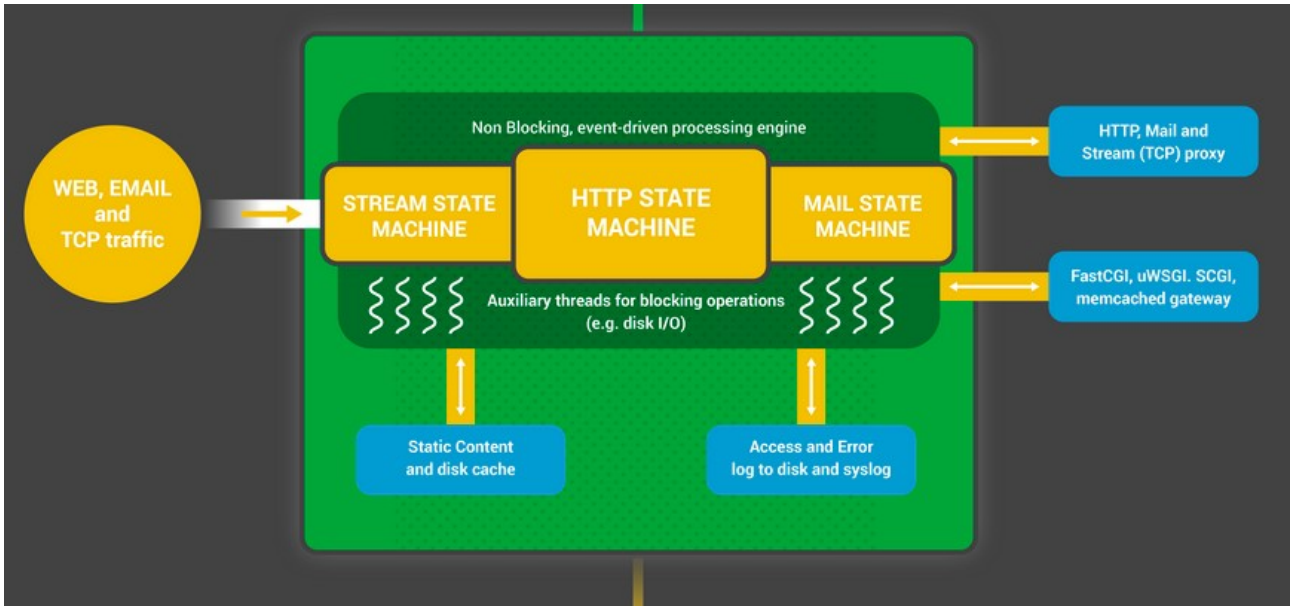
## Come funziona NGINX?

Nginx utilizza un modello di sotto processi  sintonizzati alle risorse hardware disponibili:

- Il processo **master** esegue le operazioni privilegiate come la lettura della configurazione e il vincolo delle porte, e quindi crea un piccolo numero di processi figli (i prossimi tre tipi).
- Il processo di **cache loader** viene eseguito all'avvio per caricare la cache situata nel disco in memoria, e poi esce. E basato su uno schedule conservativo, così le sue esigenze di risorse sono bassi.
- Il processo di **cache manager** viene eseguito periodicamente e rilascia parti della cache per mantenerla entro le dimensioni configurate.
- Il **worker proccess** fa tutto il lavoro! Gestiscono le connessioni di rete, leggere e scrivere i contenuti su disco, e comunicare con i server upstream.


---

# Dentro al processo Worker



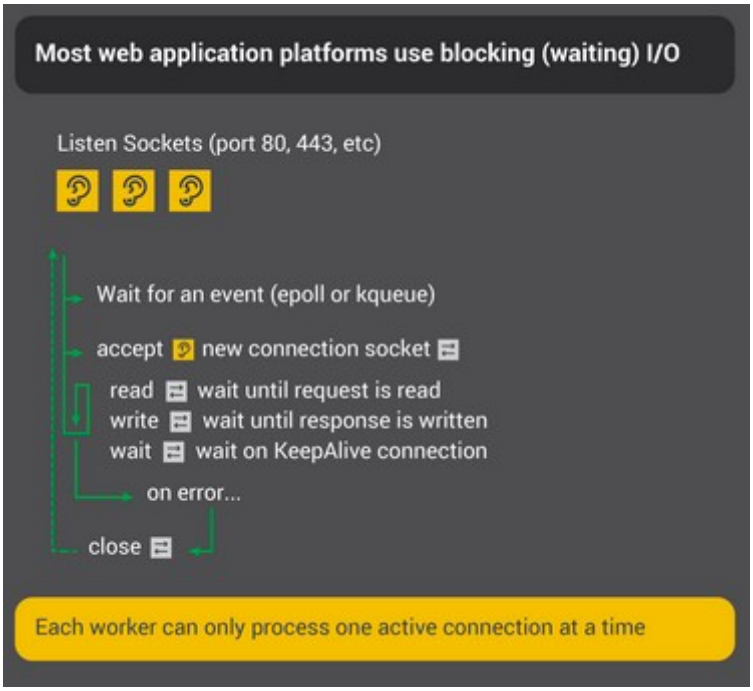

Per ogni processo worker di ENGINX vengono inizializzate le configurazioni e una seri di socket in ascolto per il processo principale.
I processi worker attendono eventi su socket listener (accept_mutex e kernel socket sharding). Gli eventi sono inizializzati come una nuova connessione. Queste connessioni sono assegate a una “state machine” - HTTP è lo stato più comune ma ne esistono altri: TCP (grezzo), FTP, SMTP IMAP e POP3.
Lo “state machine” è un set di istruzioni che dice a Nginx come processare le richieste effetuate al server.

---

# Temporizzazione (Scheduling) sullo “state machine”

## Esempio

Pensate allo “state machine” come alle regole per gli scacchi. Ogni transazione HTTP è una partita a scacchi. Su un lato della scacchiera è il server web - un maestro che può prendere decisioni molto rapidamente. D'altra parte è il client remoto - il browser web, relativamente lento, che accede al sito o applicazione su una rete.
La maggior parte dei server web e applicazioni web utilizzano una connessione processo-connessione o thread-connesione per giocare il gioco degli scacchi. Ogni processo o thread contiene le istruzioni per giocare una partita fino alla fine. Durante il tempo il processo è gestito dal server, passa la maggior parte del suo tempo 'bloccati' - in attesa per il cliente di completare la sua prossima mossa.

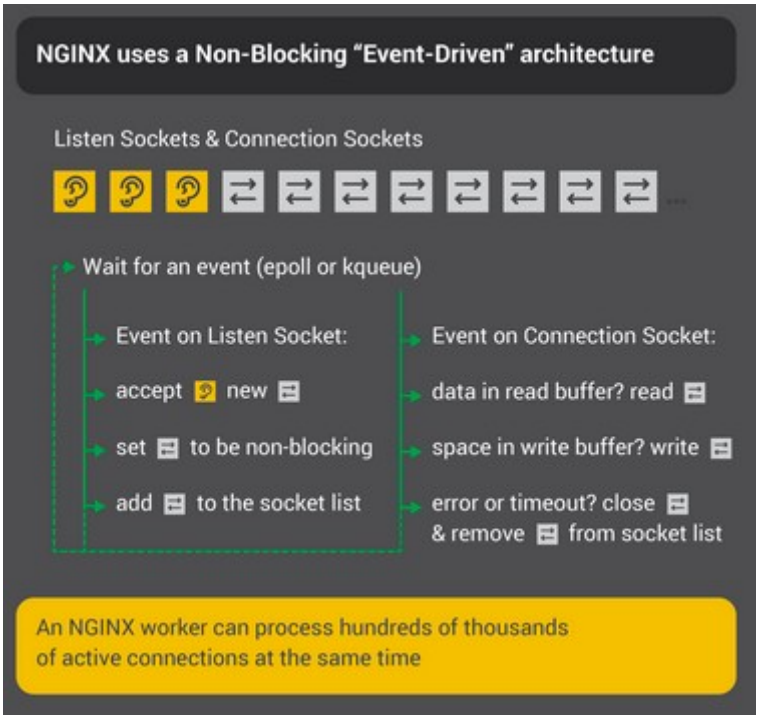



1. Il processo del server Web ascolta le nuove connessioni (nuovi giochi avviati da clienti) sulle prese di ascoltare.
2. Quando si ottiene un nuovo gioco, in quella partita, il processo si blocca dopo ogni mossa in attesa di una risposta del cliente.
3. Una volta completato il gioco, il processo del server web potrebbe attendere per vedere se il cliente vuole iniziare una nuova partita (questo corrisponde a una connessione keepalive). Se la connessione è chiusa (il cliente va via o si verifica un timeout), il processo del server web restituisce all'ascolto di nuovi giochi.

Il punto importante da ricordare è che ogni connessione HTTP attiva (ogni gioco di scacchi) richiede un processo dedicato o un thread (un maestro). Tuttavia, c'è un enorme squilibrio: la connessione HTTP piuttosto leggera, rappresentato da un descrittore di file, una piccola quantità di memoria, mappe di un thread o processo separato è un oggetto molto pesante per il SO. Si tratta di una convenienza di programmazione, ma è uno spreco in maniera massiccia.

## Nginx, il vero maestro di scacchi

Ecco come un processo worker di Nginx gioca (ricorda - di solito c'è un worker per ogni core della CPU) "a scacchi." È un grande maestro che può giocare centinaia (infatti, centinaia di migliaia) di giochi contemporaneamente.




1. Il worker attende gli eventi sulle prese di ascoltare e di collegamento.
2. Gli eventi si verificano sui socket e il worker li gestisce:
   - Un evento sul socket listener significa che un cliente ha iniziato una nuova partita a scacchi. Il worker crea una nuova presa di collegamento.
   - Un evento su un socket di connessione significa che il client ha fatto una nuova mossa. Il worker risponde prontamente.

Quindi un worker non si blocca su una sola connesione, in attesa del suo "avversario" (il client) per rispondere. Quando ha fatto la sua mossa, procede immediatamente ad altri giochi in cui mosse sono in attesa di essere processate, o accoglie i nuovi giocatori con altri worker (uno per ogni CPU).

Fonti:
https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/
http://searchsoa.techtarget.com/definition/event-driven-architecture
https://it.wikipedia.org/wiki/Scalabilità
https://it.wikipedia.org/wiki/Commutazione_di_contesto