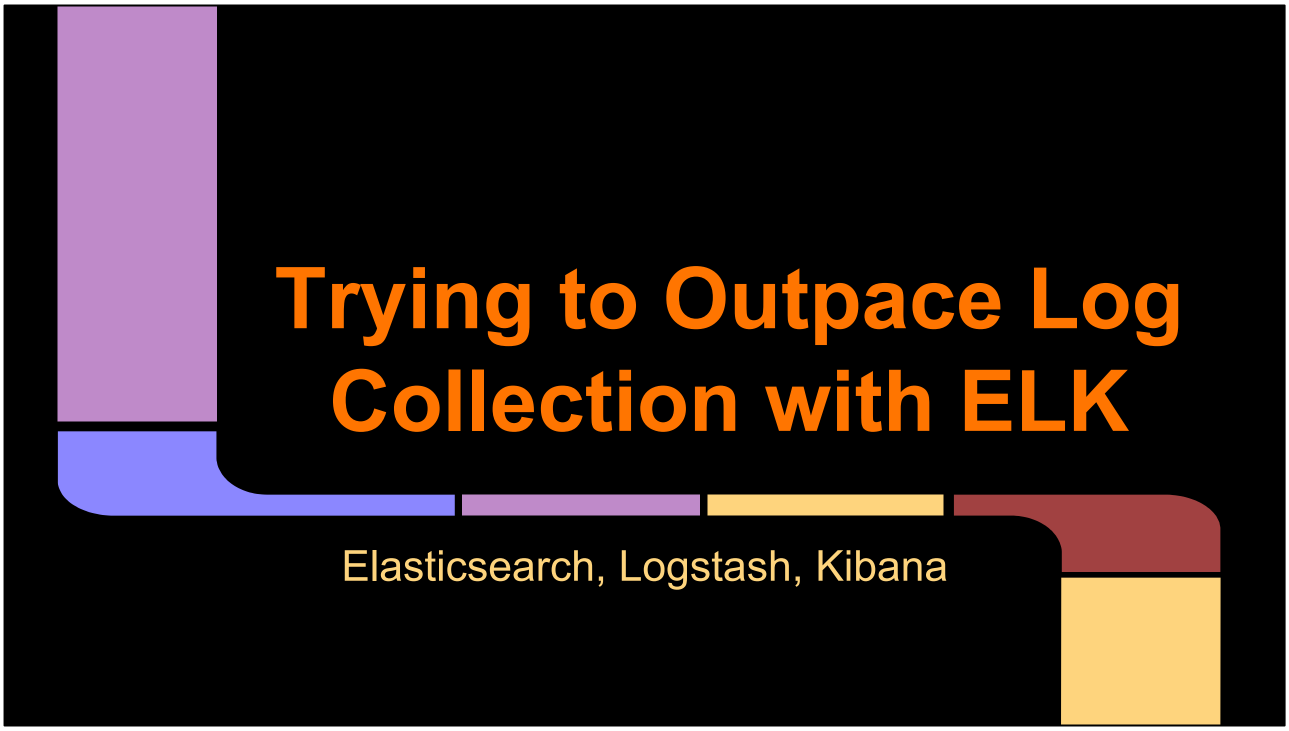


Trying to Outpace Log Collection with ELK

Elasticsearch, Logstash, Kibana

Time: 0:10

Hi everyone! I’m Neil Schelly, a sysadmin up at Dyn in New Hampshire. I’ve been focused for the last year on a project to get centralized logging in place for our network. Here I am to share a bit of that experience with all of you.

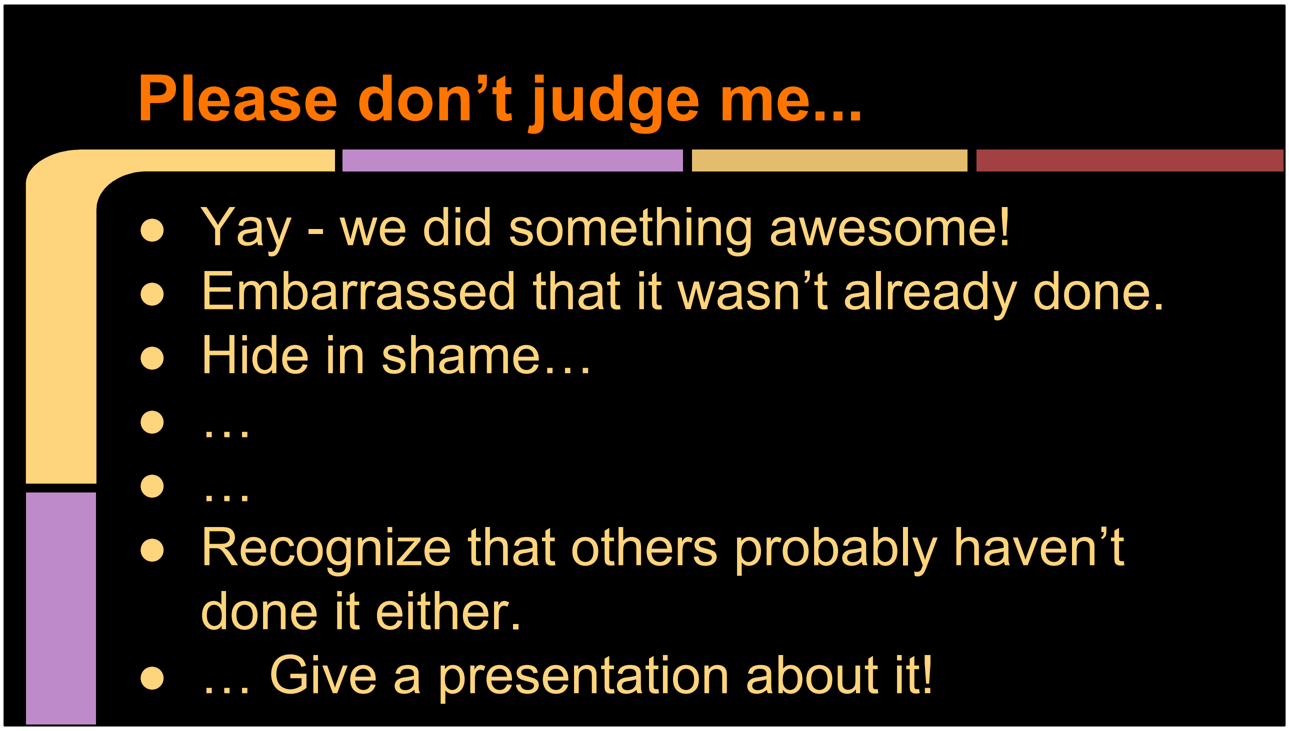
Disclaimer: Judgement Free Zone

https://www.flickr.com/photos/coconinonationalforest/5376143040

Time: 0:15

Disclaimer: This is a judgement free zone, and we'll get to why in a bit. I also have a penchant to browse Flickr for creative commons photos for presentation slides. I'm sorry in advance.

I am not sure what this guy's job is once he finds a loose chain, and I'm pretty sure he won't be able to run fast enough if he does.



Please don't judge me...

- Yay - we did something awesome!
- Embarrassed that it wasn't already done.
- Hide in shame…
- …
- …
- Recognize that others probably haven't done it either.
- … Give a presentation about it!

Time: 0:15

- Yay - we did something awesome!
- And then… We're slightly embarrassed that it wasn't already done.
- Kinda want to hide in shame.
- Hmm…
- Eventually recognize that others probably also haven't done it yet.
- Presentation!

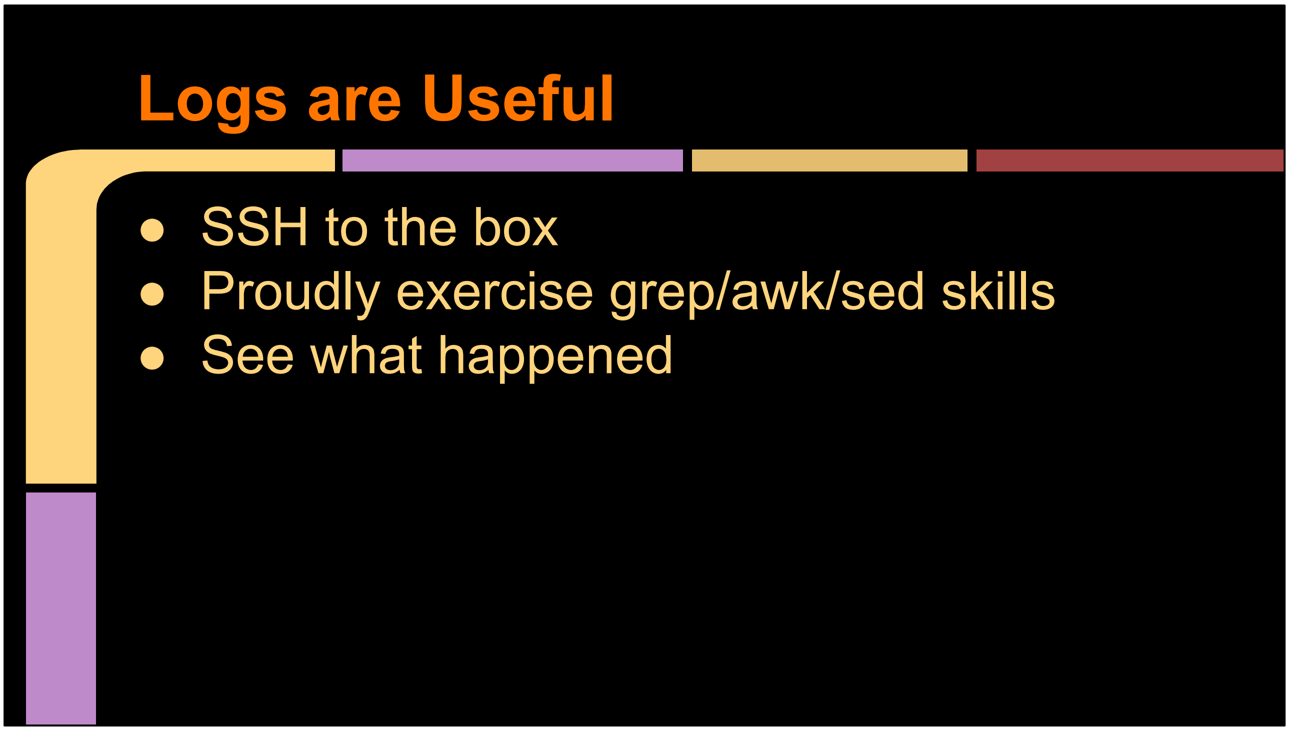
I feel like there should be a bullet point for profit in here somewhere, but I haven't figured out how that works yet.

The Problem

https://www.flickr.com/photos/iangbl/338035861

Time: 0:05

Starting with a problem statement… We have logs. Lots of logs. Don’t care about all of them, but definitely care about some of them.



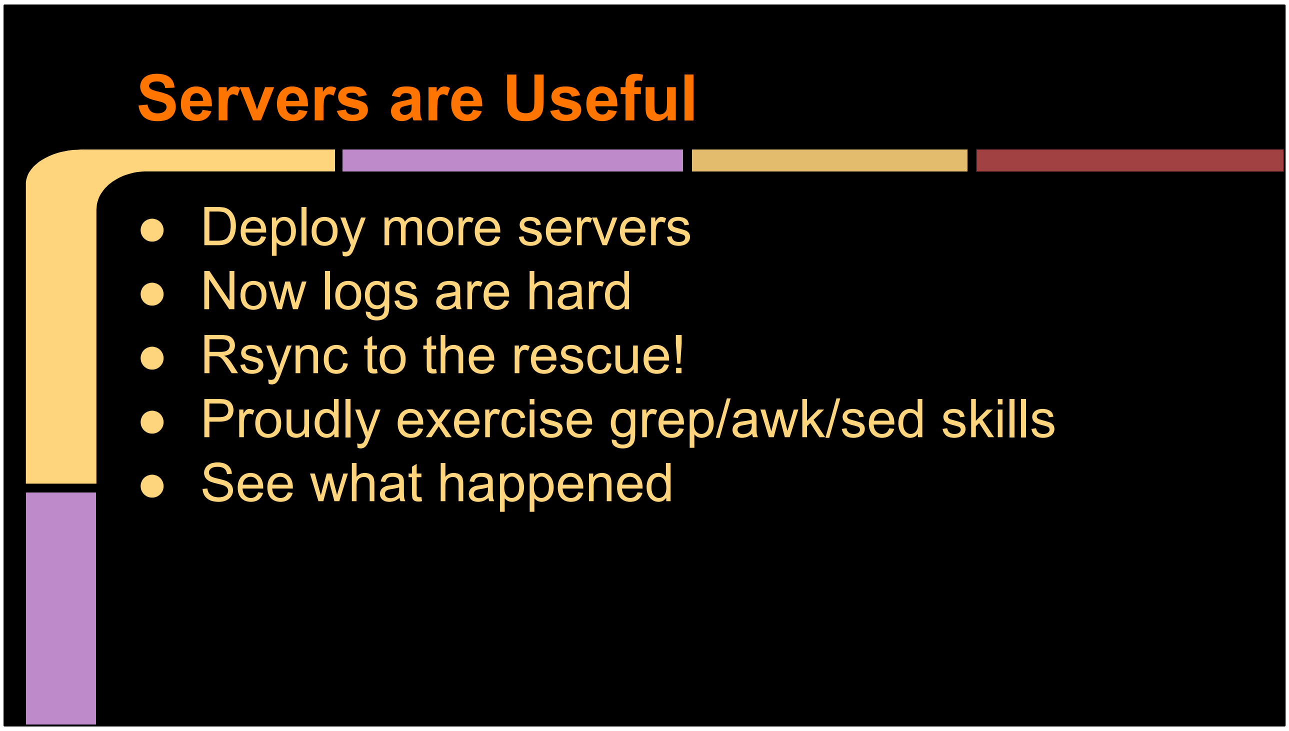
Logs are Useful

- SSH to the box
- Proudly exercise grep/awk/sed skills
- See what happened

Time: 0:10
Logs are useful

We know this. We've all administered something like this.

SSH in, do some grep-fu, and we know what happened!



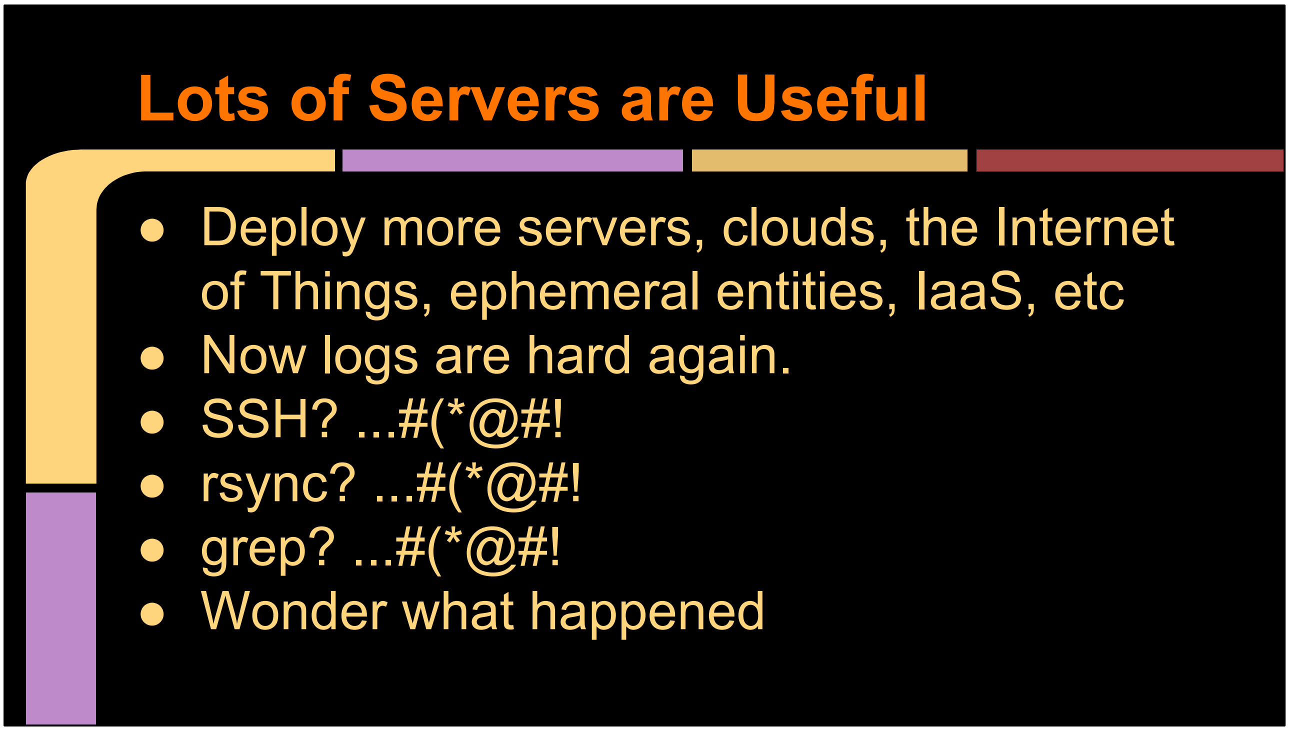
Servers are Useful

- Deploy more servers
- Now logs are hard
- Rsync to the rescue!
- Proudly exercise grep/awk/sed skills
- See what happened

Time: 0:10
Multiple servers are also useful.

But the same approach can work well with some rsync thrown in.

The magic of rsync only buys so much time before...



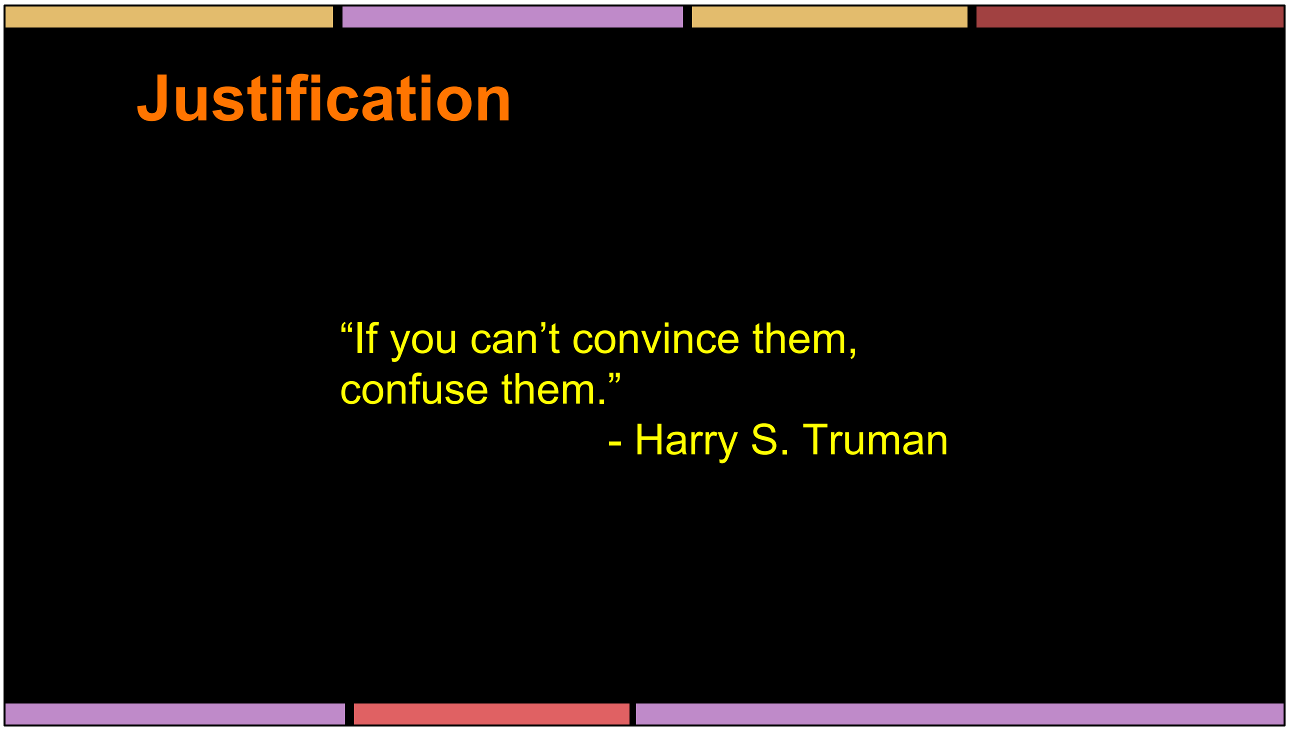
Lots of Servers are Useful

- Deploy more servers, clouds, the Internet of Things, ephemeral entities, IaaS, etc
- Now logs are hard again.
- SSH? ...#(*@#!
- rsync? ...#(*@#!
- grep? ...#(*@#!
- Wonder what happened

Time: 0:15

Then we moved on to more servers, clouds, the Internet of Things, ephemeral entities, IaaS, etc…

Complex systems are more complex. Logs got hard again. At this point, we just end up wondering what happened.



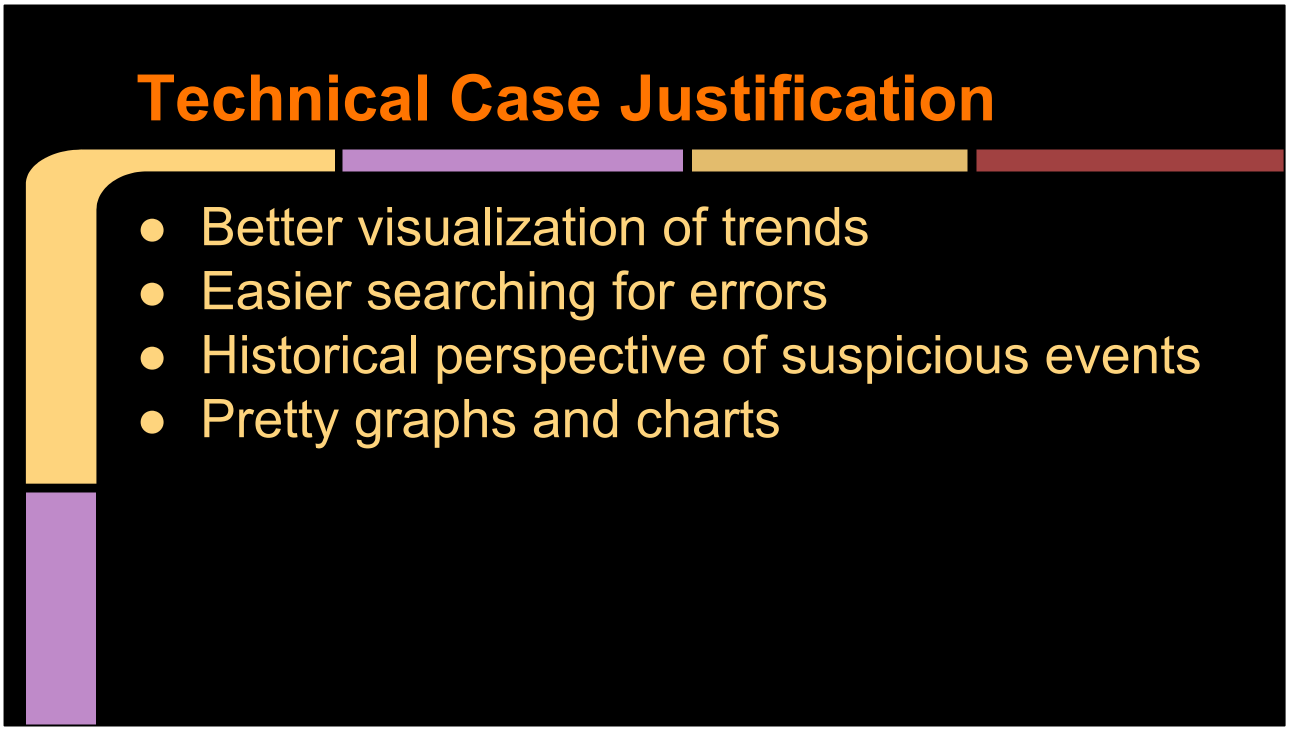
Justification

“If you can’t convince them,
confuse them.”
- Harry S. Truman

Time: 0:05

So we know we want to create a centralized place to monitor the whole system. How do you justify it?

In case President Truman’s advice doesn’t work for you, I’ll show what worked for us.



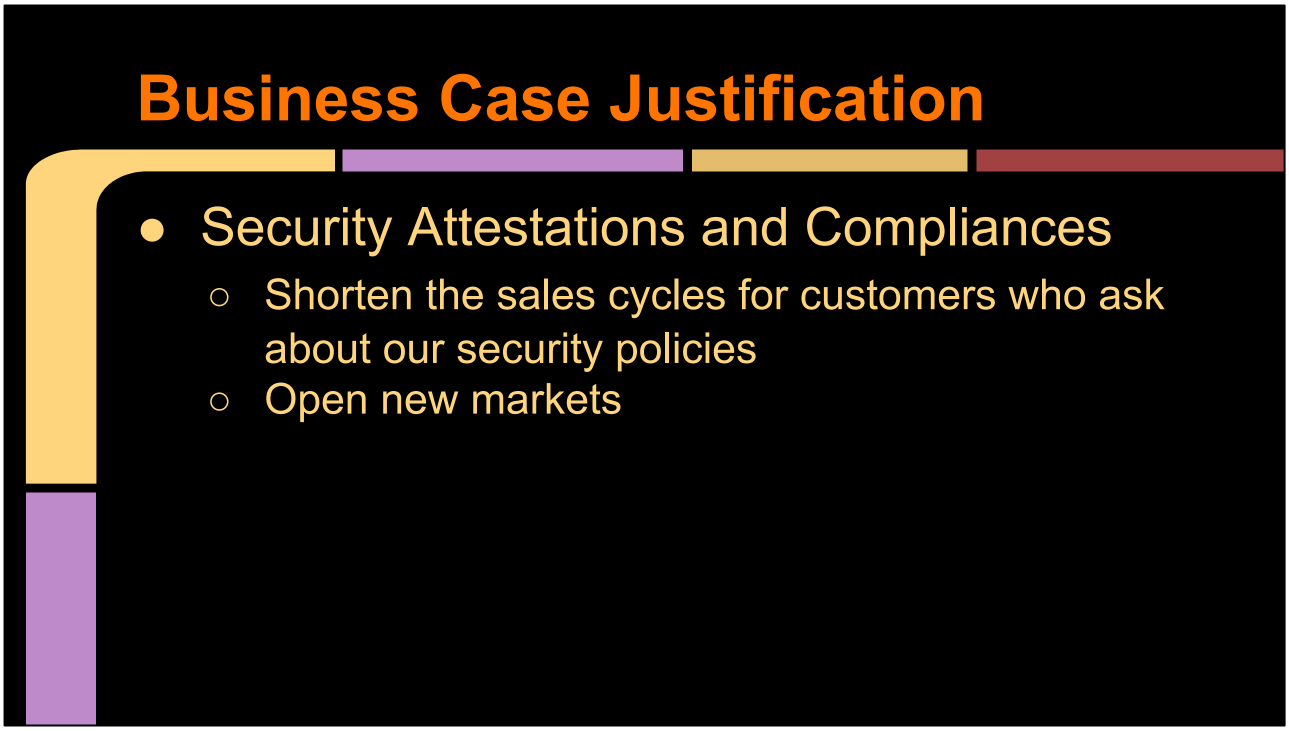
Technical Case Justification

- Better visualization of trends
- Easier searching for errors
- Historical perspective of suspicious events
- Pretty graphs and charts

Time: 0:15

The technical case is easiest. Never underestimate the value of pretty graphs and charts. We had all these justifications long before this project got approval to happen.

It doesn't make money. It doesn't scale the product to support more people, which would make money. It helps the people support the product more easily as more customers use the system. That's pretty indirect to the folks who sign the checks.



Business Case Justification

- Security Attestations and Compliances
  - Shorten the sales cycles for customers who ask about our security policies
  - Open new markets

Time: 0:20

Here's how we convinced folks to sign checks. This is still a judgement-free zone in case you forgot.

- Security attestations and compliances generally require you to prove you are paying attention to what's going on with your systems. Auditors are really impressed by centralized logging systems.
- Shorten sales cycles when customers ask questions about security policies
- Open new markets to customers who demand certain certifications

These justifications may work for you. YMMV.

The State of the Logs

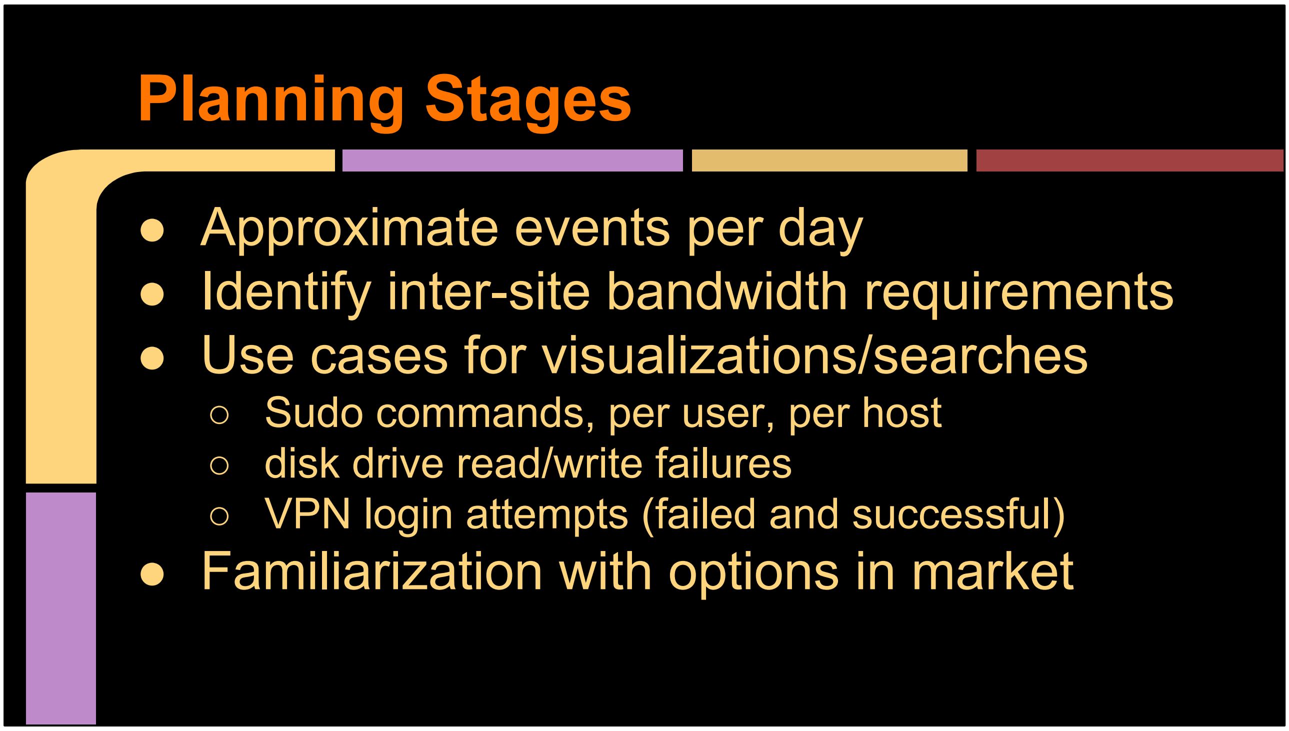
https://www.flickr.com/photos/alanenglish/3509549894

The Prototype

https://www.flickr.com/photos/laffy4k/404321726

Time: 0:06

So we've got some ideas and we've got some justification to explore them and some mandates to start paying better attention to logs. It's time to start playing.



Planning Stages

- Approximate events per day
- Identify inter-site bandwidth requirements
- Use cases for visualizations/searches
  - Sudo commands, per user, per host
  - disk drive read/write failures
  - VPN login attempts (failed and successful)
- Familiarization with options in market

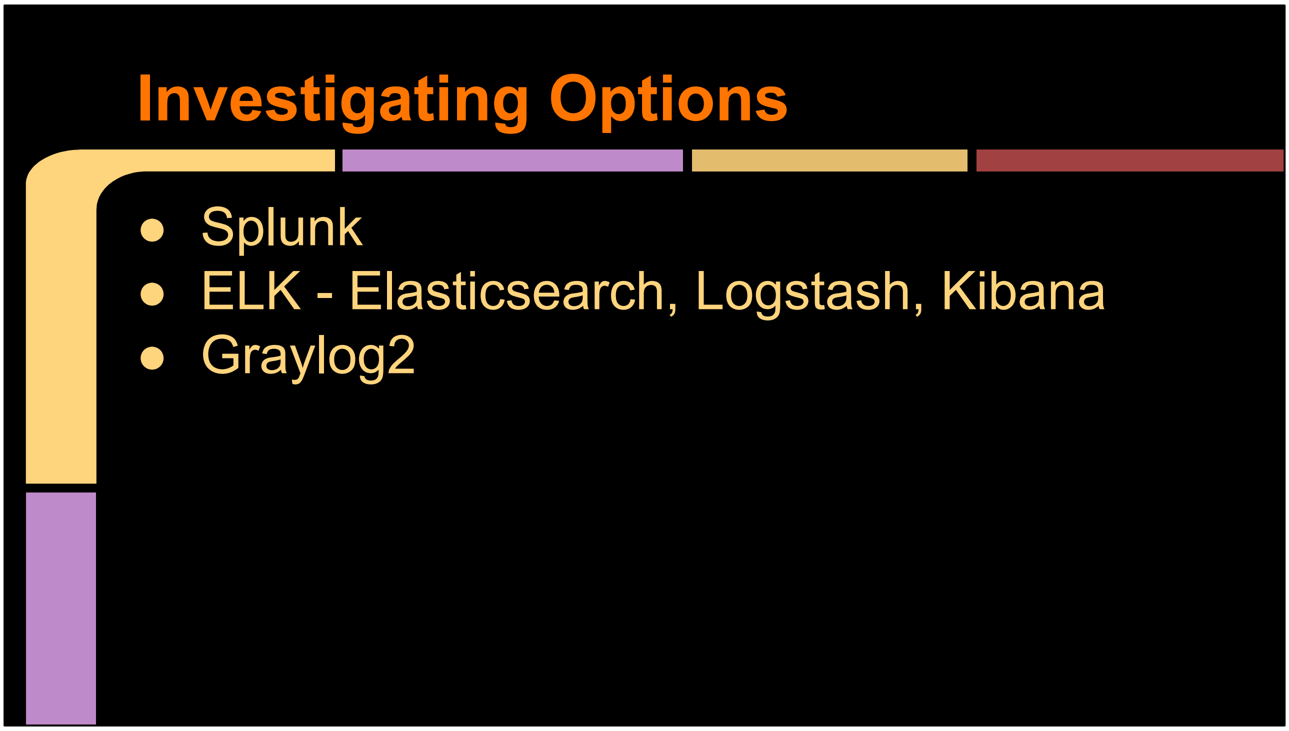
Time: 0:25

For the purposes of planning scale, you want to know how many events you want to handle, how much space that will take, etc.

- Look for places where logs won't be evenly distributed in your network
- Find out whether some log sources will have larger messages than others in any significant way
- This can help in predicting constraints that should be explored.

Come up with use cases. There should be some information that you already know you want to look for once the logs are all searchable in one place. If not, you probably don't yet need this project on your plate.

Finally, look at the options in the market people are using to solve these problems. Come up with the options.



Investigating Options

- Splunk
- ELK - Elasticsearch, Logstash, Kibana
- Graylog2

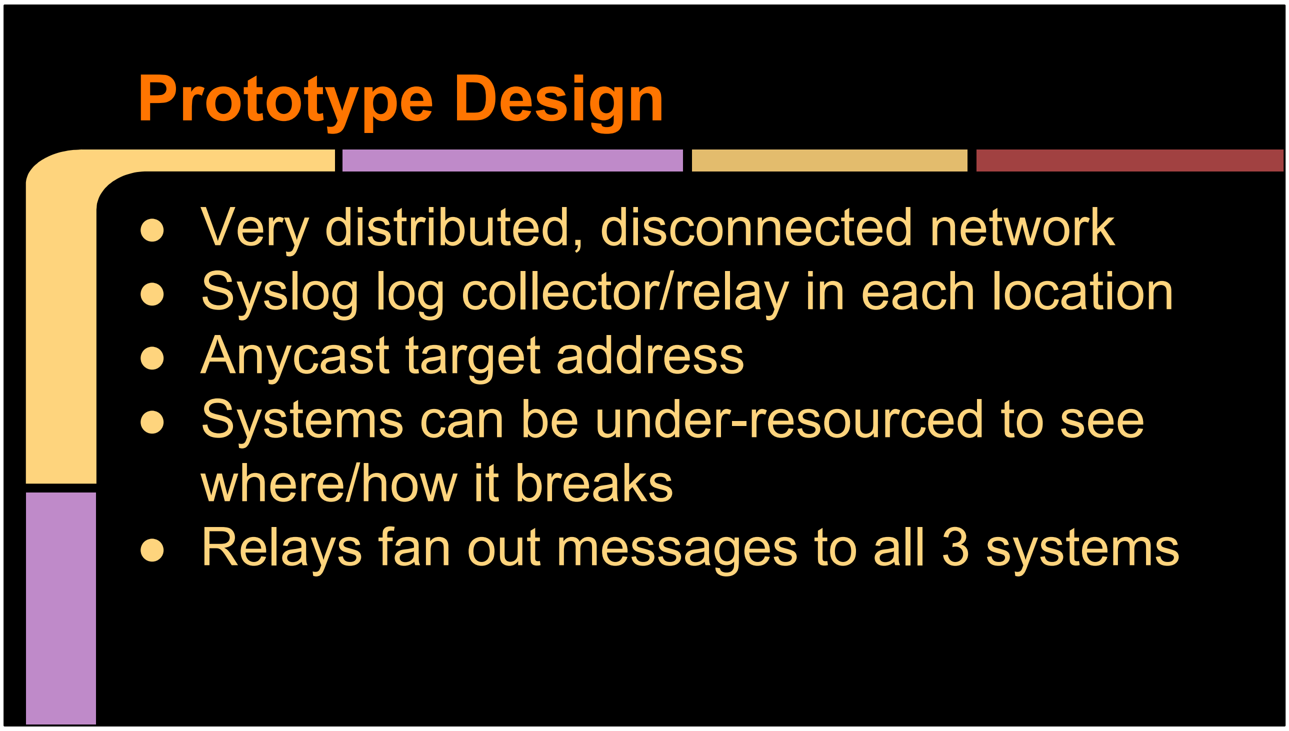
Time: 1:00

Once you get to look at the market, you get into 3 primary options out there.

Splunk is the 800lb gorilla in the market. Structurally, Splunk is a collection of data nodes and agents running on machines that tail log files or watch for traffic on listening ports or something along those lines. It's all configured in the web interface. The data can be distributed amongst all the data nodes.

ELK is the Elasticsearch, Logstash, Kibana combination that the Elasticsearch company offers as their solution. Logstash is the ingestion and parsing piece of the puzzle with listening ports or pulling in data from other sources or tailing log files, etc. It will process the events, parse out any fields for specific indexing or searching statistics, filter and modify events as desired, and deliver parsed events downstream. Elasticsearch is a cluster that fulfills the search engine/indexing piece. It ingests JSON documents, allows keyword searching, field indexing, statistics aggregation, etc. Kibana is a web application entirely in HTML and CSS and Javascript that requests information from the Elasticsearch HTTP REST API and displays it in interesting ways to the end user.

Graylog2 is a master/slave cluster that acts as an application server frontend for Elasticsearch. That daemon is responsible for configuring listening ports for incoming data, ingesting data on those ports into an Elasticsearch cluster, and providing access to the data via a web interface within that application server.



Prototype Design

- Very distributed, disconnected network
- Syslog log collector/relay in each location
- Anycast target address
- Systems can be under-resourced to see where/how it breaks
- Relays fan out messages to all 3 systems

Time: 0:20

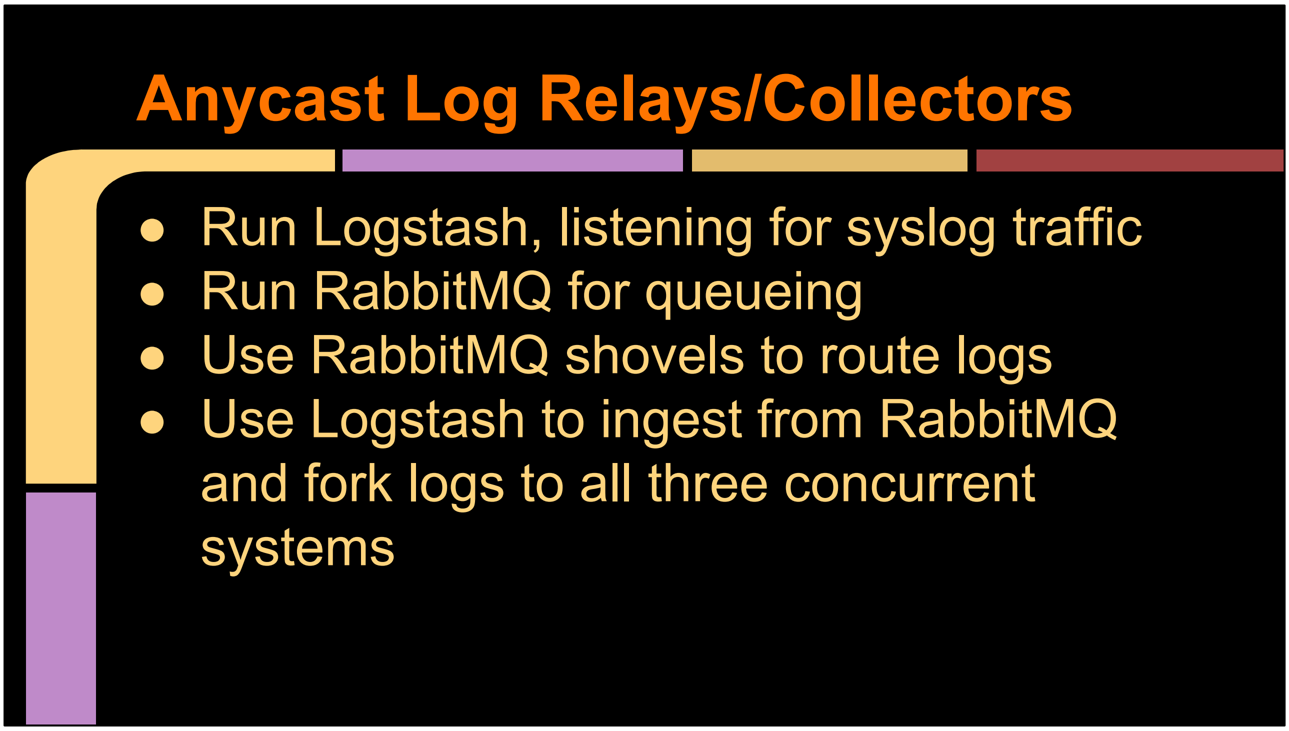
For our prototype design, we came up with something like this.

Our network is very distributed and disconnected, so we setup a collector to relay logs in each site.

It’s available at an anycasted address so every machine in the network can send to the same name/IP.

Systems can be under-resourced so you can find your pain points in the prototype stages.

Eventually, we wanted messages to fan out to all 3 systems to get familiar with them all



Anycast Log Relays/Collectors

- Run Logstash, listening for syslog traffic
- Run RabbitMQ for queueing
- Use RabbitMQ shovels to route logs
- Use Logstash to ingest from RabbitMQ and fork logs to all three concurrent systems

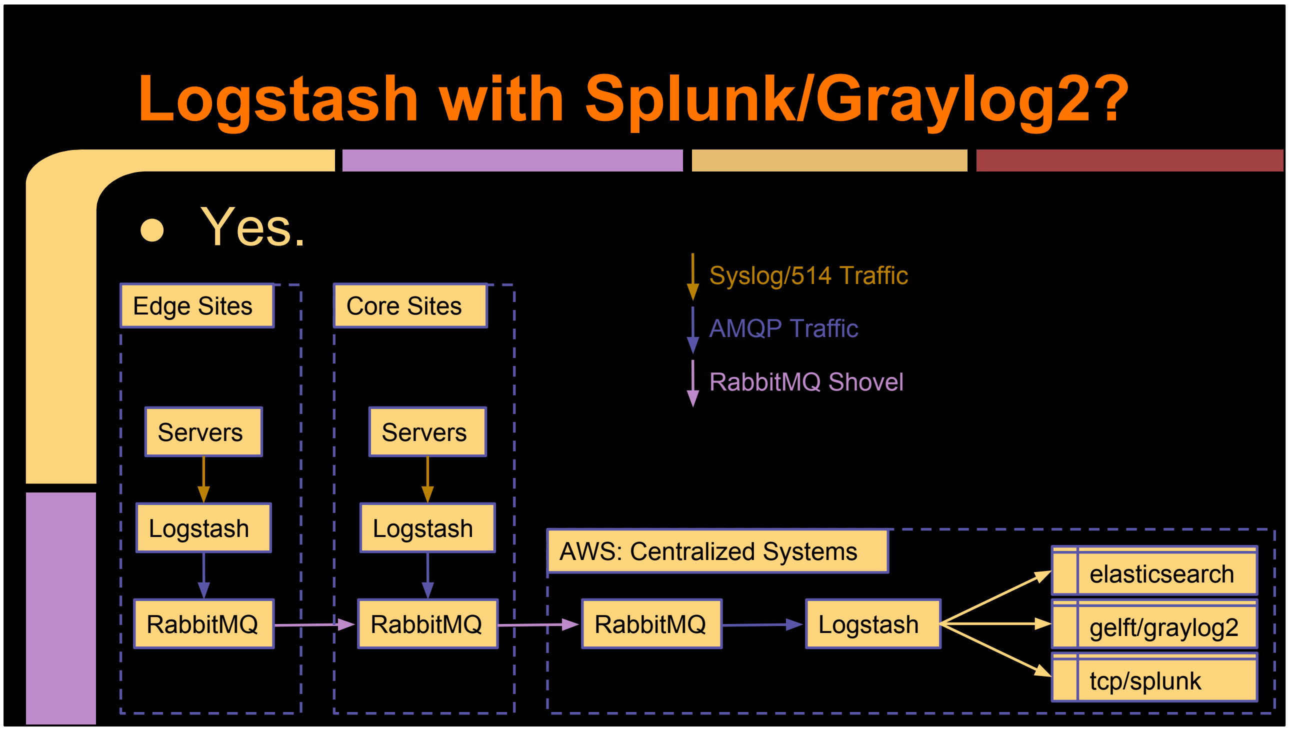
Time: 1:00

So on our prototype relays, we have Logstash running and listening on port 514 sockets for syslog traffic. It parses those logs into JSON and send them to a local instance of RabbitMQ. Once the messages are in a queue, we're using RabbitMQ's shovel plugin to move those logs to queues on other relay machines.

The shovel is a dedicated AMQP client thread that will run inside the RabbitMQ process's Erlang virtual machine. That client is designed to do simple things like read from one queue and publish to another. In our infrastructure, we built the big central parts of these systems in AWS, and most of our edge sites cannot actually route to it. They can all route to our core datacenters, and those core datacenters can get to our AWS instances.

Our edge sites' RabbitMQ daemons have shovels that pull messages off the queue and deliver them to the RabbitMQ systems in our core sites. Those RabbitMQ systems have shovels that pull messages off the queue and deliver them to the centralized logging systems in AWS.

For the purposes of the prototype, Logstash re-ingests the messages from RabbitMQ and forks out a copy of each to Splunk, Graylog2, and Elasticsearch for ELK. It's very easy to setup multiple outputs for Logstash.



Logstash with Splunk/Graylog2?

- Yes.

Time: 0:50

So that prototype design uses Logstash for everything, and that's misleading a bit.

On the relays, Logstash is listening for syslog traffic and outputting to RabbitMQ. The messages travel through AMQP to get to the centralized logging servers in AWS. From there, another Logstash process is pulling messages from RabbitMQ and outputting those messages to all 3 destinations. It's using the Elasticsearch API to send to the Elasticsearch cluster in a Kibana-friendly index, using GELF to talk to the Graylog2 server, and using raw TCP to send JSON-formatted events to Splunk.

Quick note if you're trying to accomplish a similar test setup with Splunk. You really want Logstash to separate events with newlines using the json_lines codec. If you don't set that, then it's rather complicated to get Splunk to differentiate one event from the next.

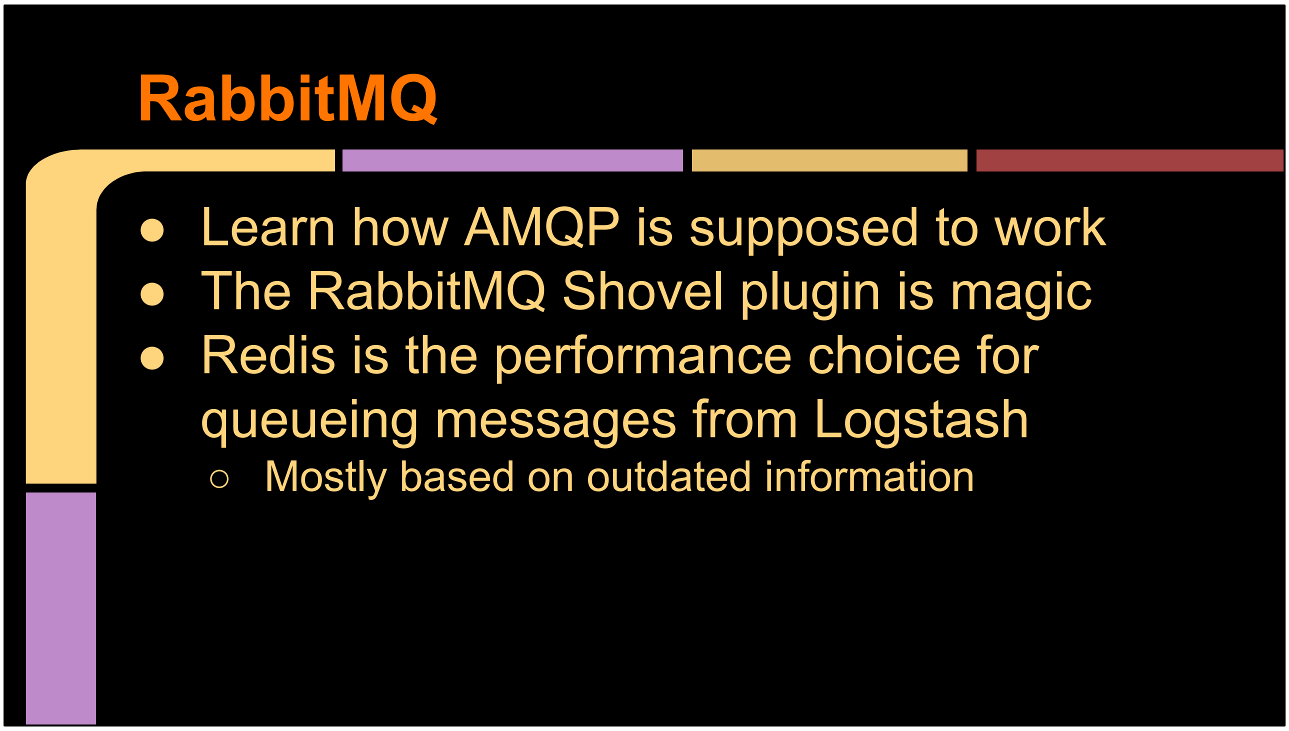
This is all valid because we were comparing the different means of searching the logs, costs of running the systems, effort to scale the systems, whether or not we could find the information our use cases defined, etc. If we went with something other than ELK, we could have re-architected without Logstash, but honestly, I might have kept it for its flexibility at this stage. There are definitely reasons you might want to use Logstash with all three systems. It's a useful tool for stream-processing of events. Logstash is really just a collection of inputs, modifiers/filters, and outputs, which you can read about at their docs site.

Things I Learned

https://www.flickr.com/photos/pfly/1537122018

Time: 0:05

So then, I learned stuff!



RabbitMQ

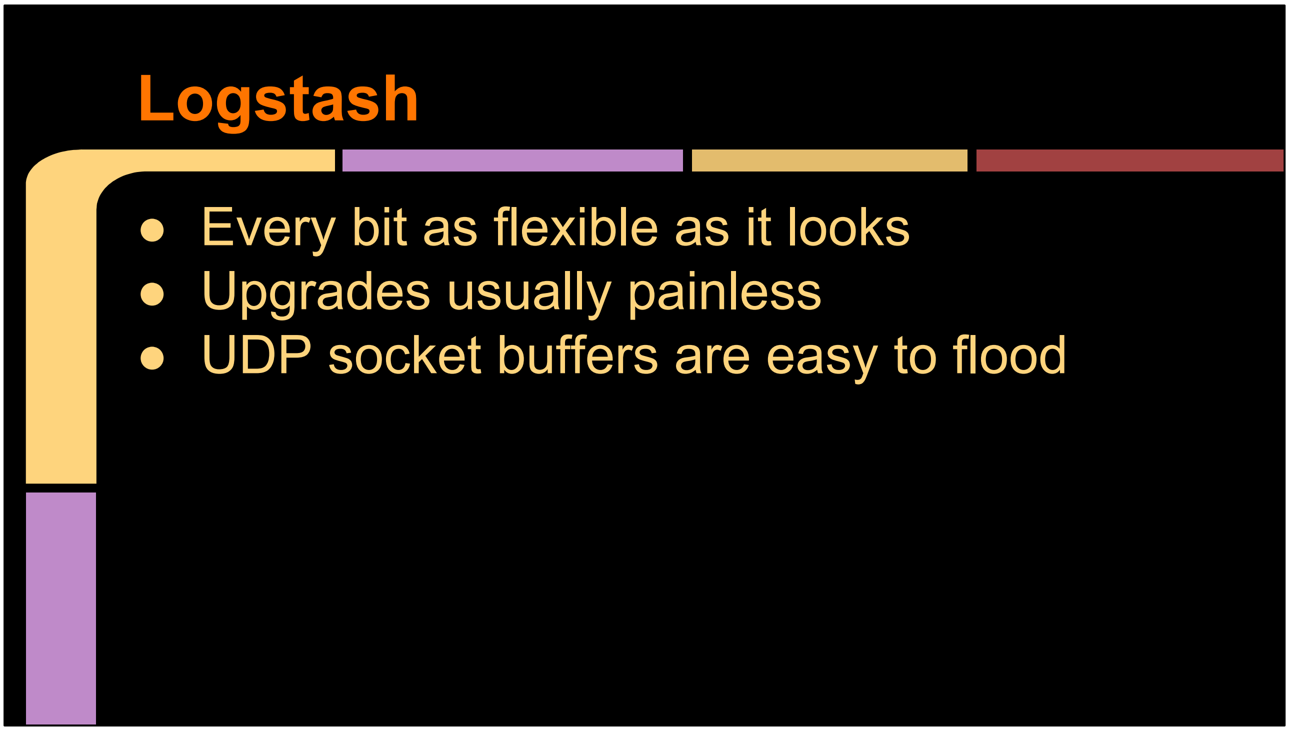
- Learn how AMQP is supposed to work
- The RabbitMQ Shovel plugin is magic
- Redis is the performance choice for queueing messages from Logstash
  - Mostly based on outdated information

Time: 0:45

RabbitMQ is the reference implementation of the AMQP protocol. You can get RabbitMQ working quickly out of the box. It's really easy, even if you've no idea what AMQP is or how it works. Eventually, you'll want to just read their documentation about what AMQP does for message passing, what exchanges and routing keys and queues are, and how the different types of exchanges and queues work. They have really great documentation on that. Some of their documentation about interacting with their REST API is a bit tough to follow, but their system diagram documentation is really helpful.

Since I'm using RabbitMQ to move messages around the network pretty quickly without any processing at each step, the RabbitMQ Shovel plugin is awesome. RabbitMQ is an Erlang daemon, so you can use this plugin to create a dedicated thread in the Erlang VM that is just a very simple AMQP client that reads from one queue and publishes to another exchange. It can read/write to local or remote RabbitMQ daemons.

There's a lot of information out there that points to Redis being the more performant queueing option for Logstash, but it really didn't fulfill my needs of controlling the routing of messages at layer 7 or using SSL. I didn't really investigate it much. More recent news is that it does support SSL, but the RabbitMQ interfaces in jRuby (used by Logstash) are a lot faster than they once were too. RabbitMQ is not a constraint on my systems anyway.



Logstash

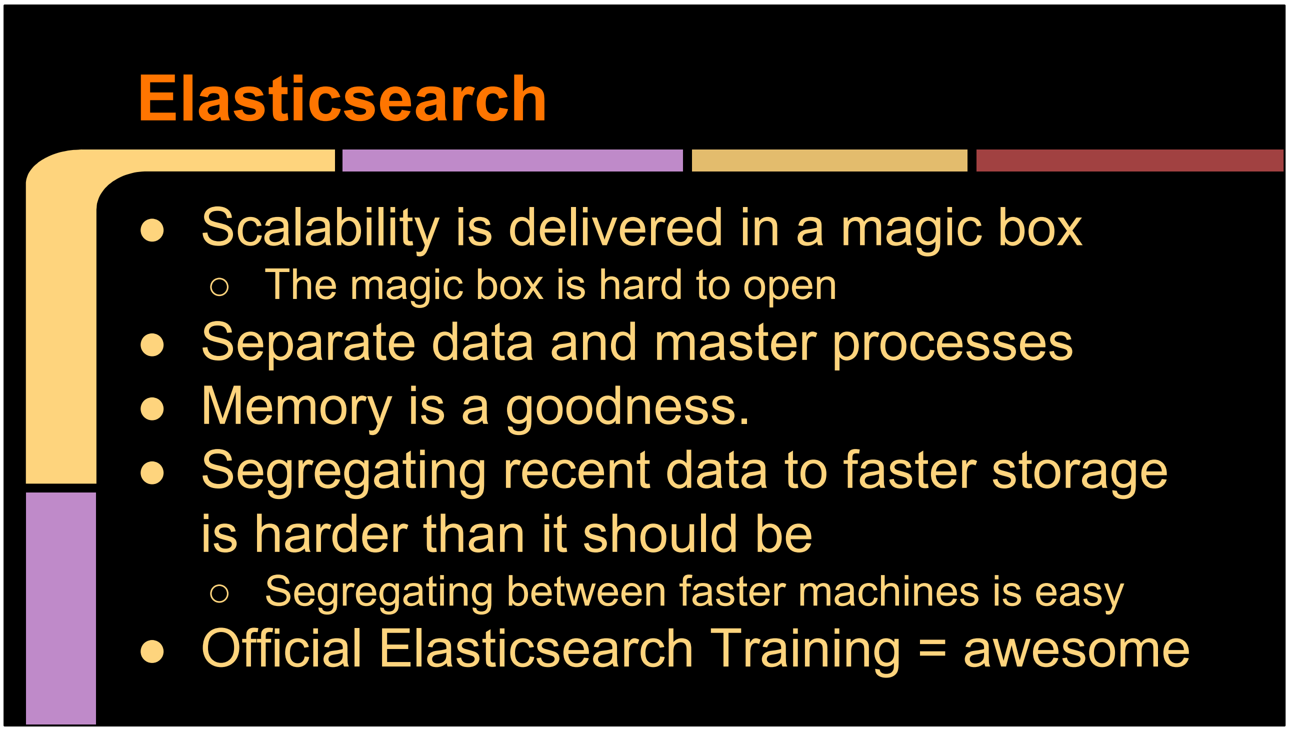
- Every bit as flexible as it looks
- Upgrades usually painless
- UDP socket buffers are easy to flood

Time: 0:30

Logstash is every bit as flexible as it looks, and it really is improving at a remarkable pace. It's been stable and reliable for many versions, despite undergoing significant changes. You don't have to keep up with their update schedule, but you will probably find it's not that hard.

If you're pummeling a Logstash daemon with tons of UDP syslog messages, you will eventually find your socket buffers overflowing, especially due to the burstiness of sending log messages. Increasing the default socket buffer sizes to around 8MB fixed this on all but my busiest relays. Some will just need more machines to properly divide the load.

To further help this out, I setup Logstash with UDP syslog listeners on multiple ports and used iptables rules to round-robin incoming port 514 packets to all the listening sockets.



Elasticsearch

- Scalability is delivered in a magic box
  - The magic box is hard to open
- Separate data and master processes
- Memory is a goodness.
- Segregating recent data to faster storage is harder than it should be
  - Segregating between faster machines is easy
- Official Elasticsearch Training = awesome

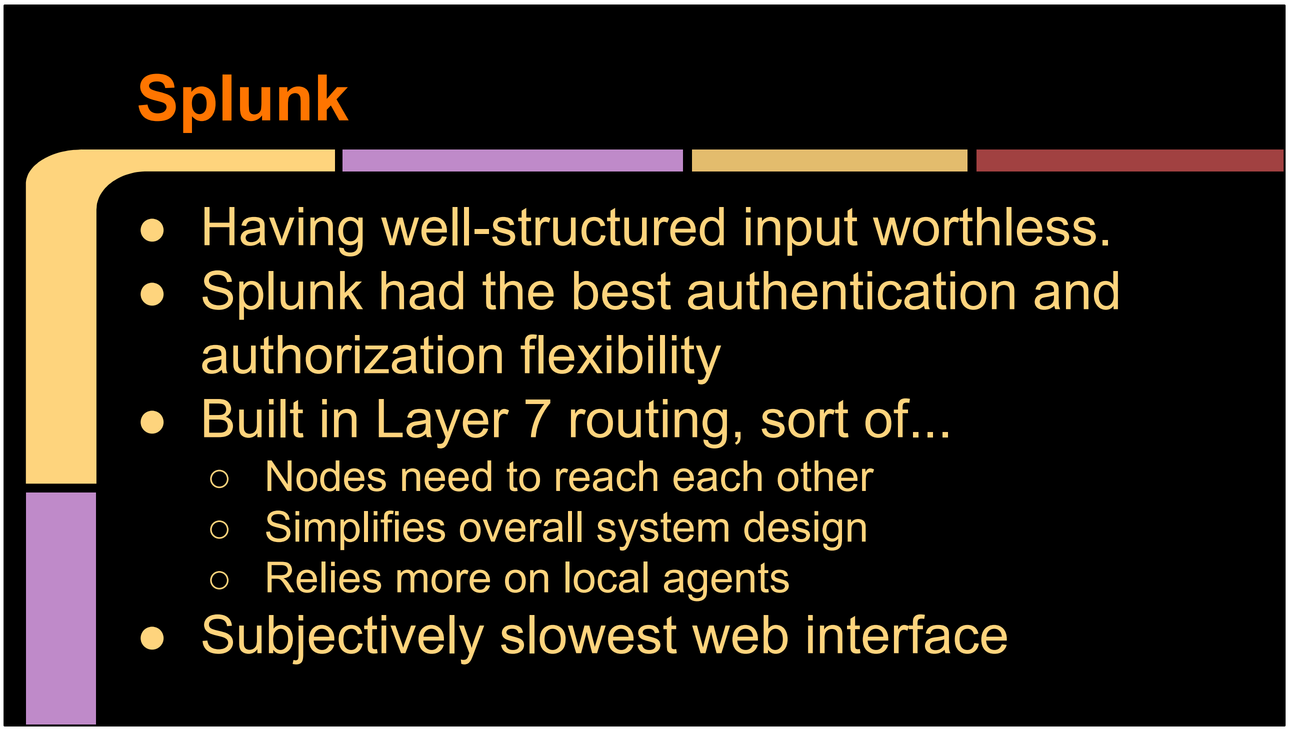
Time: 0:45

Elasticsearch is a magic black box. You give it data, and it generally gives it back to you when you search for it. Out of the box, you'll get a working system very easily, but there are many things to change before it enters production to ensure the cluster will remain stable and reliable.

Couple of important parts:

- Separate data-only and master-only nodes is a very important point, having to do with JVM garbage collection. I'll detail it more later.
- Learn about how to speed up certain searches that are important to you by making sure certain fields are parsed out of log messages by Logstash before the JSON documents are submitted to Elasticsearch for indexing. Keyword searching is great for finding things, but not for visualizing in structured forms, and it's not the fastest thing in the world with a ton of data.
- A little familiarity with using the REST API is critical to really managing Elasticsearch at scale. It's a very easy API to understand. A little Python with the Requests library and some JSON parsing was about all I needed to accomplish nearly anything.
- Memory is a goodness.
- I expected to have fast and slow storage available on my nodes and move older data to the slower drives. I made it work, but it would have been easier

- to let Elasticsearch do it at the node level instead by creating fast and slow nodes, and letting it move data to the slower nodes as it aged.
- The Elasticsearch Training is very well worth it, and they probably have a session in Boston or NYC in the near future, so you don’t have to travel far for it. I learned a lot, not just from very competent engineer instructors, but from the experiences of other folks in the training as well.



Splunk

- Having well-structured input worthless.
- Splunk had the best authentication and authorization flexibility
- Built in Layer 7 routing, sort of...
  - Nodes need to reach each other
  - Simplifies overall system design
  - Relies more on local agents
- Subjectively slowest web interface

Time: 2:00

You can talk to Splunk's sales team and get a free trial. You'll want it if you want to play with it and see how to set it up. The sales and sales engineering folks I worked with didn't really know a ton of technical details about it or anything about the competitors.

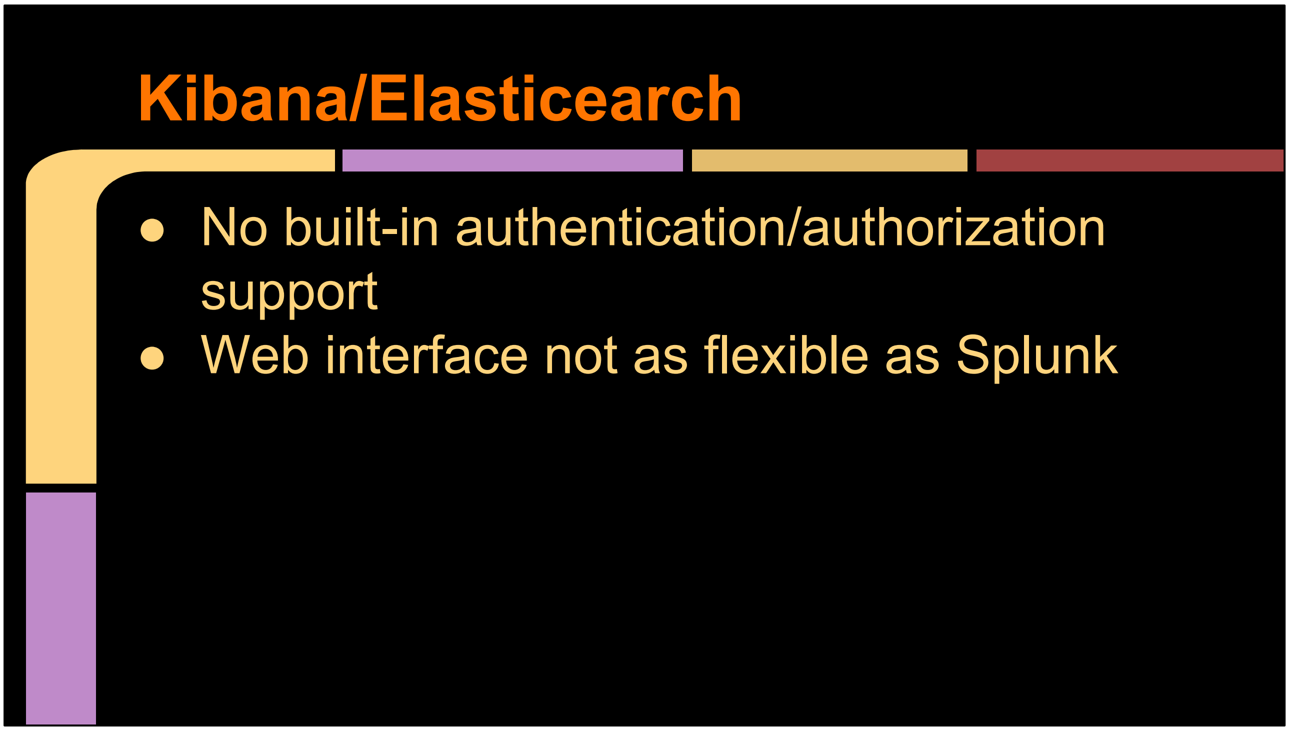
They will insist that data isn't indexed until you search it. It's definitely true that data is ingested as just raw text and you can reconfigure Splunk to interpret the data differently after the fact. And it doesn't take Splunk too long to make those changes visible to you and start giving out results for searches. That said, you can also create saved searches that will pre-seed the search indexes for certain patterns that you know you know you'll need for certain reports. One thing that is clear is that Splunk doesn't care for the purposes of my prototype that all the data was already well-formed JSON documents. Splunk focuses far more on just having the data, and letting you manipulate it at search time with filters, regular expressions, and other commands.

They have a 100-page instruction manual that shows you exactly how to use their search syntax and plugin system to generate the report you want. They use pipe symbols to link multiple filters/modifiers/etc eventually into a description of a chart type and options. It's complicated, but very powerful.

Splunk has easy integration with LDAP and other authentication backends. You can restrict types of data, reports, configuration, etc to specific users and groups with very fine-grained ACLs for every function you can imagine in the Splunk interface. If you want security out of the box that gives you the flexibility to restrict access to data in your centralized logging system, there is no other option than Splunk. Graylog2 supports some of this, and ELK doesn't have any of it, though you could certainly implement some simpler security in ELK.

Splunk expects all the nodes to be able to reach each other. A search head needs to be able to pull in and merge data from all the data nodes in the cluster, wherever they are. This makes it very flexible when it comes to making sure data is well-distributed against failures of specific machines. Adding more machines is easy and free, since there is no per-machine licensing cost.

And yes, since I brought up licensing, I'll be very clear that Splunk was good, but it was expensive. Our installation would have required a 10x budget increase to utilize Splunk. Literally, we could add up our costs to run the search/indexing infrastructure in EC2 on 3-year reserved instances and lots of EBS storage, and then throw another 0 on the end to year over year cover the licensing costs for that 3 years. They don't care about the number of machines you run on, or the amount of data you store. It's entirely based on how much data you ingest each day, letting you manage the storage needs for whatever retention you want on your own.



Kibana/Elasticsearch

- No built-in authentication/authorization support
- Web interface not as flexible as Splunk

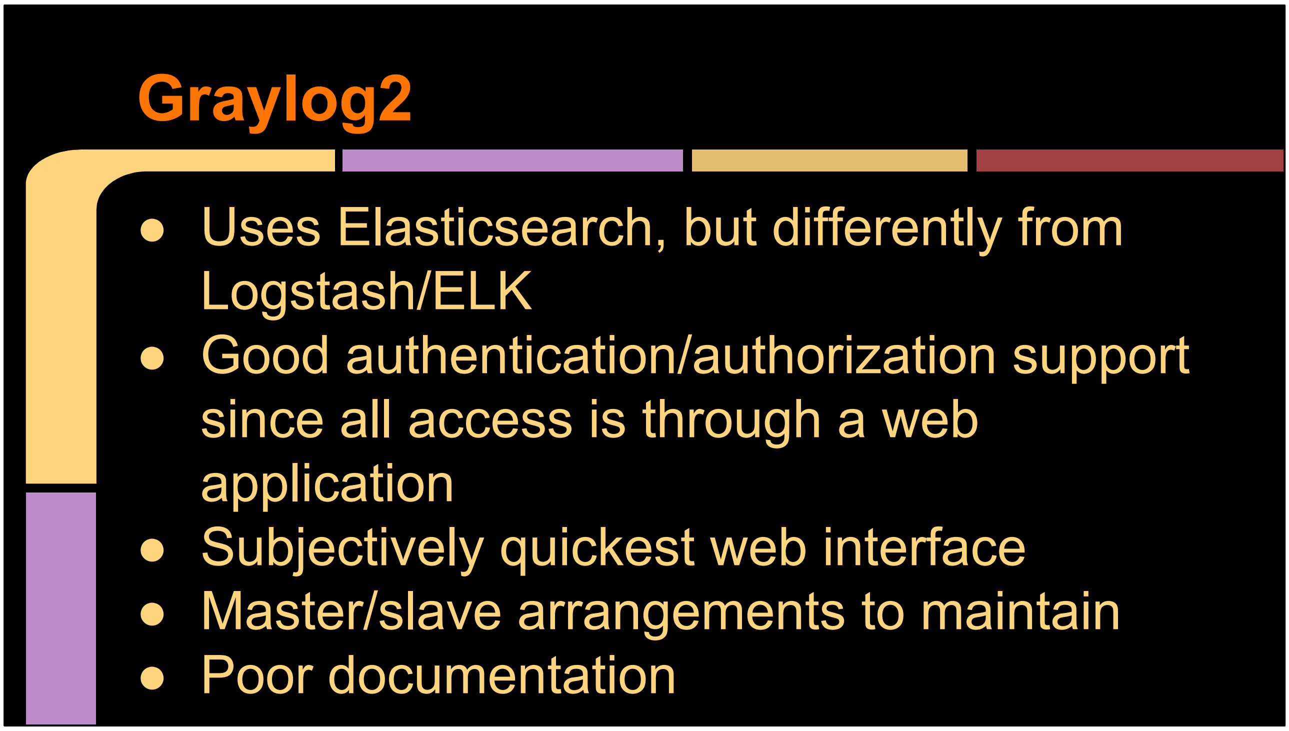
Time: 0:30

Quick explanation - All I/O and configuration with Elasticsearch can happen through its HTTP REST API. Kibana is just an HTML/CSS/JavaScript application that interacts with the Elasticsearch API via JSON requests and responses.

It's relatively easy to proxy connections through nginx or Apache and cover your whole installation behind some HTTP authentication, but you won't easily be able to filter certain data sets or elements on a per-user basis. There is an Elasticsearch plugin that is aiming to provide more of this, but it's not there yet.

Configuration of the indexing and parsing operations all happens in the configuration files for Logstash, so those parsing administration tasks are protected by filesystem access and config file changes.

Getting full use out of searching different types of data and taking advantage of the visualizations available will be easier if you understand how Elasticsearch treats fields, types, multiple values, different character sets, etc. The Elasticsearch Training is a huge help here.

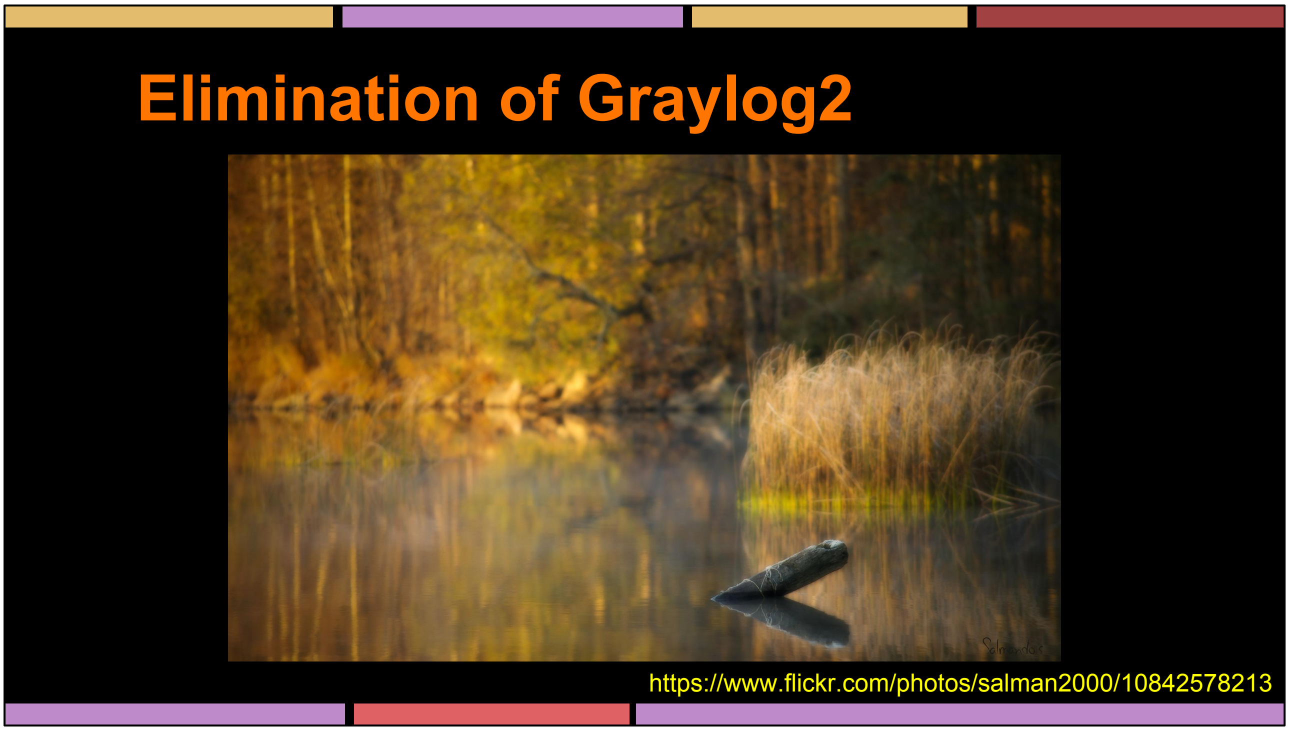


Graylog2

- Uses Elasticsearch, but differently from Logstash/ELK
- Good authentication/authorization support since all access is through a web application
- Subjectively quickest web interface
- Master/slave arrangements to maintain
- Poor documentation

Time: 0:30

- Graylog2 is also based on an Elasticsearch search indexing backend, but it's entirely hidden behind the Graylog2 applications.
- It's got authentication and authorization configuration to restrict access to particular reports to particular users or groups.
- It's got the quickest web interface, but definitely the least capable of them as well.
- It's cluster configuration is more like a master/slave arrangement. Since you're not sending logs directly into Elasticsearch, this is a big downgrade from the ELK approach, where you can send documents to any of the machines in the cluster to get them indexed. That's far more horizontally scalable than Graylog2.
- The documentation is poor.
  - Never figured out why some pages just showed errors.
  - Never figured out how to do regular expression searches
- Ultimately, the community support that makes ELK a contender for Splunk just isn't there for Graylog2.



Elimination of Graylog2

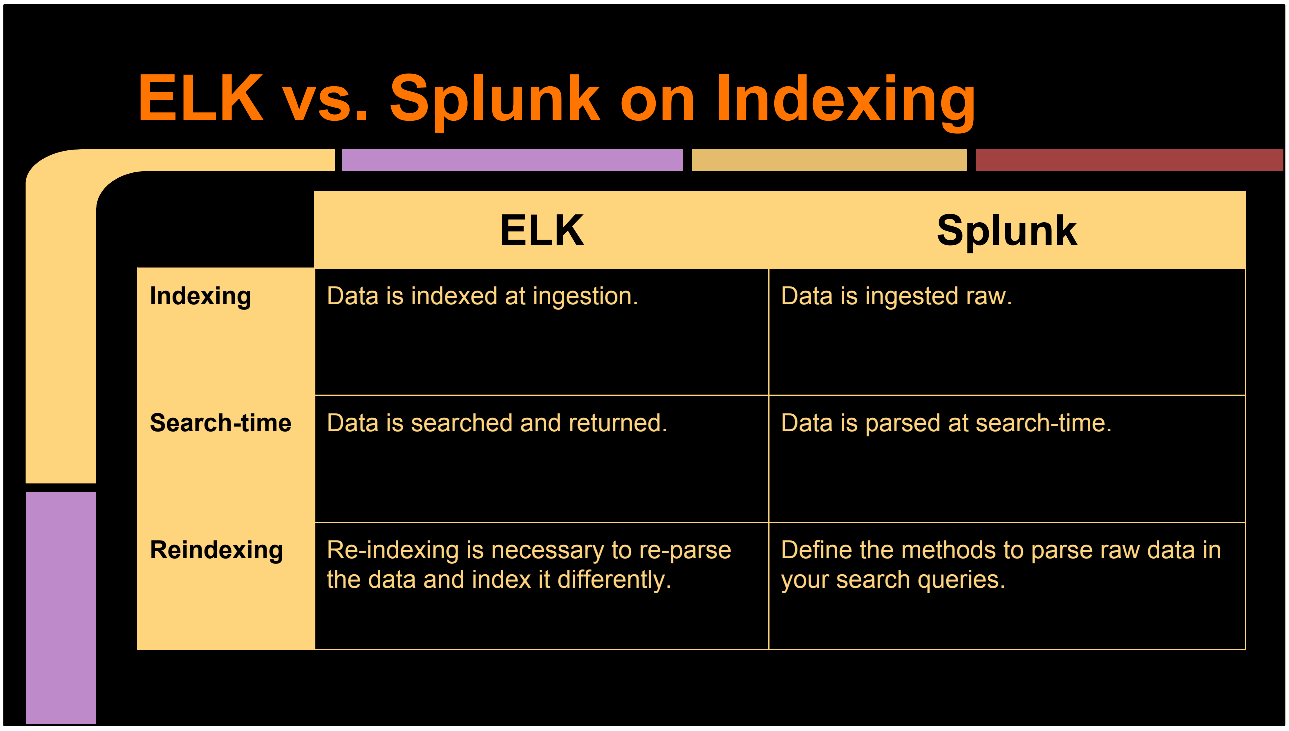
https://www.flickr.com/photos/salman2000/10842578213

Time: 1:00

It’s at this point of the comparison that Graylog2 has to be eliminated.

- We do have a lot of logs with geographic coordinates, but Graylog2 has no mapping visualization.
- Some of it’s plugins for configuration to connect it to PagerDuty or HipChat are available, but plugins aren’t as core a part of the structure of the application as with Logstash, so the selection of options is far more limited. Lots of output options would end up being custom-scripted call-out scripts that you have to write yourself and are just triggered by Graylog2.
- Graylog2 didn’t really have a good option for correlating multiple related events. Splunk could do this with sub-searches that search for some specific results in a set of data immediately following another result. Logstash could accomplish this with something like the elapsed plugin which will notice one event flow through the system and then look for the next related event to link to it in a single entry.
- Direct access to Elasticsearch with the ELK stack is a fundamental aspect of using it, whereas access to Elasticsearch with Graylog2 makes you miss out on a lot of the benefits of using the Graylog2 application. Since it’s not used heavily in the Graylog2 community, it ends up being a more poorly supported pathway in general.
- Graylog2 really doesn’t have an API-friendly way to get to the data you want.

- Ultimately, while you could find logs in the Graylog2 system to ascertain the conditions behind every use case we had, we couldn’t build a dashboard that would only show the relevant information with the necessary aggregate statistics or correlation between events that we were looking for.



ELK vs. Splunk on Indexing

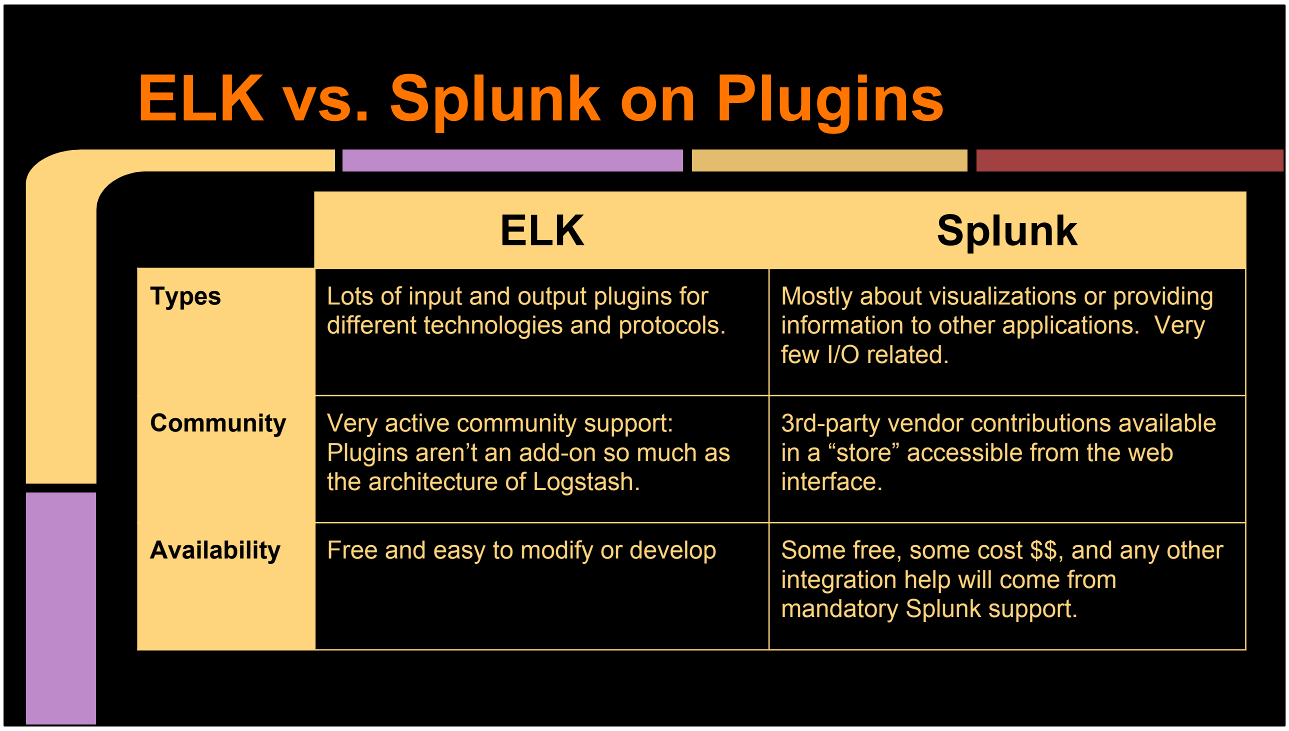
	ELK	Splunk
Indexing	Data is indexed at ingestion.	Data is ingested raw.
Search-time	Data is searched and returned.	Data is parsed at search-time.
Reindexing	Re-indexing is necessary to re-parse the data and index it differently.	Define the methods to parse raw data in your search queries.

Time: 1:10

It's easy to understand how data gets into Elasticsearch via Logstash. Logstash accepts messages, parses them with filters, conditional logic, and tags, and it spits out structured JSON documents for each event. Elasticsearch takes in those JSON documents and indexes the data for easy retrieval and statistics aggregation.

Splunk says they only ingest and store raw data, and that it's the reason they scale so well. They don't have to process anything on ingestion and its immediately available because searches are all executed against that raw data. You can redefine the formats of incoming data and assemble filter chains in the web interface that will re-parse data according to the patterns you're looking for. This is very powerful. You can also create reports that are pre-compiled and pre-indexed so they will be faster if you expect to request them often.

To accomplish this kind of flexibility in Elasticsearch, you would have to re-index the data you are processing, literally reading out the documents with something like Logstash, parsing/modifying/filtering them as you wish, and then indexing them back into the database. This is a very heavy process. In the ELK world, it's probably easier if you just imagine that once you find a pattern you want to keep being able to parse, you should make the necessary changes to your Logstash rules and indexing rules, and then just accept that the desired searches will come up empty against historical data.



ELK vs. Splunk on Plugins

	ELK	Splunk
Types	Lots of input and output plugins for different technologies and protocols.	Mostly about visualizations or providing information to other applications. Very few I/O related.
Community	Very active community support: Plugins aren't an add-on so much as the architecture of Logstash.	3rd-party vendor contributions available in a "store" accessible from the web interface.
Availability	Free and easy to modify or develop	Some free, some cost $$, and any other integration help will come from mandatory Splunk support.

Time: 1:00

Plugins are one of the biggest differentiators between these toolsets. They have very different approaches to plugins.

If you want to integrate your logging with literally everything else, Logstash is really your best bet. There's an awesome community full of people solving the same integration problems you probably have, and it's easy to make your own plugins. Logstash is really nothing except a framework of plugins for every function it serves.

If you want to collect event information from the Heroku API and a twitter feed or two, count stats to submit to graphite, translate certain types of messages into alerts in your internal chat system, send someone a page, and then store the data in S3, you're going to find the solution in Logstash.

If you want to pay someone else to integrate your logs into some package you've probably already paid a lot of money for, the Splunk store may have a solution for you. It's far more targeted at other vendor-supported integrations. The Splunk store will have a lot fo visualization tools for different types of business intelligence, custom dashboards that are ideal for particular vendors equipment logs, link usernames in logs to your Active Directory for more information about the user, etc, These are the types of solutions in the Splunk store.

Or you can put your Splunk support team to work for you. That support is a mandatory (and expensive) cost of at least your first year with Splunk to ensure you get your data to work for you, but they basically promise that they will make whatever you need happen.

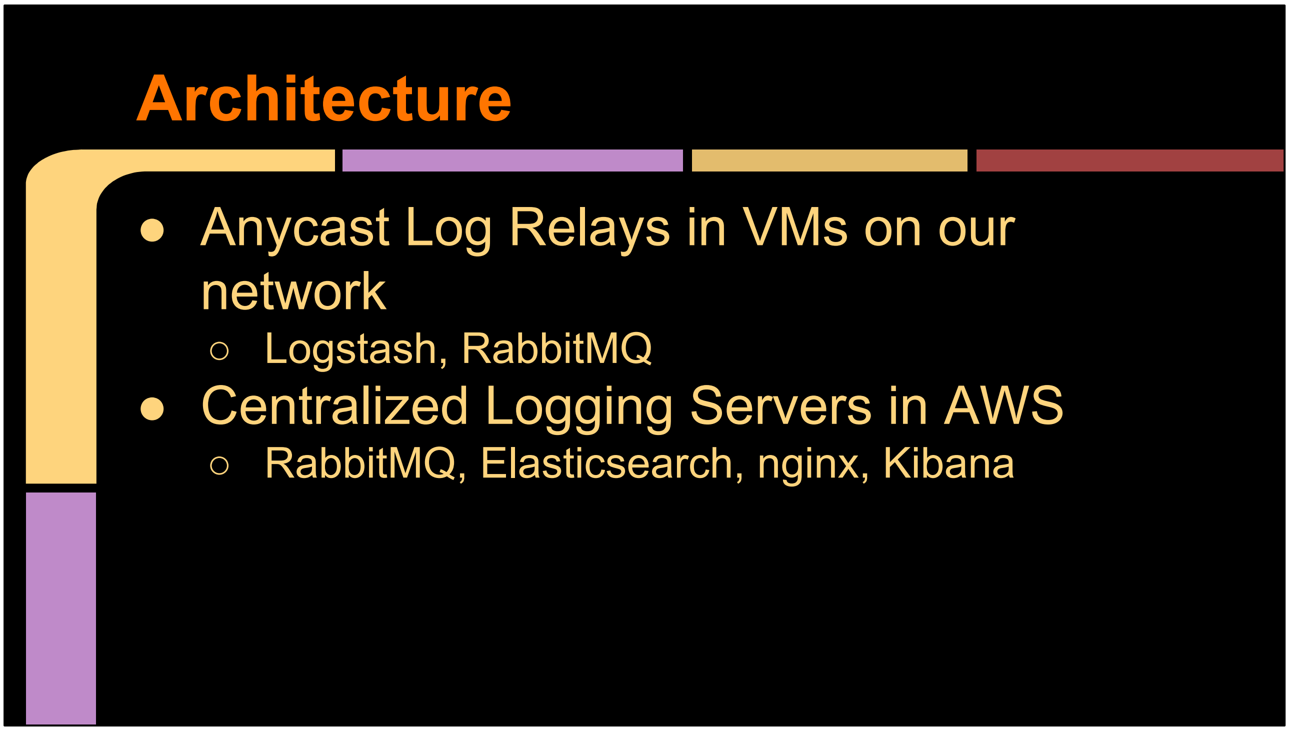
Questions before Implementation?

https://www.flickr.com/photos/paukrus/9826882836

Time: 0:10

Any questions so far before I get to the implementation of our architecture?

So we went with ELK. Cost was one motivator, but honestly the extra features that Splunk offered just weren't high on our list of requirements. They would be nice to have, but not nice enough to justify the cost. And the extra flexibility we got with Logstash has really opened up more possibilities too.



Architecture

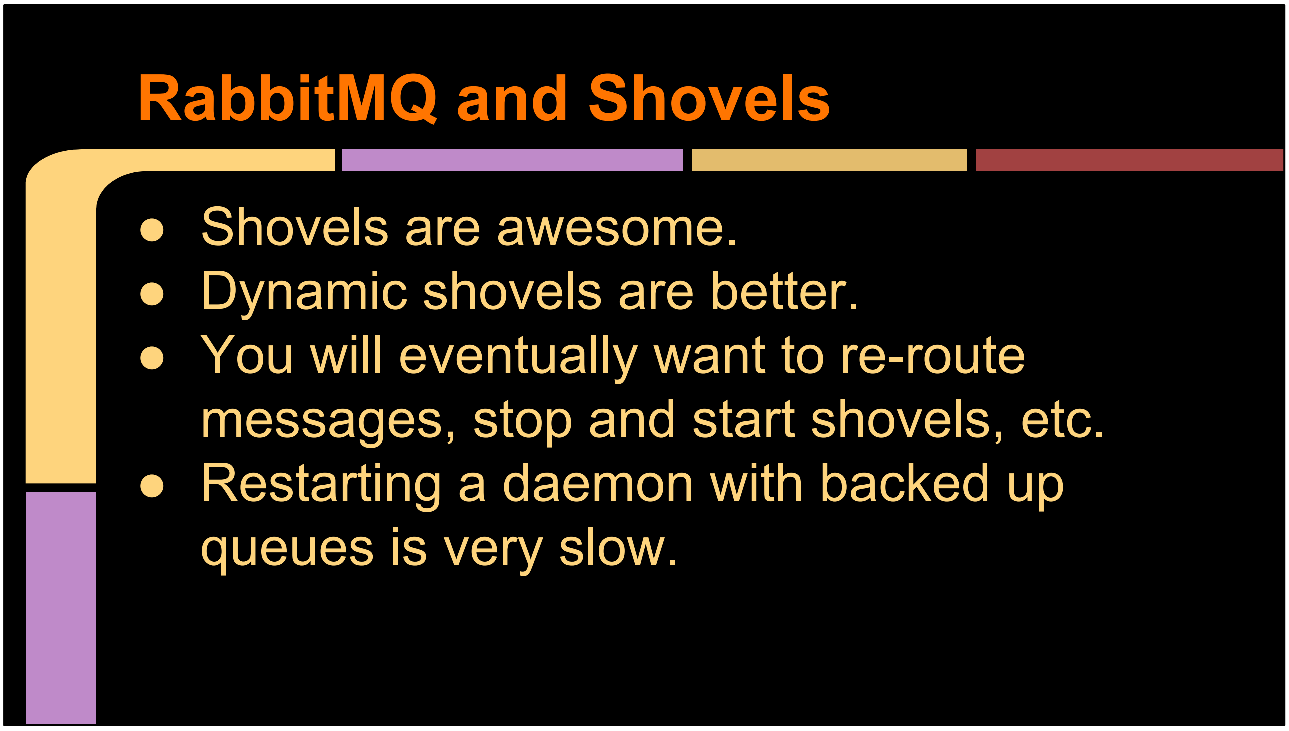
- Anycast Log Relays in VMs on our network
  - Logstash, RabbitMQ
- Centralized Logging Servers in AWS
  - RabbitMQ, Elasticsearch, nginx, Kibana

Time: 0:30

We've got two machine types in our system now.

The log relays are the logstash side of the system, accepting messages, parsing fields, etc. They deliver to local RabbitMQ exchanges, which use shovel threads to move messages from those queues into our core datacenters where we have routable access to our AWS instances.

The relays in our core datacenters have shovels that push the messages to the RabbitMQ processes on our centralized logging servers in AWS. The messages get from RabbitMQ to Elasticsearch by using an Elasticsearch River plugin - I'll talk more about that in a second, but it's worth mentioning that there are several Elasticsearch River plugins for bulk indexing data from 3rd party sources like RabbitMQ, Wikipedia, Twitter, Amazon SQS, CSV data, Dropbox, GitHub, JDBC, Redis, etc, etc, etc. In our case, we used the RabbitMQ River plugin to pull in messages formatted already for bulk indexing.. Each Centralized Logging Server runs two separate Elasticsearch data and master processes, called nodes, as well as nginx to serve the Kibana web interface and proxy requests to the Elasticsearch HTTP API.



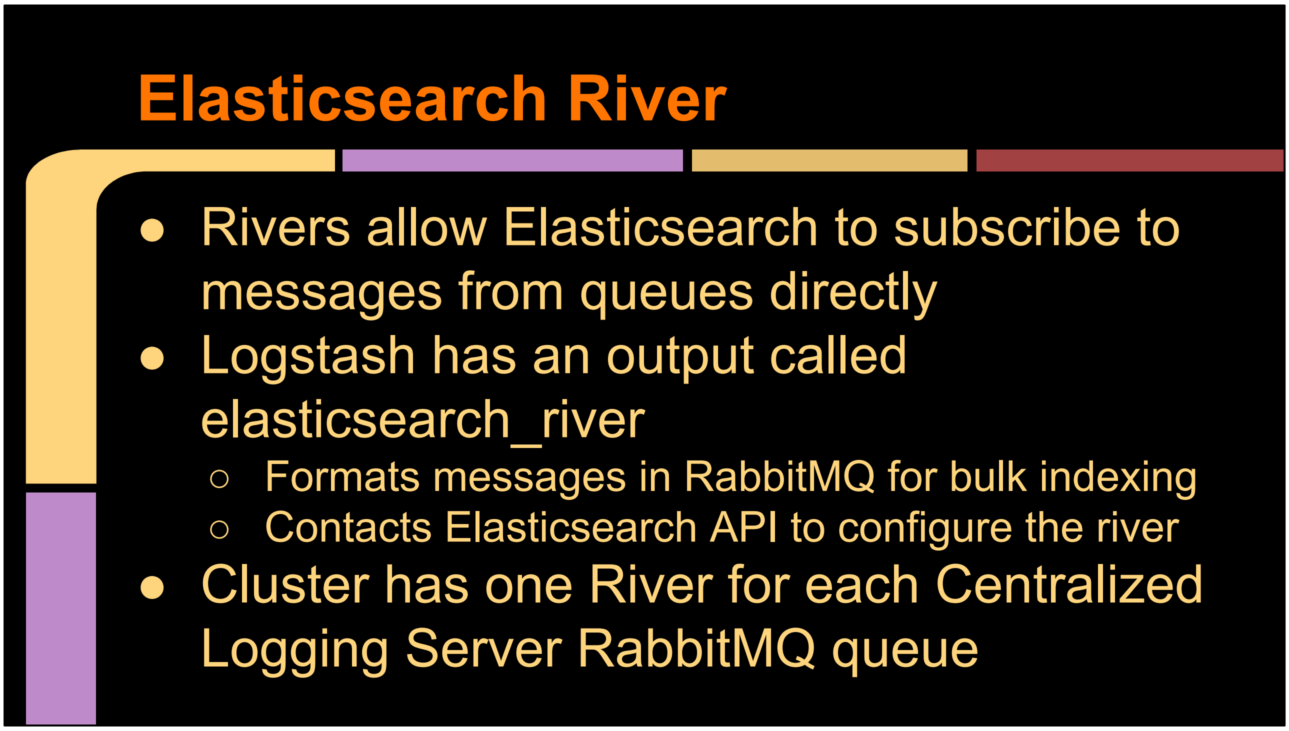
RabbitMQ and Shovels

- Shovels are awesome.
- Dynamic shovels are better.
- You will eventually want to re-route messages, stop and start shovels, etc.
- Restarting a daemon with backed up queues is very slow.

Time: 0:20

As a reminder, shovels are those threads embedded in the RabbitMQ Erlang virtual machine that act as an AMQP client reading messages from one queue and delivering them to another exchange. They can be configured in the daemon config files, but then you need to restart RabbitMQ in order to make adjustments to them. If you configured them with API calls, they can be created and destroyed and reconfigured on the fly.

Since restarting a RabbitMQ daemon with a bunch of messages in queue can take awhile, this is almost always more flexible.

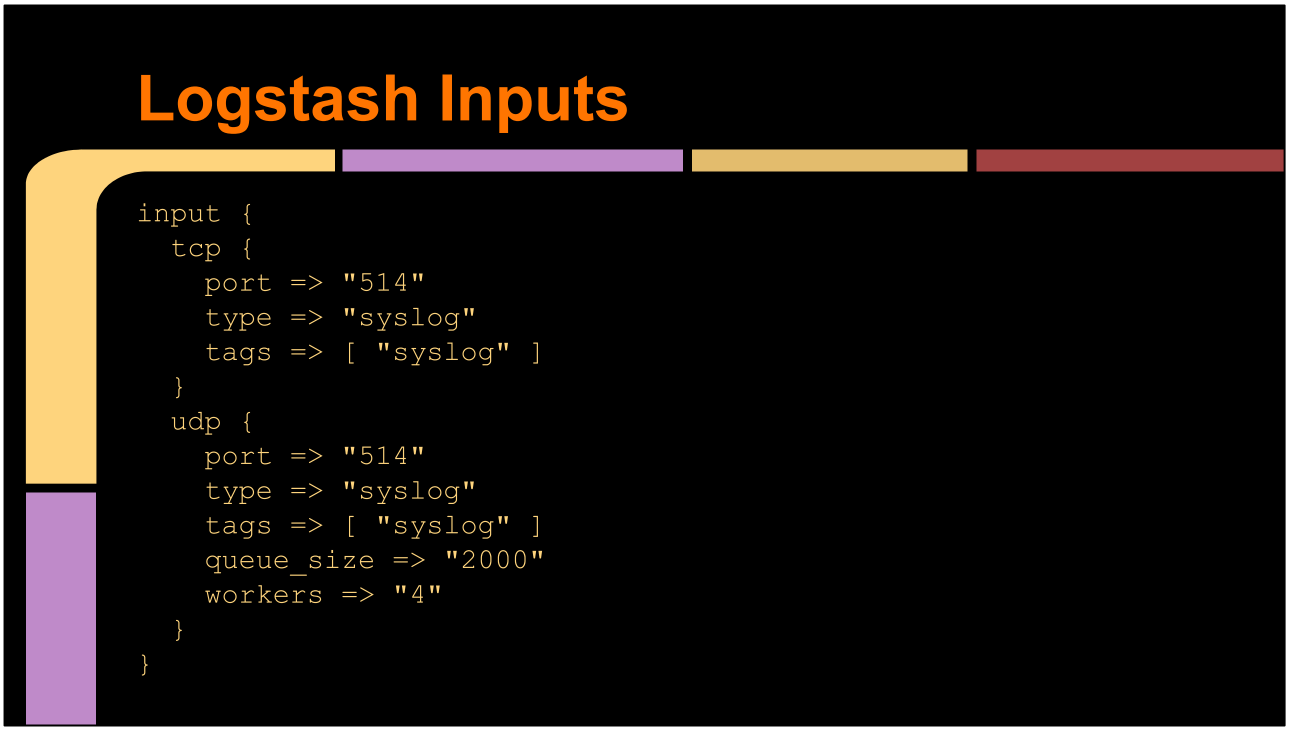


Elasticsearch River

- Rivers allow Elasticsearch to subscribe to messages from queues directly
- Logstash has an output called elasticsearch_river
  - Formats messages in RabbitMQ for bulk indexing
  - Contacts Elasticsearch API to configure the river
- Cluster has one River for each Centralized Logging Server RabbitMQ queue

Time: 1:10

- We're making use of Elasticsearch Rivers for ingesting messages into the Elasticsearch cluster. This is similar to a shovel in Elasticsearch terms.
- A separate thread in the Elasticsearch JVM will be dedicated to the river and will ingest messages formatted for bulk indexing.
- In particular, the RabbitMQ River Elasticsearch plugin creates a thread for an AMQP client to read in messages from the RabbitMQ queues.
- Logstash has an elasticsearch_river output plugin that will send messages formatted for the Elasticsearch Bulk API into RabbitMQ, and it will also setup the River thread in the corresponding Elasticsearch cluster.
  - I'm actually using a modified version of the elasticsearch_river plugin.
  - Mine is different because the Logstash processes around our system cannot all reach the Elasticsearch cluster, and I don't want them each setting up their own copy of the same rivers anyway.
  - I use a Python script talking to the Elasticsearch API to manage the rivers that should be configured at any time instead.
  - It was really easy to take the code that is used in the elasticsearch_river plugin, make a few minor changes, and deploy my "new" plugin alongside all the built-in Logstash plugins.

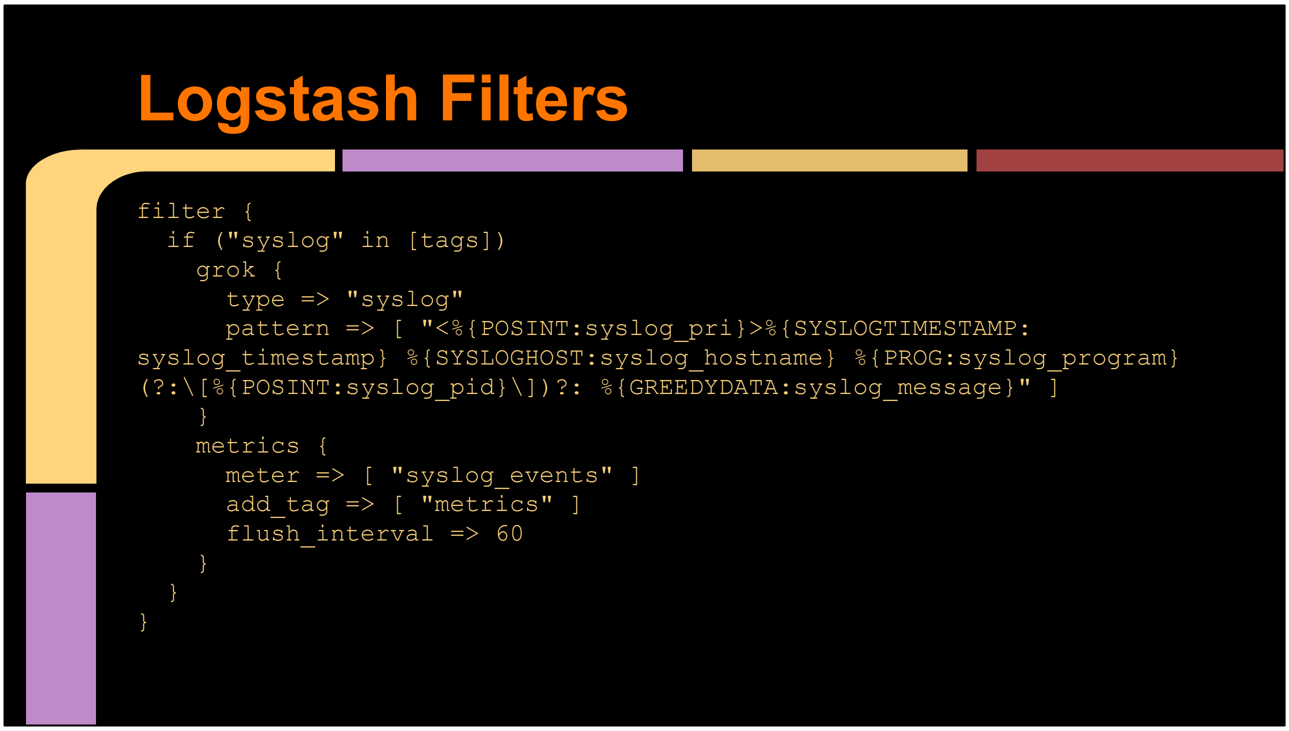


Logstash Inputs

```
input {
  tcp {
    port => "514"
    type => "syslog"
    tags => [ "syslog" ]
  }
  udp {
    port => "514"
    type => "syslog"
    tags => [ "syslog" ]
    queue_size => "2000"
    workers => "4"
  }
}
```

Time: 0:10

Logstash config files are pretty easy to understand. Here’s a simple input section that sets up listeners for syslog.



Logstash Filters

```
filter {
  if ("syslog" in [tags])
    grok {
      type => "syslog"
      pattern => [ "<%{POSINT:syslog_pri}>%{SYSLOGTIMESTAMP:
syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %{PROG:syslog_program}
(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" ]
    }
    metrics {
      meter => [ "syslog_events" ]
      add_tag => [ "metrics" ]
      flush_interval => 60
    }
  }
}
```

Time: 0:40

Here’s a filter section that can be used to do all sorts of things in response to types of messages. You can use boolean and conditional logic to skip around in here. This config is looking for all the messages that have been given the tag syslog, using grok parse out the typical syslog fields and metrics count them in our syslog_events counter so the counter can be sent to Graphite. Here’s some examples of other stuff you can do with filters...

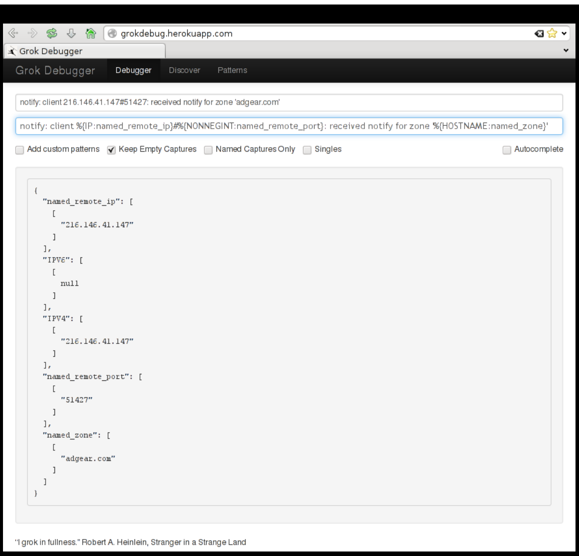
- Anonymize fields in log lines with hashes
- Encryption of fields
- GeoIP lookup of IP addresses to add lat/long coordinates
- Check IPs against CIDR network notations
- Drop, clone, or modify messages or fields
- Throttle similar or identical events
- Combine multiple lines like a stacktrace into a single event
- Process syslog PRI header
- Translations
- There are tons more...

Logstash Filters

```
if ( [syslog_program] == "named" ) {
  grok {
    match => [ "message", "^(?<named_category>[-a-z]+):" ]
    tag_on_failure => []
  }
  grok {
    match => [ "message", "notify: client %{IP:named_remote_ip}#%
{NONNEGINT:named_remote_port:int}: received notify for zone '%
{HOSTNAME:named_zone}'" ]
    add_tag => "named_received_notify"
    tag_on_failure => []
  }
}
```

Time: 0:10

Grok is kind of an ugly thing to parse, but it’s really just regular expressions under the covers.



```
notify: client 216.146.41.147#51427: received notify for zone 'adgear.com'
```

```
notify: client %{IP:named_remote_ip}#%{NONNEGINT:named_remote_port}: received notify for zone %{HOSTNAME:named_zone}'
```

Add custom patterns [✓ Keep Empty Captures] Named Captures Only Singles Autocomplete

```
{
  "named_remote_ip": [
    [
      "216.146.41.147"
    ]
  ],
  "IPV6": [
    [
      null
    ]
  ],
  "IPV4": [
    [
      "216.146.41.147"
    ]
  ],
  "named_remote_port": [
    [
      "51427"
    ]
  ],
  "named_zone": [
    [
      "adgear.com"
    ]
  ]
}
```

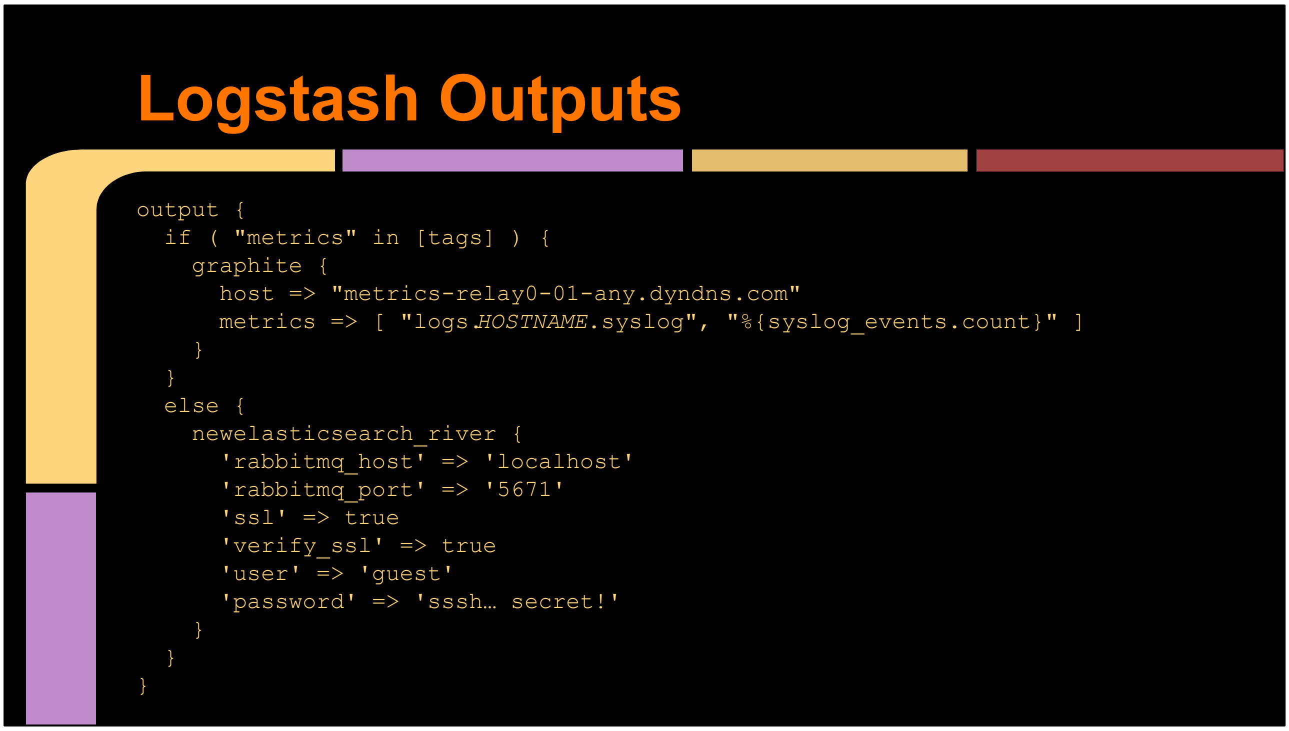
"I grok in fullness." Robert A. Heinlein, Stranger in a Strange Land

grok debugger = awesome

Time: 0:10

If you're like me, you'll really want to play with grok a bunch. This Heroku app called the Grok Debugger is awesome.

grokdebug.herokuapp.com lets you key in a pattern and a log message and you'll see easily how well it parses.



Logstash Outputs

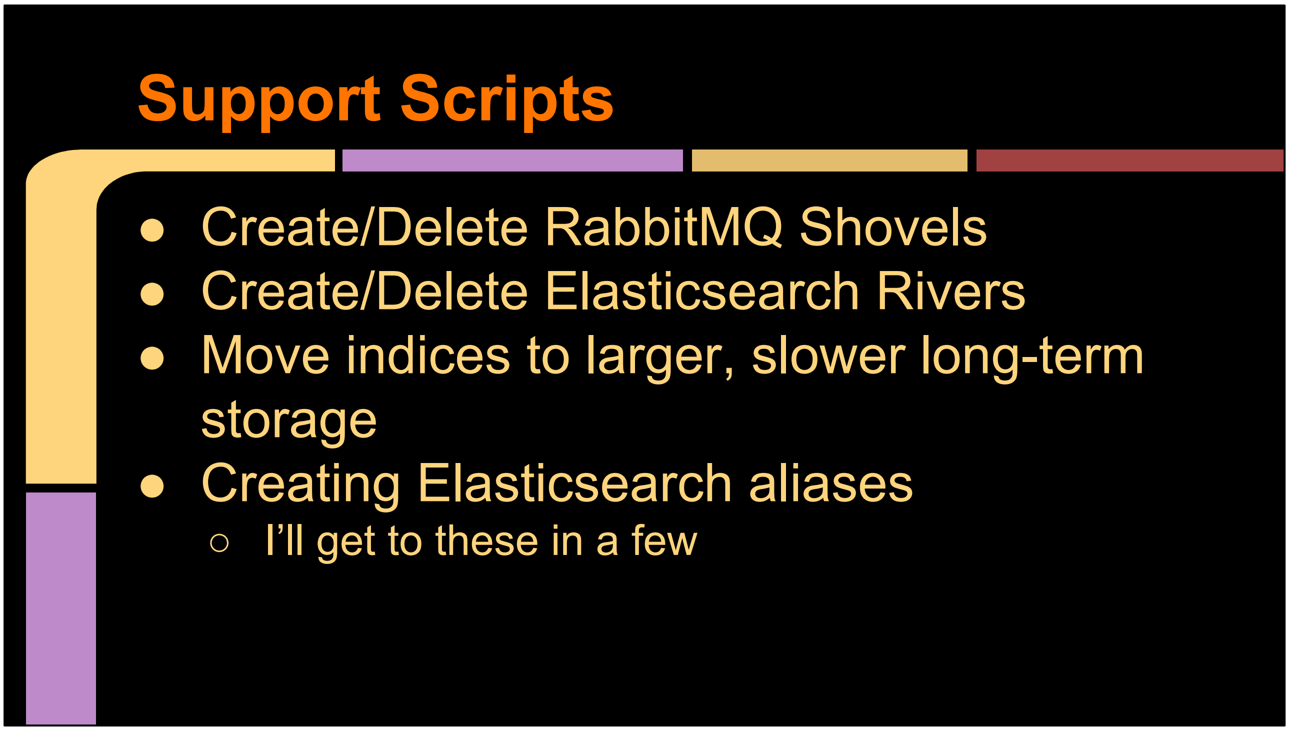
```
output {
  if ( "metrics" in [tags] ) {
    graphite {
      host => "metrics-relay0-01-any.dyndns.com"
      metrics => [ "logs.HOSTNAME.syslog", "%{syslog_events.count}" ]
    }
  }
  else {
    newelasticsearch_river {
      'rabbitmq_host' => 'localhost'
      'rabbitmq_port' => '5671'
      'ssl' => true
      'verify_ssl' => true
      'user' => 'guest'
      'password' => 'sssh… secret!'
    }
  }
}
```

Time: 0:25

And here's an example outputs section from Logstash, again making use of simple boolean and conditional logic.

Our metrics filter created a new event every 60 seconds with the metrics tag. It's a counter for the number of matching messages. We're sending those events off through the Graphite output plugin.

All the other messages are going off to our Elasticsearch cluster.



Support Scripts

- Create/Delete RabbitMQ Shovels
- Create/Delete Elasticsearch Rivers
- Move indices to larger, slower long-term storage
- Creating Elasticsearch aliases
  - I'll get to these in a few

Time: 0:30

I did create a few really helpful scripts to interact with the RabbitMQ and Elasticsearch APIs for our system.

This system is highly dependent on being able to freely move messages around among the RabbitMQ queues. It's important to have both these first two scripts available for automation and interaction.

Logstash indexes data into Elasticsearch with indices named for the date, so it's easy to see when no new information is coming into an index. If you want to move it, you can close the index and do whatever you like with the files on the filesystem, symlink them from somewhere else, etc.

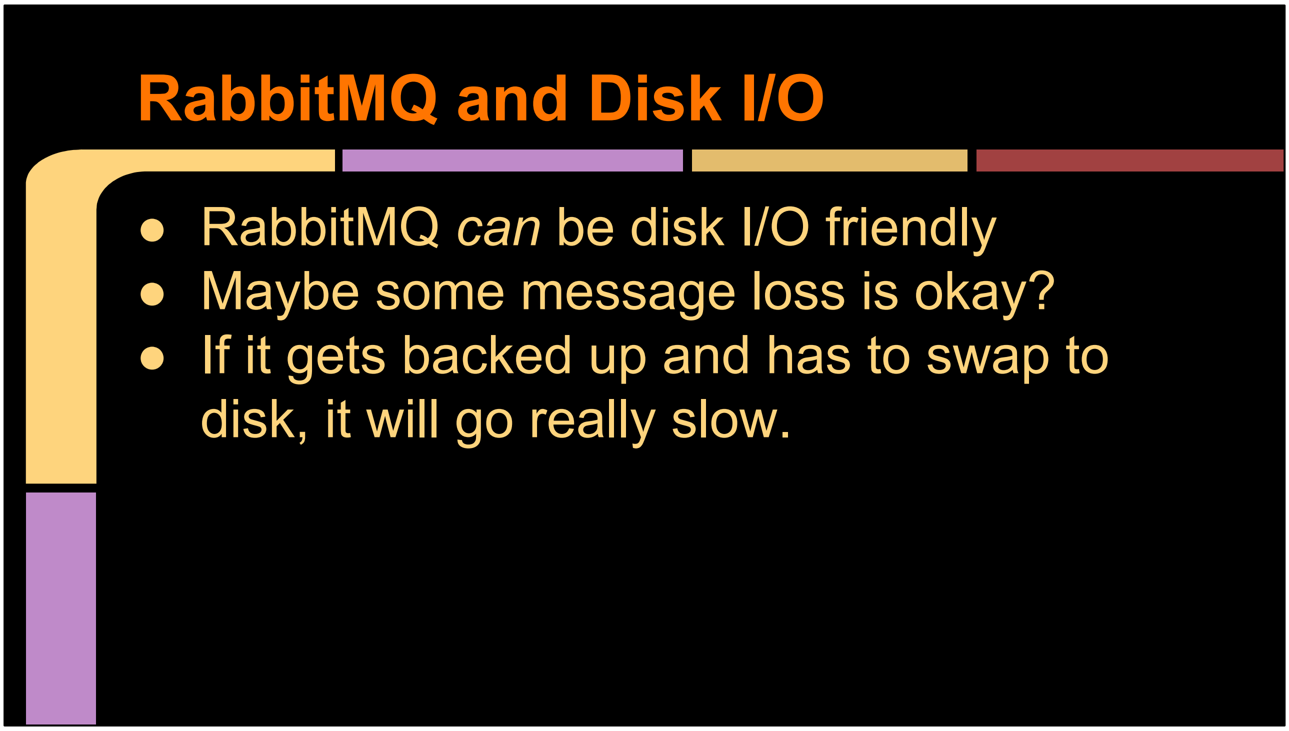
And finally, there's a script to manage the Elasticsearch aliases, which I'll get to in a few minutes.

Pitfalls

https://www.flickr.com/photos/eurleif/186807031

Time: 0:10

If you follow everything I've told you, you'll probably have an issue somewhere, but I won't pretend to know where your particular pitfalls will be. I'll tell you a bit about some of ours.



RabbitMQ and Disk I/O

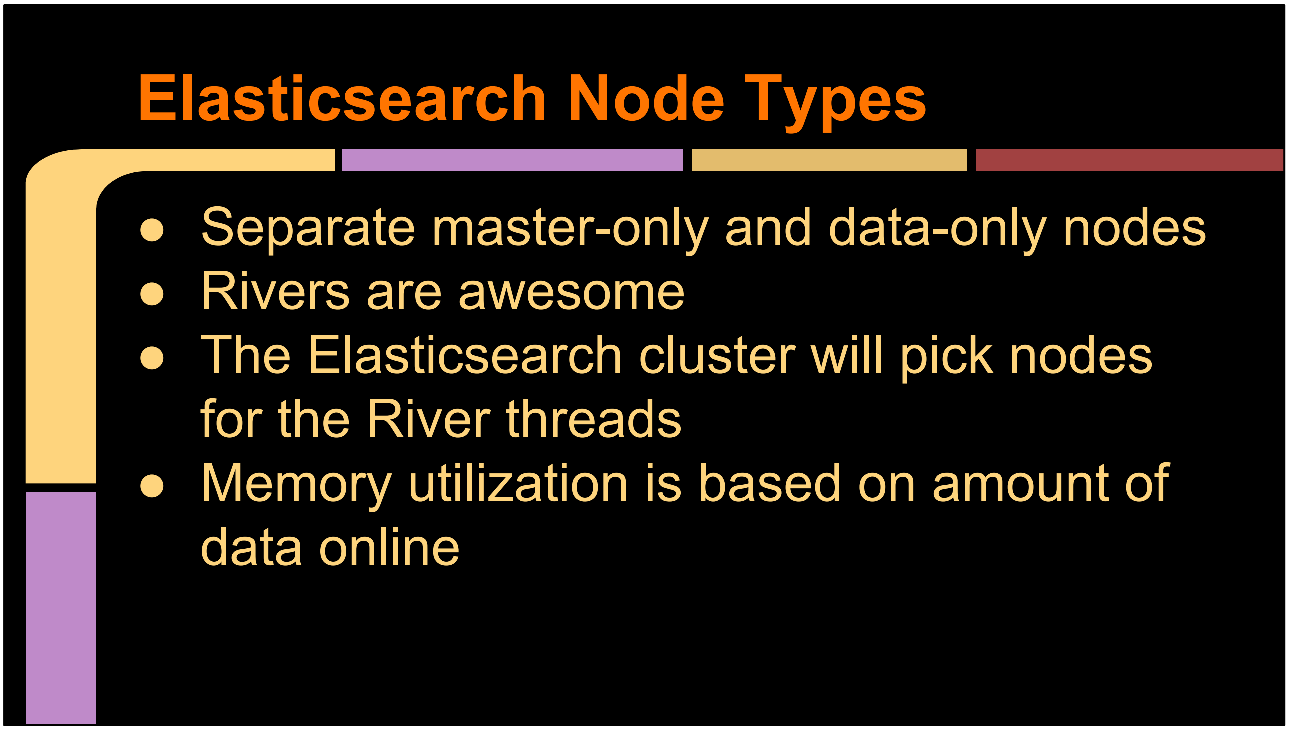
- RabbitMQ *can* be disk I/O friendly
- Maybe some message loss is okay?
- If it gets backed up and has to swap to disk, it will go really slow.

Time: 0:25

If you're okay with messages only existing in memory, you can opt to just lose messages if the daemon stops/starts or the queues fill up. A persistent message will be pushed to disk if it isn't picked up right away. So you don't want your queues full of persistent messages to get backed up.

If a queue full of persistent messages starts swapping to disk, it will slow down a lot and you'll want to adjust the shovels to route around it to keep the system moving smoothly. A message queue that has no new messages entering it will drain much faster than a queue that has a constant stream of new messages coming in because of the FIFO nature of the queues.

If you want persistence for messages, then that advice before about dynamic shovels before will be very helpful. Restarting a daemon with a full queue will be slow. If you manually re-route things under high-load conditions, you'll be happy you have easy control over dynamic shovels.



Elasticsearch Node Types

- Separate master-only and data-only nodes
- Rivers are awesome
- The Elasticsearch cluster will pick nodes for the River threads
- Memory utilization is based on amount of data online

Time: 1:10

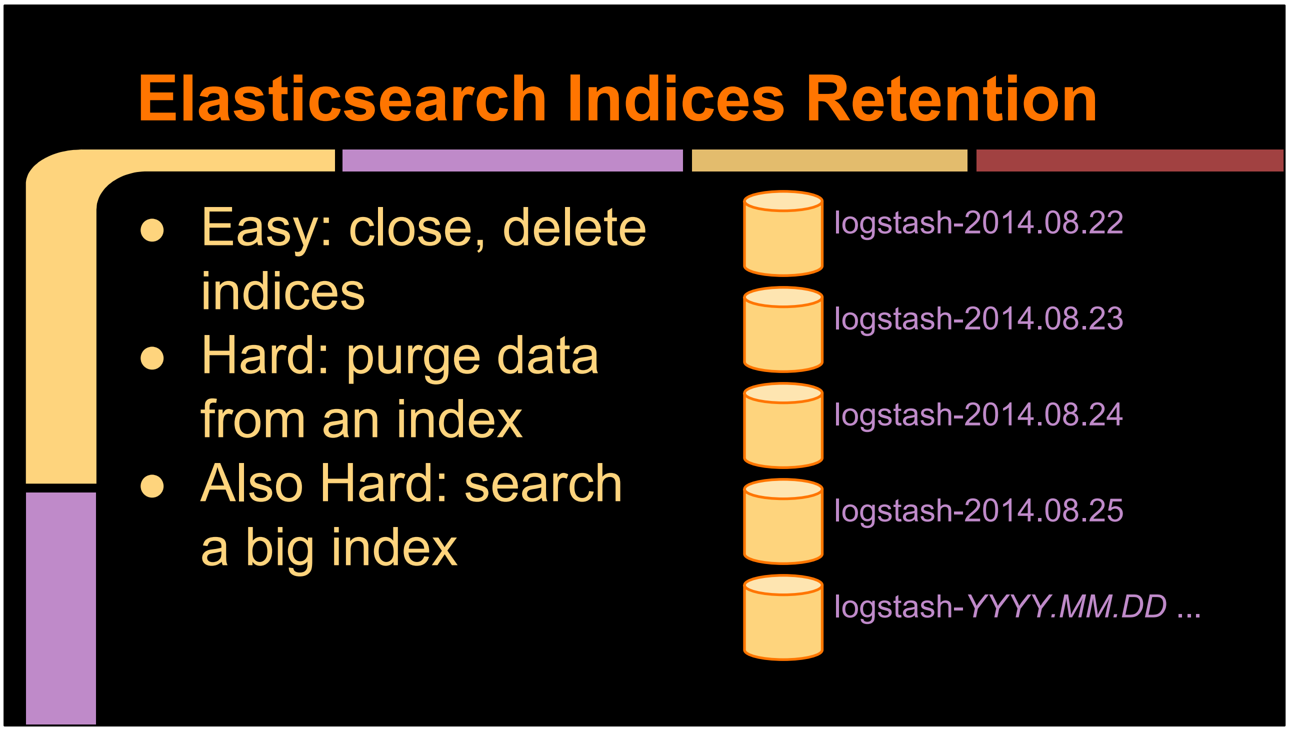
Each Elasticsearch process is called a node in the cluster. Out of the box, one node will be elected as the master, responsible for allocating shards of indices to different nodes in the cluster. The master isn't a lot of work, but it's important work.

There are two types of garbage collection in Java, old generation and young generation. When the process is really running tight on memory, the old generation stop-the-world garbage collection will trigger and the process will go silent to everything else until it's done. If the master enters old generation garbage collection for too long, then the rest of the cluster may decide to elect a new master since it appears to be gone. When it comes back, you have two masters, a split brain, and some data loss.

This is a real problem for Elasticsearch no matter what, but it's actually not too hard to avoid in practice. Each machine/server in our setup runs two Elasticsearch processes. One is configured to be master-eligible, but not data-eligible. The other is configured to be data-eligible, but not master-eligible. The master-eligible nodes have minimal memory, less than half a gig. The data-eligible nodes will have about half the system's memory dedicated to their stack. The rest can go to the OS for disk cache and other needs. Once the masters are responsible for sitting idly waiting for an election or doing the relatively minimal work of being a master, they won't use enough memory resources to ever require old generation garbage collection. The data nodes may, but they can also disappear for a bit without repercussions to the cluster.

Once you have dedicated master and data nodes, make sure your master nodes aren’t also eligible for getting rivers assigned to them. The cluster will select nodes to run the River threads, and unless you tell it otherwise, it may select one of your master-only nodes. You don’t want the masters doing any more work than they have to.

After that, you can still find your cluster struggling with availability or stability if it doesn’t have enough memory space around for working with result sets and caches. Closing old indices will be necessary. Every open shard/index will take up more memory. It’s trivial to bring older indexes online if they are on disk, and only takes a few seconds. You should just make sure to have enough memory and enough nodes in your cluster to have all the indices you want open. Older indices should be opened just when you explicitly want to search them. More memory is always better.



Elasticsearch Indices Retention

- Easy: close, delete indices
- Hard: purge data from an index
- Also Hard: search a big index

logstash-2014.08.22
logstash-2014.08.23
logstash-2014.08.24
logstash-2014.08.25
logstash-*YYYY.MM.DD* ...

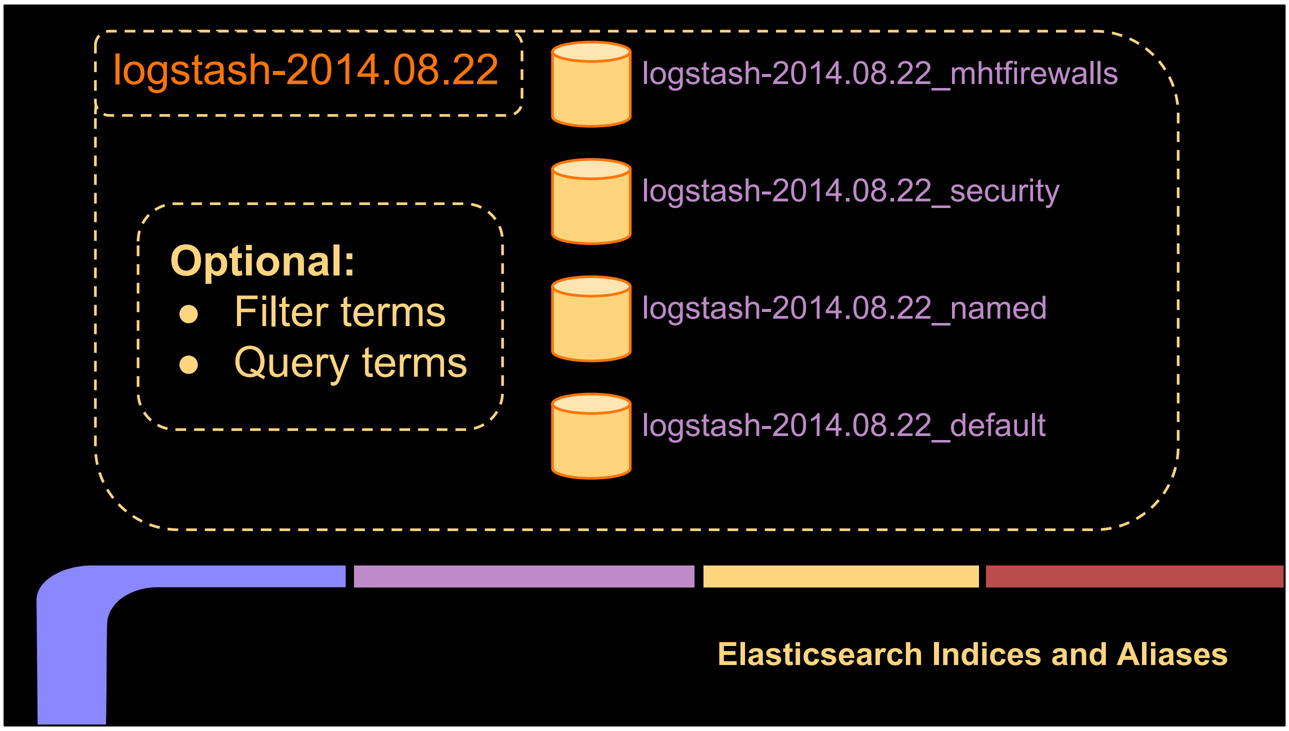
Time: 0:35

When using Logstash to put data into Elasticsearch, it gets arranged into indices named like this by default.

This makes age-based management of indices really easy. Opening, closing, and deleting indices are trivial operations requiring just a few seconds. Moving an index is just the work of the disk I/O while an index is closed.

It's really important though to recognize that you cannot just delete *some* of the data from these indices. They are all-or-nothing operations. Trimming or purging some information from an index effectively requires rebuilding the index from empty.

Also, as these indices get bigger, their performance will suffer. We started noticing that for our cluster when indices were about 100GB in size, maybe 200 million records, but every cluster will use a different number of different types of machines, different sharding strategies between nodes, different document sizes, etc. It's hard to extrapolate from one setup to another where you'll run into limitations.


logstash-2014.08.22
Optional:
● Filter terms
● Query terms
logstash-2014.08.22_mhtfirewalls
logstash-2014.08.22_security
logstash-2014.08.22_named
logstash-2014.08.22_default
Elasticsearch Indices and Aliases

Time: 0:40

We specifically wanted to maintain different retention rules for different types of logs. We want to be able to easily save firewall and security-related logs for a longer amount of time without storing everything from those days. Also, a lot of our saved dashboards only needed to access one clear subset of the logs we were storing.

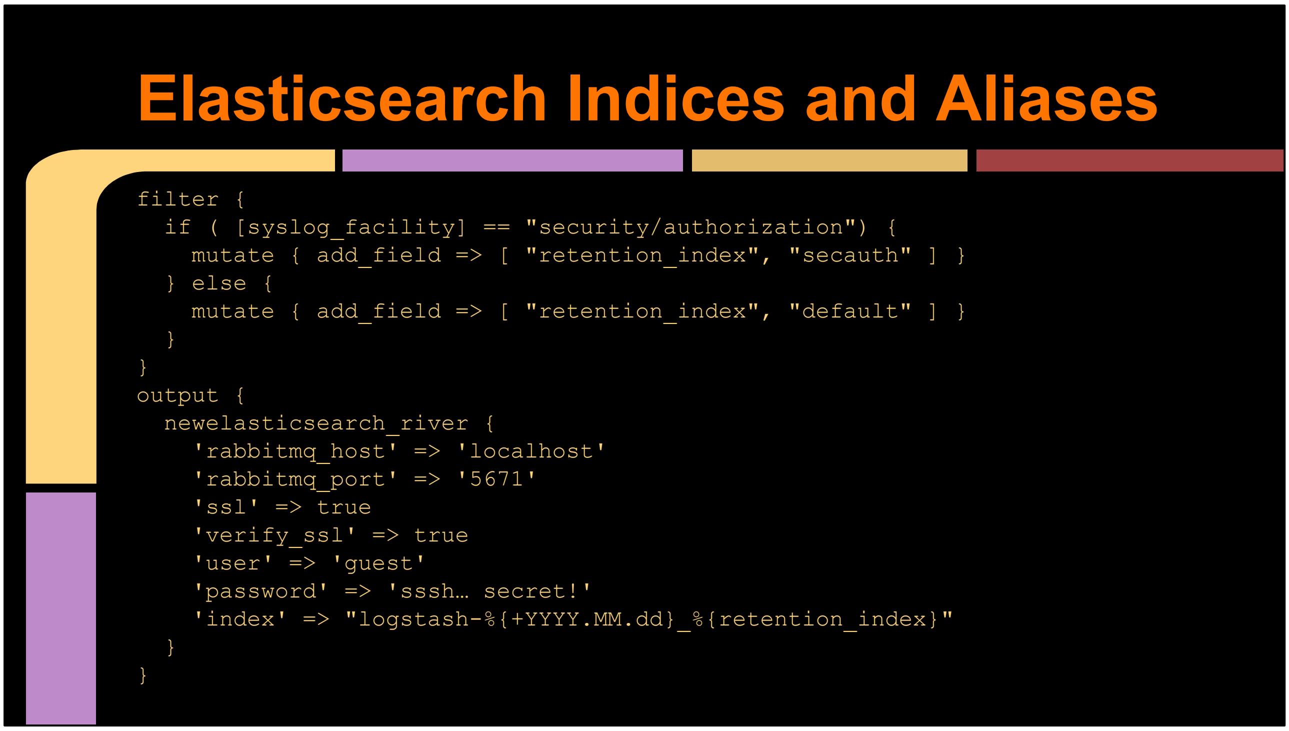
Elasticsearch has a function called aliases that can help here. Aliases are very similar in concept to views in a SQL database. You can specify source indices and query/filter parameters as part of your index alias.

We configured Logstash to store data in indices named like these on the right, and we create the typical Logstash-DATE index names as aliases. The Kibana web interface will default to searching the alias, and it will work as expected. As far as search queries are concerned , searching an alias and searching an index are identical requests.

But this setup gains us two really helpful advantages:

- We can delete the default index that we aren't that attached to, while keeping the security logs for longer-term retention. We can keep the named logs for as long as the DNS team tells us they want to track them.
- If building a dashboard that focuses on only BIND logs, we can speed it up a good bit by letting it focus on only the *_named* indices. It doesn't have to

- search the other indices that were only going to return 0 results anyway.



Elasticsearch Indices and Aliases

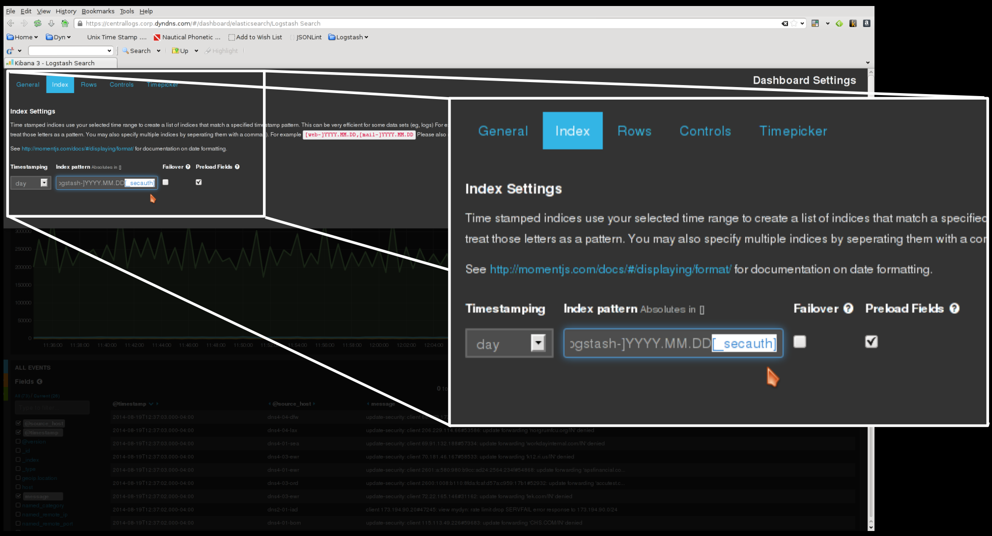
```
filter {
  if ( [syslog_facility] == "security/authorization") {
    mutate { add_field => [ "retention_index", "secauth" ] }
  } else {
    mutate { add_field => [ "retention_index", "default" ] }
  }
}
output {
  newelasticsearch_river {
    'rabbitmq_host' => 'localhost'
    'rabbitmq_port' => '5671'
    'ssl' => true
    'verify_ssl' => true
    'user' => 'guest'
    'password' => 'sssh… secret!'
    'index' => "logstash-%{+YYYY.MM.dd}_%{retention_index}"
  }
}
```

Time: 0:15

Here's a snippet of how I configured this sharding index/alias setup with Logstash's filters and outputs.

You can see in the filter section that I added a field to events according to matching certain characteristics.

Then, in the output section, I used that new field as part of the destination index name.



Dashboard Settings

General | Index | Rows | Controls | Timepicker

Index Settings

Time stamped indices use your selected time range to create a list of indices that match a specified
treat those letters as a pattern. You may also specify multiple indices by seperating them with a co

See http://momentjs.com/docs/#/displaying/format/ for documentation on date formatting.

Timestamping: day
Index pattern Absolutes in []: ›gstash-]YYYY.MM.DD[_secauth]
Failover: ☐
Preload Fields: ☑

Elasticsearch Indices and Aliases

Time: 0:10

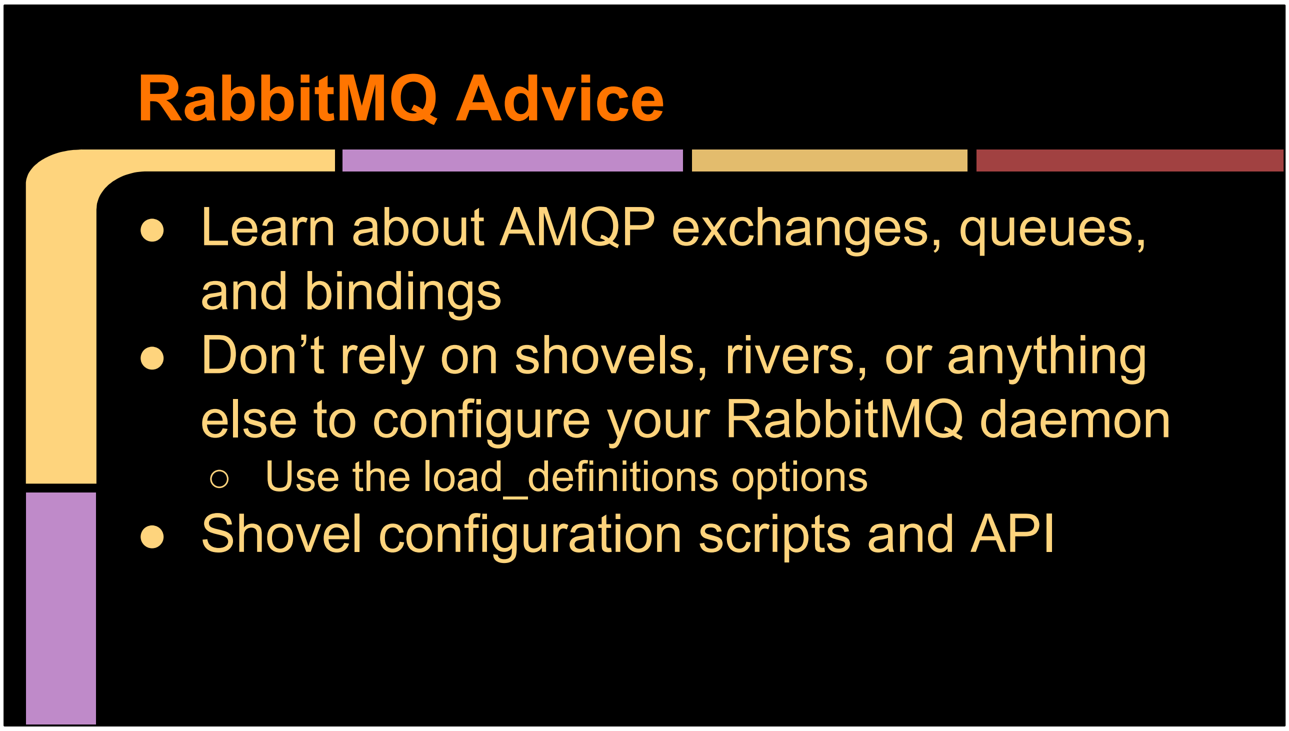
Here’s where we can configure the Kibana dashboard interface to look in only specific indices. There are little gear icons in Kibana for settings. The one in the upper right corner lets you specify dashboard settings like what indices to query.

Advice

https://www.flickr.com/photos/patgaines/4558323774

Time: 0:10

So here’s my best attempt to give some takeaways in the form of advice for the various tools employed in our architecture.



RabbitMQ Advice

- Learn about AMQP exchanges, queues, and bindings
- Don’t rely on shovels, rivers, or anything else to configure your RabbitMQ daemon
  - Use the load_definitions options
- Shovel configuration scripts and API

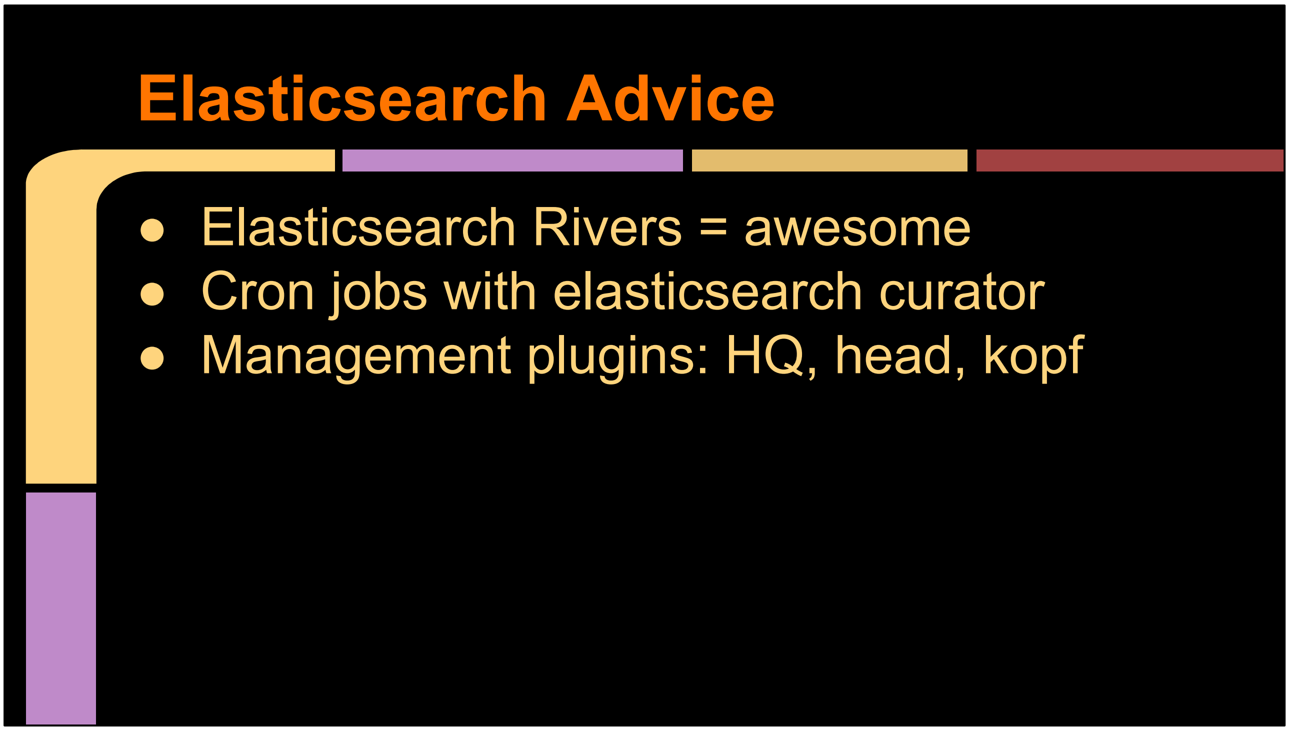
Time: 0:45

It’s worth learning about the types of exchanges, queues, and bindings in AMQP once you are trying to get a bit more flexibility.

Nearly every AMQP client (including Elasticsearch Rivers, RabbitMQ shovels, and the Logstash RabbitMQ output plugin) will have options to automatically set up these elements for you. Even RabbitMQ as a server will automatically bind certain queues to your default exchange. These are helpful to getting started quickly, but I eventually found too much confusion trying to make them play nicely. For example, creating a queue will fail if another queue already exists with the same name but slightly different settings. This probably won’t be an issue if you just setup a RabbitMQ daemon and leave it alone, but automation is a lot easier with explicit controls.

I found it best to have RabbitMQ configured to use the load_definitions option, which allows setting up the users, passwords (hashed), exchanges, queues, and bindings at startup in the configuration files.

I also found it best to set up shovels via API calls, rather than with static configuration.



Elasticsearch Advice

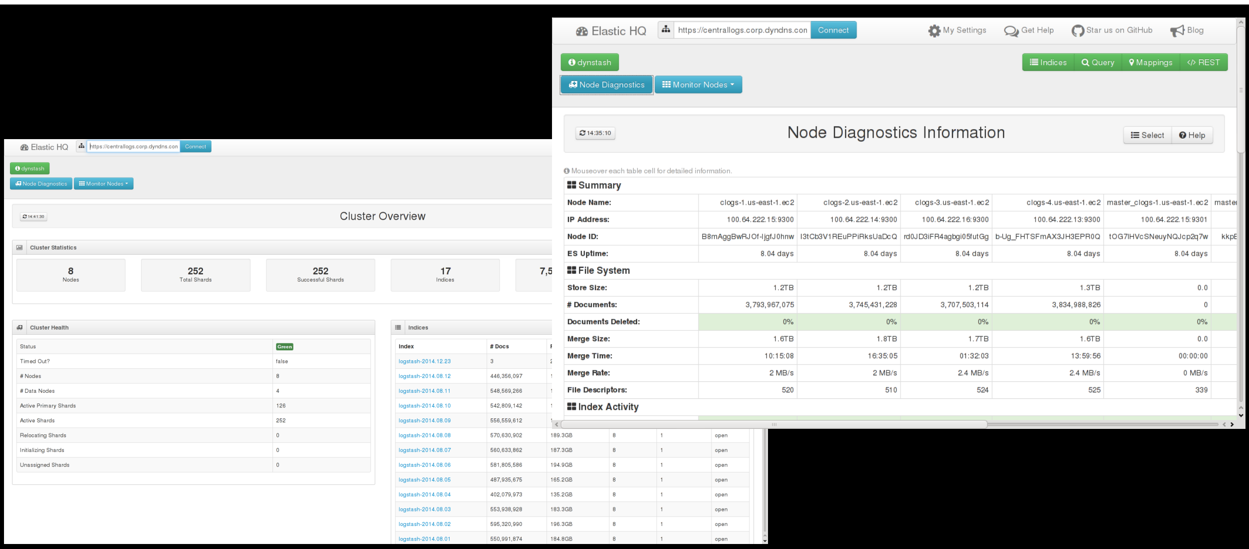
- Elasticsearch Rivers = awesome
- Cron jobs with elasticsearch curator
- Management plugins: HQ, head, kopf

Time: 0:30

The Elasticsearch River plugins are a great approach to ingesting large amounts of data fast. Setting them up with API scripts is really easy and worthwhile.

The Elasticsearch curator is a script that Elasticsearch has created to automate index management. Use it as much as possible in your long-term index management, performance management, etc.

There are lots of management plugins for visually interacting with an Elasticsearch cluster. HQ, head, and kopf are my favorites. I'll show screen shots of each in a second. They are all pretty easy to try out since they all are HTML and JavaScript and CSS web pages that interact with Elasticsearch via JSON calls like Kibana. You can open them locally and just connect to a remote Elasticsearch cluster API via URL, or you can install them to be hosted from your cluster directly.



Cluster Overview

Cluster Statistics

8 Nodes	252 Total Shards	252 Successful Shards	17 Indices	7,5

Cluster Health

Status	Green
Timed Out?	false
# Nodes	8
# Data Nodes	4
Active Primary Shards	126
Active Shards	252
Relocating Shards	0
Initializing Shards	0
Unassigned Shards	0

Indices

Index	# Docs				
logstash-2014.12.23	3				
logstash-2014.08.12	446,356,097				
logstash-2014.08.11	548,569,266				
logstash-2014.08.10	542,809,142				
logstash-2014.08.09	556,559,612				
logstash-2014.08.08	570,630,902	189.3GB	8	1	open
logstash-2014.08.07	560,633,862	187.3GB	8	1	open
logstash-2014.08.06	581,805,586	194.9GB	8	1	open
logstash-2014.08.05	487,935,675	165.2GB	8	1	open
logstash-2014.08.04	402,079,973	135.2GB	8	1	open
logstash-2014.08.03	553,938,928	183.3GB	8	1	open
logstash-2014.08.02	595,320,990	196.3GB	8	1	open
logstash-2014.08.01	550,991,874	184.8GB	8	1	open

Node Diagnostics Information

Mouseover each table cell for detailed information.

Summary

Node Name:	clogs-1.us-east-1.ec2	clogs-2.us-east-1.ec2	clogs-3.us-east-1.ec2	clogs-4.us-east-1.ec2	master_clogs-1.us-east-1.ec2	maste
IP Address:	100.64.222.15:9300	100.64.222.14:9300	100.64.222.16:9300	100.64.222.13:9300	100.64.222.15:9301	
Node ID:	B8mAggBwRJOf-ljgfJ0hnw	I3tCb3V1REuPPiRksUaDcQ	rd0JD3iFR4agbgi05futGg	b-Ug_FHTSFmAX3JH3EPR0Q	tOG7lHVcSNeuyNQJcp2q7w	kkpE
ES Uptime:	8.04 days	8.04 days	8.04 days	8.04 days	8.04 days	

File System

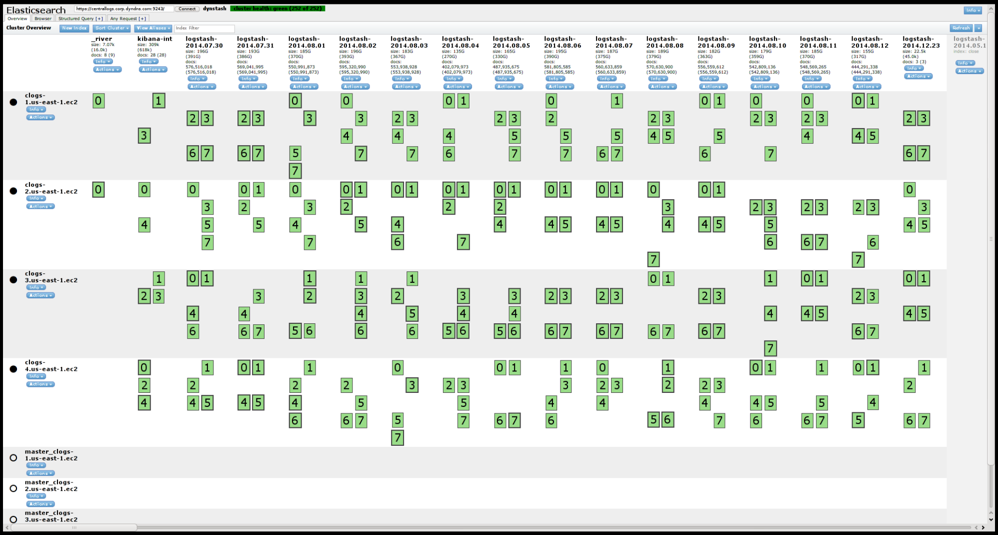
Store Size:	1.2TB	1.2TB	1.2TB	1.3TB	0.0	
# Documents:	3,793,967,075	3,745,431,228	3,707,503,114	3,834,988,826	0	
Documents Deleted:	0%	0%	0%	0%	0%	
Merge Size:	1.6TB	1.8TB	1.7TB	1.6TB	0.0	
Merge Time:	10:15:08	16:35:05	01:32:03	13:59:56	00:00:00	
Merge Rate:	2 MB/s	2 MB/s	2.4 MB/s	2.4 MB/s	0 MB/s	
File Descriptors:	520	510	524	525	339	

Index Activity

Elasticsearch HQ

Time: 0:15

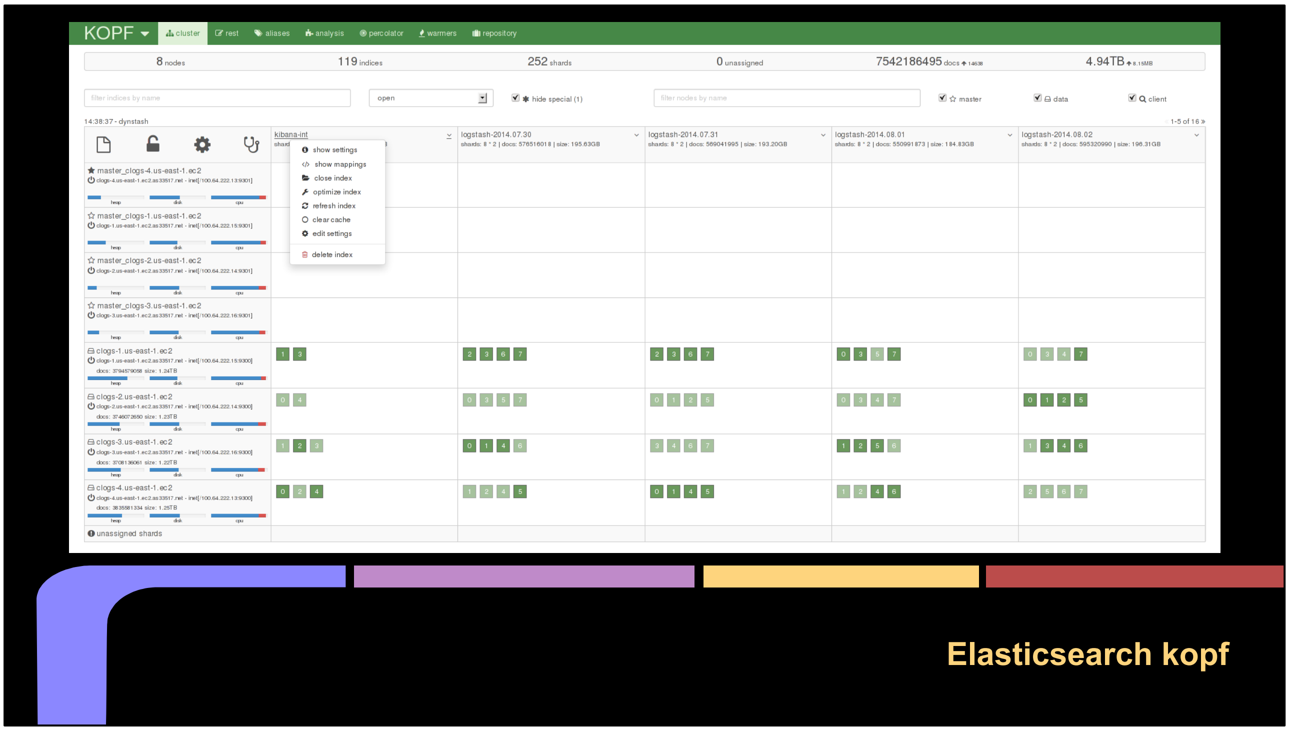
Here’s a couple of screenshots of Elasticsearch HQ. It’s one of the prettiest management interfaces for Elasticsearch, that is best for viewing the performance of the cluster and nodes, the statistics about the indices, etc.



Elasticsearch head

Time: 0:15

Elasticsearch head is the best way to view the distribution and migration/recovery of index shards between nodes. It also includes a great way to “browse” actual document data in the cluster as well.



KOPF cluster rest aliases analysis percolator warmers repository

8 nodes | 119 indices | 252 shards | 0 unassigned | 7542186495 docs ↑ 14638 | 4.94TB ↑ 8.15MB

filter indices by name | open | ✓ hide special (1) | filter nodes by name | ✓ master | ✓ data | ✓ client

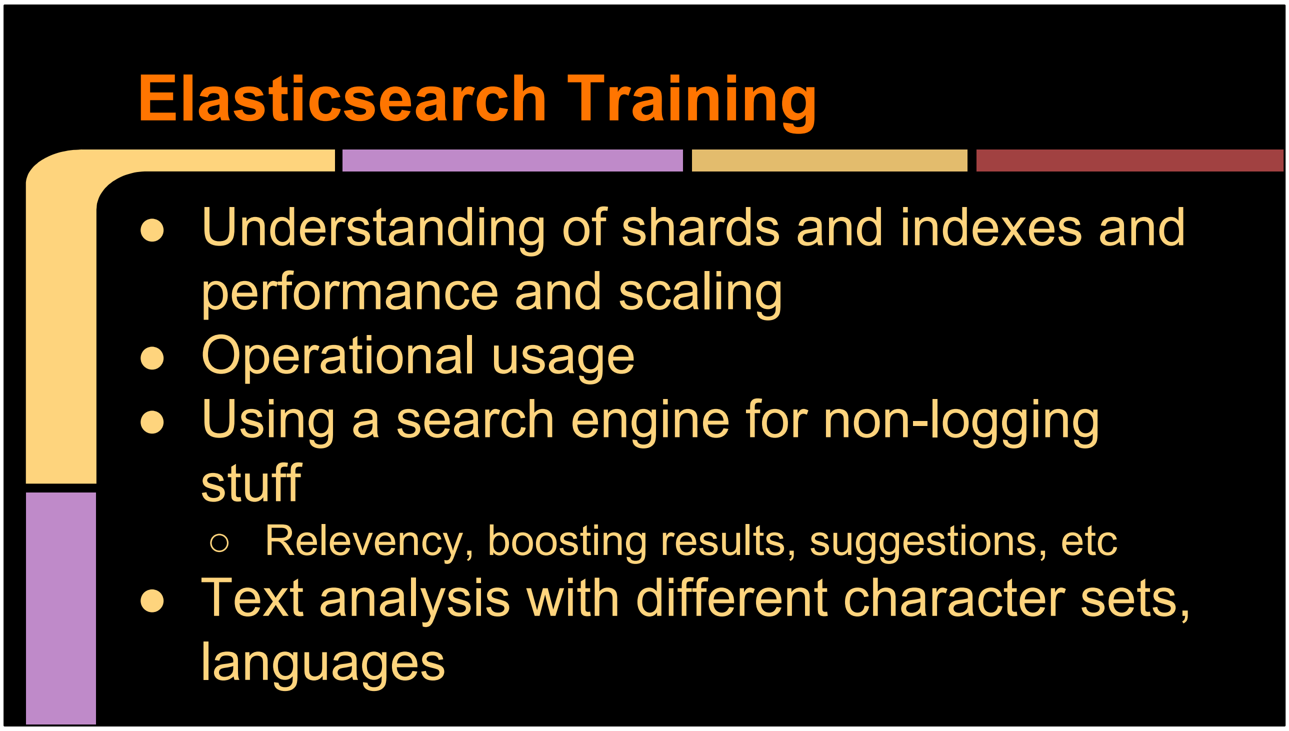
14:38:37 - dynstash 1-5 of 16 »

	kibana-int shard	logstash-2014.07.30 shards: 8 * 2 | docs: 576516018 | size: 195.63GB	logstash-2014.07.31 shards: 8 * 2 | docs: 569041995 | size: 193.20GB	logstash-2014.08.01 shards: 8 * 2 | docs: 550991873 | size: 184.83GB	logstash-2014.08.02 shards: 8 * 2 | docs: 595320990 | size: 196.31GB
master_clogs-4.us-east-1.ec2 clogs-4.us-east-1.ec2.as33517.net - inet[/100.64.222.13:9301] heap disk cpu					
master_clogs-1.us-east-1.ec2 clogs-1.us-east-1.ec2.as33517.net - inet[/100.64.222.15:9301] heap disk cpu					
master_clogs-2.us-east-1.ec2 clogs-2.us-east-1.ec2.as33517.net - inet[/100.64.222.14:9301] heap disk cpu					
master_clogs-3.us-east-1.ec2 clogs-3.us-east-1.ec2.as33517.net - inet[/100.64.222.16:9301] heap disk cpu					
clogs-1.us-east-1.ec2 clogs-1.us-east-1.ec2.as33517.net - inet[/100.64.222.15:9300] docs: 3794579058 size: 1.24TB heap disk cpu	1 3	2 3 6 7	2 3 6 7	0 3 5 7	0 3 4 7
clogs-2.us-east-1.ec2 clogs-2.us-east-1.ec2.as33517.net - inet[/100.64.222.14:9300] docs: 3746072650 size: 1.23TB heap disk cpu	0 4	0 3 5 7	0 1 2 5	0 3 4 7	0 1 2 5
clogs-3.us-east-1.ec2 clogs-3.us-east-1.ec2.as33517.net - inet[/100.64.222.16:9300] docs: 3708136061 size: 1.22TB heap disk cpu	1 2 3	0 1 4 6	3 4 6 7	1 2 5 6	1 3 4 6
clogs-4.us-east-1.ec2 clogs-4.us-east-1.ec2.as33517.net - inet[/100.64.222.13:9300] docs: 3835581334 size: 1.25TB heap disk cpu	0 2 4	1 2 4 5	0 1 4 5	1 2 4 6	2 5 6 7
unassigned shards					

Elasticsearch kopf

Time: 0:20

Kopf combines the visualization of shard distribution between nodes with a lot more management and configuration settings to affect how the cluster is configured, how the cluster handles recovery, management of aliases, percolators, warmers, etc.



Elasticsearch Training

- Understanding of shards and indexes and performance and scaling
- Operational usage
- Using a search engine for non-logging stuff
    - Relevency, boosting results, suggestions, etc
- Text analysis with different character sets, languages

Time: 0:30

There's a lot of Elasticsearch usage beyond logging, and their "Core Elasticsearch" Training goes into all of it. It may not be directly applicable to the task of logging, but the applicable material was worth it. The most valuable part of the training is that it's taught by engineers in Elasticsearch who know the answers to the questions you'll find yourself and others asking. The other people in the room are often large users of Elasticsearch who have valuable insights for you too. I learned as much from the experience and questions others brought up as I did in the class material.

Logstash Advice

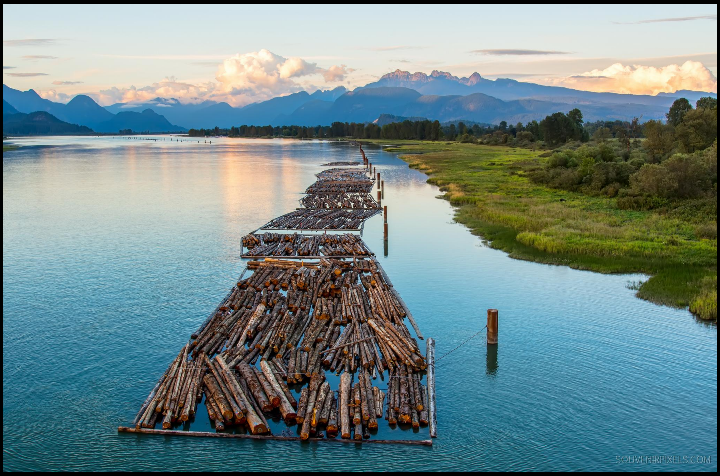
- Use it
- Be an expert logger

https://www.flickr.com/photos/minifig/95040634

Time: 0:15

In all seriousness, Logstash is really useful as a stream processor of events. Even if you go with Splunk or Graylog2, you may find a place to use Logstash. It will do whatever you can think of, and it's really easy to understand and make useful. I think I have a lot less advice for Logstash than RabbitMQ and Elasticsearch in my setup because it's just a lot more straightforward.

Questions?



https://www.flickr.com/photos/james_wheeler/9619984584