highscalability.com

Egnyte Architecture: Lessons learned in building and scaling a multi petabyte content platform - High Scalability -

36-46 minutes

This is a guest post by Kalpesh Patel, an Engineer, who for Egnyte from home. Egnyte is a Secure Content Platform built specifically for businesses. He and his colleagues spend their productive hours scaling large distributed file systems. You can reach him at @kpatelwork.

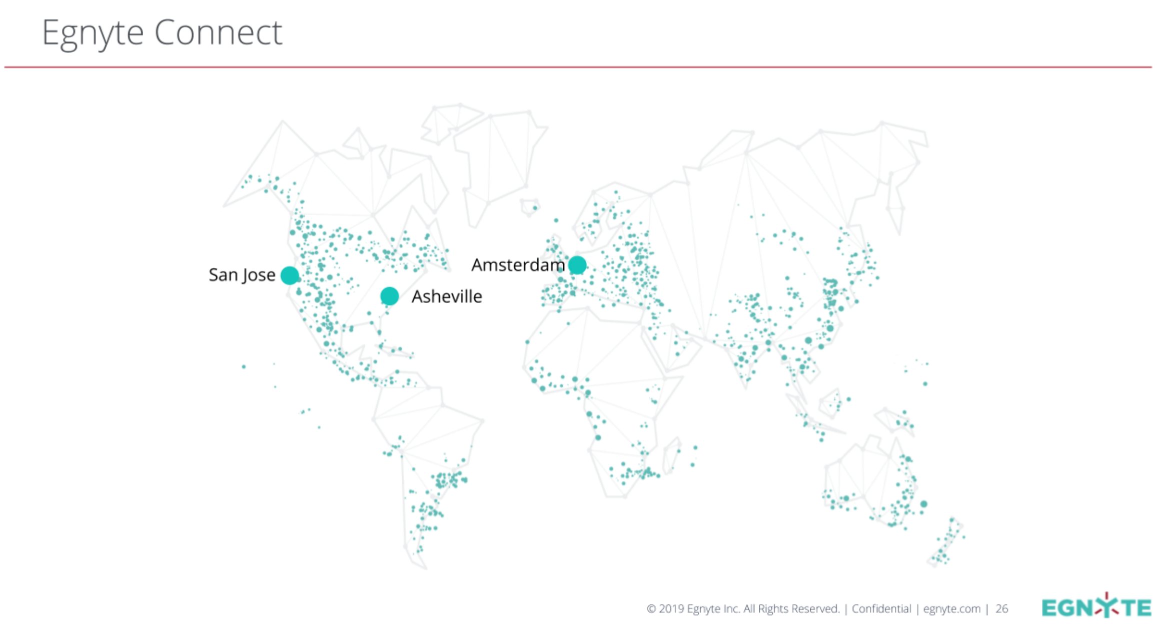
Introduction

Your Laptop has a filesystem used by hundreds of processes. There are a couple of downsides though in case you are looking to use it to support tens of thousands of users working on hundreds of millions of files simultaneously containing petabytes of data. It is limited by the disk space; it can’t expand storage elastically; it chokes if you run few I/O intensive processes or try collaborating with 100 other users. Let’s take this problem and transform it to a cloud-native file system used by millions of paid users spread across the globe and you get an idea of our roller coaster ride of scaling this system to meet monthly growth and SLA requirements while providing stringent consistency and durability characteristics we all have come to expect from our laptops.

Egnyte is a secure Content Collaboration and Data Governance platform, founded in 2007 when Google drive wasn't born and AWS S3 was cost-prohibitive. Our only option was to roll up our sleeves and build basic cloud file system components such as object store ourselves. Over time, costs for S3 and GCS became reasonable and with Egnyte’s storage plugin architecture, our customers can now bring in any storage backend of their choice. To help our customers manage ongoing data explosion, we have designed many of the core components over the last few years. In this article, I will share the current architecture and some of the lessons we learned scaling it along with some of the things we are looking to improve upon in the near future.

Egnyte Connect Platform

Egnyte Connect platform employs 3 data centers to fulfill requests from millions of users across the world. To add elasticity, reliability and durability, these data centers are connected to Google Cloud platform using high speed, secure Google Interconnect network.


Egnyte Connect
San Jose
Asheville
Amsterdam
© 2019 Egnyte Inc. All Rights Reserved. | Confidential | egnyte.com | 26
EGNYTE

Egnyte Connect runs a service mesh extending from our own data centers to google cloud that provides multiple classes of services:

Collaboration

- Document store
- Preview
- Video Transcoding
- Sharing
- Link
- Permissions
- Tags
- Comments
- Tasks
- Recommendations

Hybrid Sync

- On prem data processing
- Large files or low bandwidth
- Offline access
- Edge caching

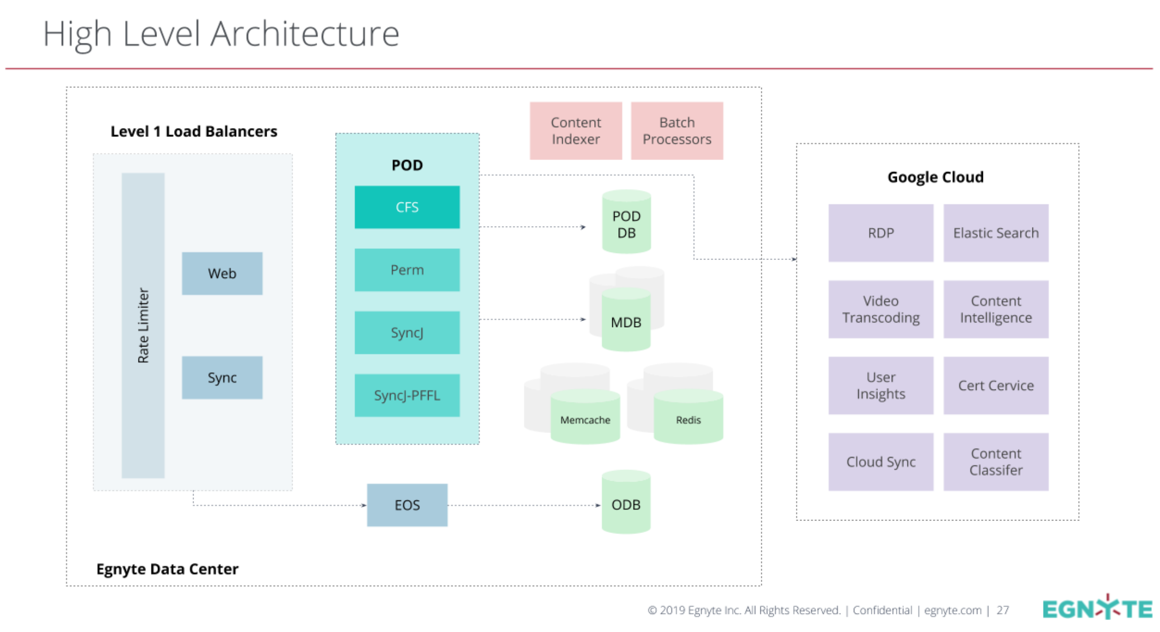
Infrastructure Optimization

- Migration to cloud

- Optimize on prem cold storage cost
- Consolidate repositories

In general, Egnyte connect architecture shards and caches data at different levels based on:

- Amount of data
- Data interdependence
- Level of concurrent reads
- Level of concurrent writes



Egnyte Connect Tech Stack

Cloud Platform

1. Google cloud

2. Azure
3. Hosted Data Centers

Languages:

1. Java
2. Python
3. Go
4. C

Object Stores

- Egnyte Object Store
- GCS
- S3
- Azure

Application Servers

- Tomcat

Databases

- MySQL
- Redis

- BigTable
- DataStore
- Elasticsearch

Caches

- Memcached
- Redis
- Nginx for disk based caching

Load Balancers / Reverse Proxy

- HAProxy
- Nginx

Message Queues

- Google Pub/Sub
- Rabbit
- Scribe
- Redis

Deployment Management

- Puppet

- Docker
- Ansible
- Jenkins
- Gitlab
- Kubernetes

Analytics

- New Relic
- OpenTSDB/bosun
- Grafana
- MixPanel
- Tableau
- BigQuery

Misc

- ZooKeeper
- Nagios
- Apache FTP server
- Kong
- ReactJS/Backbone/Marionette/JQuery/npm/Nightwatch

- Rsync
- PowerDNS
- Mashery
- SOA architecture based on REST APIs.
- Java used to power core file system code
- Python used to power client-side code, certain microservices, migration scripts, internal scripts
- Native Android and iOS apps
- Native desktop and server hosted clients that allow both interactive as well as hybrid sync access to the entire namespace

Stats

- 3 primary regions with one in Europe connected to respective GCP regions using Google Interconnect
- 500+ Tomcat service instances
- 500+ Storage nodes powered by Tomcat/Nginx
- 100+ MySQL nodes
- 100+ Elasticsearch nodes
- 50+ Text extraction instances(autoscaled)
- 100+ HAProxy instances

- Many other types of service instances
- Tens of petabytes of data stored in our servers and other object stores such as GCS, S3 and Azure Blobstore
- Multiple terabytes of extracted content indexed in Elasticsearch
- Millions of desktop clients syncing files with the cloud for offline access
- Millions of desktop clients accessing files interactively

Getting To Know You

What is the name of your system and where can we find out more about it?

Egnyte Connect is the content collaboration and data management platform. CFS(cloud file system), EOS (Egnyte object store), Content Security, Event Sync, Search Service, User behavior based recommendation service form major parts of the system. You can find more about these in the For The Techies section at our blog.

What is your system used for?

Egnyte Connect as the content collaboration and data management platform is used by thousands of customers as the single Secure Content Platform for all of their document management needs. It is built for hosting a variety of file services and to cloud-enable their existing file repositories. It can be accessed from a variety of endpoints like FTP, WebDAV, mobile, public API, and browsers and features strong audit and security components.

Why did you decide to build this system?

In 2007, businesses had started to become more distributed; customers were using multiple devices to access their files and there was a need to make this experience as smooth as possible. We built Egnyte Connect - a secure distributed file system that combines Hybrid Sync with Cloud File System to answer content collaboration needs of businesses in wide variety of business needs. With the fragmentation of data across on-premises and cloud repositories, along with increasing compliance needs due to initiatives such as GDPR, we built Egnyte Protect to help our customers satisfy their compliance and governance needs.

How is your project financed?

Egnyte was initially bootstrapped company. We later went on and raised $137.5 million in multiple rounds from Goldman Sachs, Google Ventures, KPCB, Polaris Partners and Seagate.

What is your revenue model?

Egnyte does not offer free accounts. Customers start with a 15-day free evaluation trial period and after that, they convert to paid account with revenue model based on number of seats, storage and other enterprise features.

How do you market your product?

We started with SEM/SEO but over time as we grew, we used many channels to acquire customers like Social media, Biz dev, Trade shows, SEM, SEO, Inbound marketing and high touch sales for Enterprise customers.

How long have you been working on it?

Egnyte was founded in 2007. It is 12 years old currently and cash flow positive.

How big is your system? Try to give a feel for how much work your system does.

We store multibillion files and tens of petabytes of data. In “Egnyte connect” we observe more than 10K API requests per second on average as per New Relic with avg response time of <60ms. We allow access from 3 primary regions due to safe harbor rules and location proximity. More on this is in the stats section. Our “Egnyte Protect” solution also continuously monitors the content and access for compliance, governance and security breaches for many of our customers.

How many documents, do you serve? How many images? How much data?

We store multibillion files and tens of petabytes of data. We store all kinds of files. Top 5 file extensions stored by Egnyte are pdf, doc/docx, xls/xlsx, jpeg, and png.

What is your ratio of free to paying users?

All our users are paid users. We offer a free 15-day trial and after that, they convert to paid account.

How many accounts have been active in the past

month?

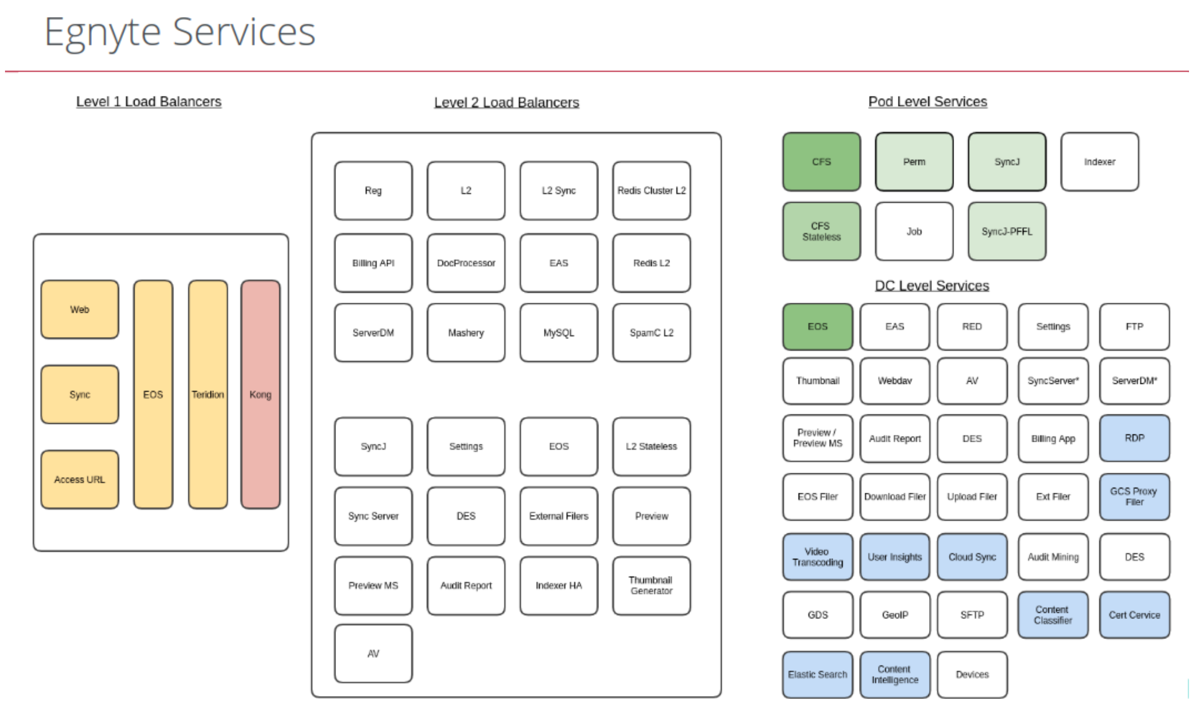
All of our customers are paid accounts and almost everyone is active during the month. We power their secure content platform needs, Who doesn’t use electricity at home?

How Is Your System Architected?

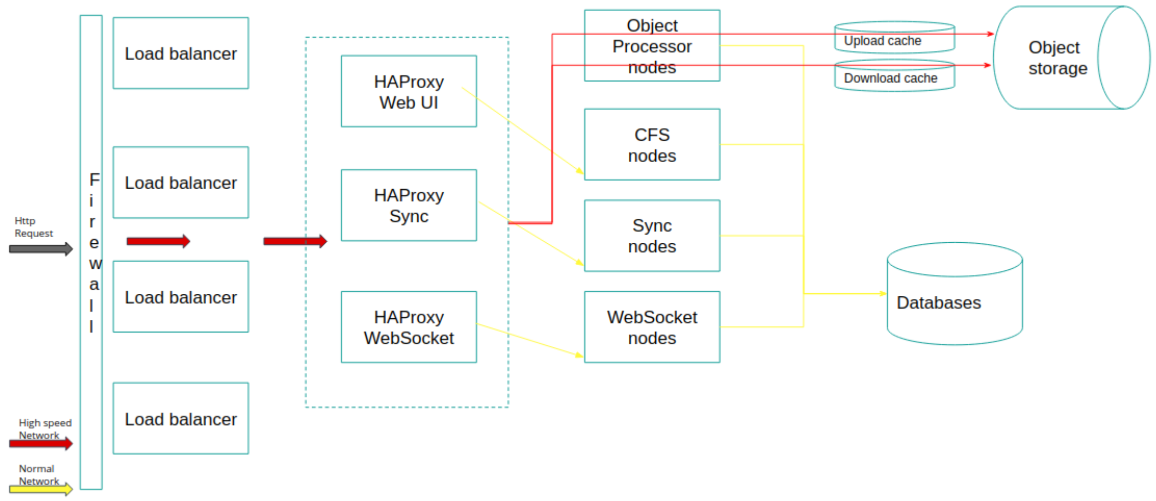
What is the architecture of your system?

We use a service-oriented architecture based on REST and it allows us to scale each service independently. This also allows us to move some of the backend services to be hosted in the public cloud. All services are stateless and use databases or our own object store for storage.

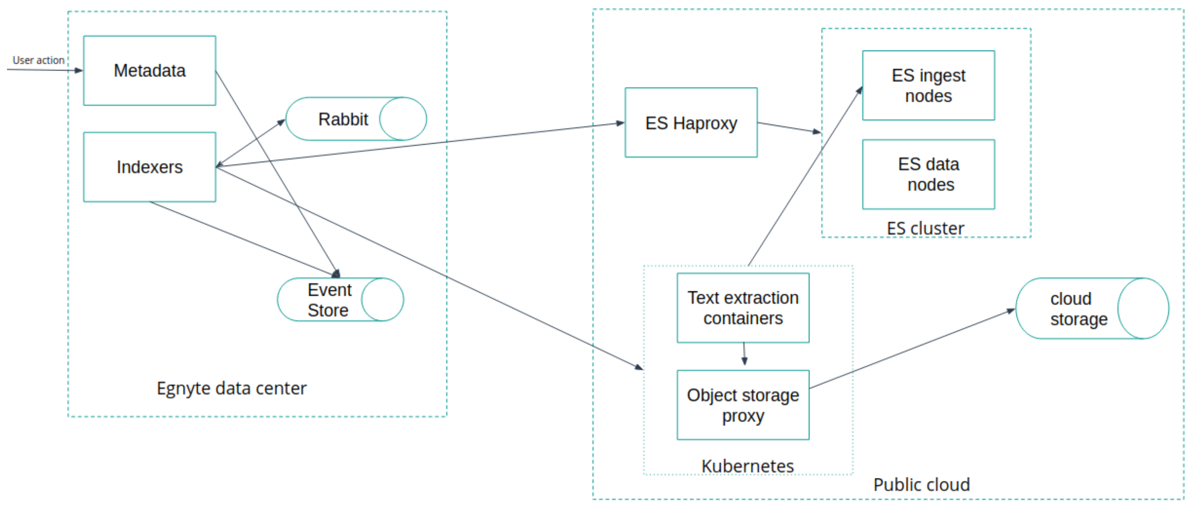
A 10000ft overview of Egnyte Connect services looks like below.



A 10000ft overview of typical request flow looks like below



A 10000ft overview of Search architecture looks like below



What particular design/architecture/implementation challenges does your system have?

Some of the biggest architecture challenges are:

1. Scaling the file storage frugally
2. Scaling the metadata access
3. Real-Time sync of files to desktop clients
4. Bandwidth optimization
5. Impact isolation
6. Caching (distributed and in-memory)
7. Feature rollout

What did you do to meet these challenges?

1. For storage, we wrote our own and now we use a pluggable storage architecture to store to any public

cloud-like S3, GCS, Azure...

2. To scale metadata, we moved to Mysql and started using sharding. At some point, we were throwing more hardware temporarily to get some breathing room in order to peel the layers of 'scaling onion' one by one.
3. For real-time sync, we had to change our sync algorithm to work more like Git where the client receives incremental events and tries to do eventual consistent sync with cloud state.
4. For Feature rollout, we built a custom settings service that allows engineers to write code behind a feature flag. This way, you can release your code even in sleeper mode and collect data and then enable the features by customer, user or by a host group or by a POD or by a data center. With this level of control, even a new engineer can confidently write their code behind a feature flag and release it to production without worrying about downtime.
5. Monitor, Monitor, and Monitor. You can't optimize what you can't measure. On the downside, at some point, we were monitoring too much to the extent that we were not able to focus on all metrics. We had to shift focus and rely on anomaly detection tools like New Relic,

bosun, ELK, OpenTSDB and custom reports to allow us to focus on problems that are or about to trend from green->yellow->red. The intent is to catch them when they are yellow and before customer notices.

How does your system evolve to meet new scaling challenges?

We have re-architected many layers many times. I will try to list the few iterations of core metadata, storage, search layers over the last 7 years.

1. Version 1: files metadata in Lucene, files stored in DRBD Filers mounted via NFS, search in Lucene. Chokepoint: Lucene updates were not real-time and it had to be replaced.
2. Version 2: files metadata in Berkeley DB, files stored in DRBD Filers mounted via NFS, search in Lucene. Chokepoint: We broke the limits of NFS and it was choking left and right and it had to be replaced with HTTP.
3. Version 3: files metadata in Berkeley DB, files stored in EOS Filers served via HTTP, search in Lucene. Chokepoint : Even sharded Berkeley DB was choking under the stress and there was a database crash with

recovery taking hours, it had to be replaced.

4. Version4: files metadata in MySQL, files stored in EOS Filers served via HTTP, search in Lucene. Chokepoint: The public cloud started becoming cheaper.
5. Version5: files metadata in MySQL, files stored in EOS/GCS/S3/Azure and served via HTTP, search in Lucene. Chokepoint: Search started choking and had to be replaced.
6. Version6: files metadata in MySQL, files stored in EOS/GCS/S3/Azure served via HTTP, search in Elasticsearch. This is the current architecture.
7. Version7 (Future): Move all compute to the public cloud, carve out more services for impact isolation, dynamic resource pooling to manage pets and cattle efficiently.

Do you use any particularly cool technologies or algorithms?

- We use exponential backoffs when calling between core services and services to have circuit breakers to avoid the thundering herd.
- We use a fair-share allocation on core service node resources to incoming requests. Each incoming request

on the core service node is tagged and classified into various groups. Each group has a dedicated capacity and if one customer is making 1000 requests per second and the other is making 10 request then this system would ensure that the other customers would not starve due to noisy neighbor issues. The trick is that if you are the only customer using the system at the moment you can go full throttle but as more customers come at the same time you share the capacity among them. For some large customers, we carve out dedicated pools to ensure a consistent response time.

- Some of the core services with SLA are isolated in PODs and this ensures that one bad customer won't choke the entire data center, but this may soon require reincarnation.
- We use event-based sync in our desktop sync client code, as server events are happening they get pushed to the client from server and the client replays them locally.
- We employ large scale data filtering algorithms to let large clusters of clients synchronize with Cloud File System.

- We use different types of caching techniques depending on the problem statements. Few flavors are:
- Traditional
- In Memory - Simple
- Immutable objects
- In memory large datasets
- Locks used for strong consistency across processes
- Partial updates implemented to avoid GC
- In Memory - High volume mutating datasets
- Coarse grained invalidation
- Fine grained invalidation
- Disk based caching

What do you do that is unique and different that people could best learn from?

Focus on the core capability of your startup and if you are facing technically hard problems and you have to build something custom for it then roll up the sleeves and go for it. There are many unique things but the storage layer, event-based sync is definitely worth learning, here are more details on it Egnyte object store and Egnyte Canonical File System.

What lessons have you learned?

- You can’t optimize what you can’t measure: Measure everything possible and relevant and then optimize parts of systems that are used 80% of the time first.
- When you are small, introduce technologies slowly, don’t try to find the perfect tool out there for the problem you have in hand. Coding is the easiest part of the lifecycle but its maintenance like deployment/operations/learning curve will be hard if you have too many technologies. As you become bigger, you would have enough fat in your deployment to divide into services as you go along. Learn to keep one or two service templates to implement microservices and don’t go wild on using different tech stack for each service.
- As a startup, sometimes you have to move fast. Introduce the solution that you can do best right now and re-architect it over time if you see traction. You may try 10 different approaches and only 1 may see traction, so do something fast and the ones where you see traction, re-architect them to meet the scale of the business.
- Look for a single point of failure and hunt them down

relentlessly. Put an extra effort to fix problems that keep you up at night and go from defensive to offensive mode as soon as possible.

- In SOA, build circuit breakers to shed load early and start sending 503s if your service is choked. Instead of penalizing everyone, see if you can do a fair share allocation of resources and penalize only the abusive requests.
- Add auto-heal capability in service consumers, a service can choke and the consumers like the desktop clients or other services can do exponential backoff to release pressure on server and auto-heal when the service is functional again.
- Always be available: Have a service level circuit breaker and a circuit breaker by customer. For e.g., if accessing file system over WebDAV or FTP has performance issues, and it will take 4 hours to fix, then for those 4 hours, you can just kill FTP/WebDAV at kong/firewall and ask customers to use web UI or other mechanisms to work. Similarly, if one customer is causing an anomaly that is choking the system then temporarily disable that customer or service for that customer and re-enable it when issue is fixed. We use feature flags and circuit breakers for this.

- For highly scalable services, going outside of java process is costly, even to go to Memcache or Redis so we do in-memory cache with varying TTL for some highly used data structures like access control computation, feature flags, routing metadata etc.
- There is a pattern we see about processing Big datasets. Its futile to optimize the code and squeeze every drop out of the lemon, we may end up making the code complex and the lemonade bitter. Quite often, the simplest solutions come from going back to the drawing board and seeing if we can:
- Reduce the dataset size that we need to operate on by using heuristics
- Reorganizing the way we store data in memory or on disk.
- Denormalizing the data on write and avoiding joins.
- Time based filtering like Archiving older data.
- Creating smaller shards in multi-tenant data structures.
- Use events to update the cache dataset instead of a full reload.
- Keep it simple: New engineers join every month so the goal is to have them productive from week one - a

simple architecture ensures easy induction.

Why have you succeeded?

Traction trumps everything. We reached product/market fit when the EFSS market was just exploding. The timing with good execution, customer-first-focus, financial discipline by management team lead to success. A lot of competitors went to the freemium model and raised a boatload of money but we were charging from day one and this allowed us to focus on growing the solution/team as the market demand scaled up. Being focused on paid customers allowed us to deliver an enterprise-class solution without paying the freemium penalty.

What do you wish you would have done differently?

I wish the public cloud was not as cost-prohibitive when we started. I also wish we were on SOA from day one, it took us some time to reach there but we are there now.

What wouldn't you change?

Architecture should be malleable. Four years ago, I had a different answer for a given problem but at this moment, I am not sure. I mean that as your scale grows then design patterns and strategies that used to work 2 years ago and allowed you to go from defensive to offensive positioning may buckle under pressure or becomes cost-prohibitive. As long as the change will allow the system to become resilient or bring 10x change and buy us another 3-4 years of scale, I would go ahead and try to change it. I can't comment 2 years from now, I would have the same thoughts, they may change. The architecture changes as you encounter the next growth spurt.

How much upfront design should you do?

Excellent question. The answer is “it depends”,

- If you are designing something like a core storage layer or core metadata layer then adding 2 more weeks to your design won’t hurt much. When we were migrating from Berkeley DB to MySQL on our core metadata layer, I was under pressure and I had thought of taking a shortcut, when I ran it through our CTO, he advised on taking a bit more time and “Doing the right thing” and as a retrospective that was an excellent decision.
- For a public API, it's good to do a decent front design

as you won’t get a second chance to change it and you will have to maintain it for the next 4-5 years.

- If its petabytes of data and migrating it would be a huge pain then I would give it even a month more and do more POCs.
- However, if you are designing something for internal service and migrating it to a new architecture won’t be a year-long then I advise doing very minimal front design and just build the version quickly and iterate on it as the usage grows.

How are you thinking of changing your architecture in the future?

- Moving from static PODs to dynamic PODs and handle pets as well as cattle seamlessly.
- Carve out more resilient services and isolate the impact.
- Move our entire compute to the public cloud while keeping the data center for serving files. This will allow us to autoscale up/down as load comes. We already use the public cloud to autoscale some async processing like Video Transcoding, text extraction, data migration, Search, etc.

- Once in the cloud, use more of the autoscaled services like BigTable, PubSub, Spanner, ...
- Move our deployment from VMs to containers in Kubernetes for pending services.
- Automate schema management for some remaining databases
- Remove joins from some of the fastest-growing tables by rearchitecting.
- Rewrite the caching layer for metadata and use Redis data structures instead of Memcache.
- Convert heavily used flows from strongly consistent to eventually consistent.

How Is Your Team Setup?

How many people are in your team?

Around 700 employees and contractors. There are 200 Engineers(DevOps/OPS/QA/Developers/…), the rest are sales, marketing, support, product management, HR, etc.

Where are they located?

A fairly distributed engineering team at the start but now gravitating mostly in Mountain View, Poland, India. Some remote employees like myself and a handful of others work from home.

Who performs what roles?

It's a big team, we have Product managers, UX team, DevOps, scrum teams, architects, engineers performing various roles. Initially at the start engineering team was flat and everyone would report to VP of engineering but now we have added a layer of management in between.

Do you have a particular management philosophy?

If you develop something then you own the lifecycle of that product, which means you would work with QA, DevOps to ensure it’s tested/deployed. When it goes to production you would monitor it using the various internal tools like New Relic/Grafana, Kibana and if there is a regression you would fix it. I am also a big fan of 1 person 1 task philosophy, this way if the engineer runs into a wall he would find some way to overcome it eventually rather than giving up too early.

If you have a distributed team how do you make

that work?

Autonomy, 1-1 communication, give them challenging problems, care personally and challenge directly and they would be motivated.

What is your development environment?

- Ubuntu for server teams
- UI team uses Windows/mac and connect to local Ubuntu VM for REST API server or connect to shared QA instance
- Eclipse/Idea
- AWS for builds
- Maven
- docker
- Gitlab
- Jenkins
- Confluence
- JIRA
- Google office suite
- Slack

What is your development process?

We use Scrum and have weekly releases for the cloud file system team. We use a variant of git-flow, for every ticket, we clone the repo and we run automation tests on every merge request. A merge request has to be approved by 2 engineers and only then JIRA ticket can be resolved. Once it's resolved our pipeline takes over and the ticket catches the next release train. The next release train is verified by automated REST API tests and some manual smoke tests.

We eat our own dogfood and the code goes to UAT (used by all employees) 2-3 days before release, we catch any surprises not detected by automated tests. We do a production deploy every Wednesday and monitor new relic, exception reports daily for any anomalies. We changed deployment in the middle of the week for both work-life balance and also this way we would have all engineers available in case release runs into an issue.

If it's a long-running feature then the Engineers would usually work behind a feature flag and commit the code in sleeper mode in various phases so his code is tested every week instead of a big bang. We handle large migrations also the same way where we migrate 1 customer at a time and turn on the feature for that customer only, some of our large migrations have run for 3-4 months.

Is there anything that you would do different or that you have found surprising?

Many engineers work from home and it's surprising to see given autonomy, many remote employees are as productive and motivated as the HQ employees.

What Infrastructure Do You Use?

Which languages do you use to develop your system?

Java/Python mostly and some small services in Go/C

How many servers do you have?

We have ~3000+ instances managed by puppet.

- 500+ Tomcat service instances
- 500+ Storage nodes powered by Tomcat/Nginx
- 100+ MySQL nodes
- 100+ Elasticsearch nodes

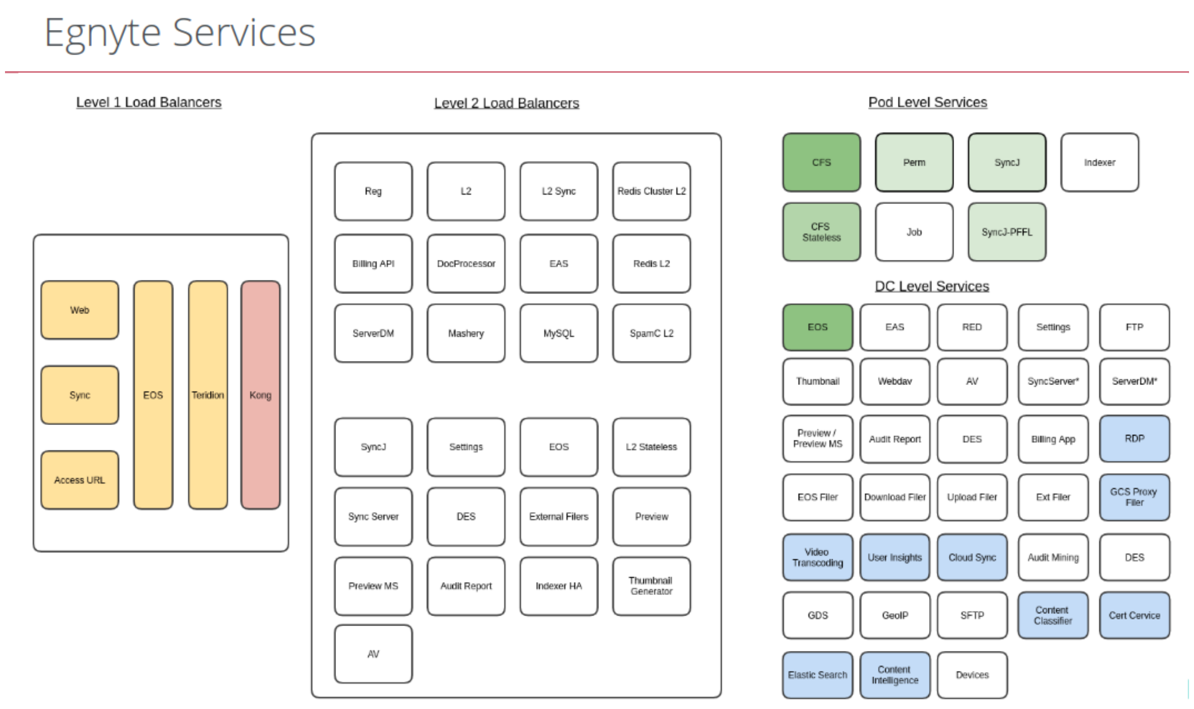
- 50+ Text extraction instances(autoscaled)
- 100+ HAProxy instances
- and many other types of service instances

How is functionality allocated to the servers?

We use a service-oriented architecture and servers are allocated based on the type of service. Some of the top-level services are :

- Metadata
- Storage
- Object service
- Web UI
- Indexing
- Sync
- Search
- Audit
- Content Intelligence
- Real-Time event delivery
- Text extraction
- Integrations

- Thumbnail generation
- Antivirus
- Spam
- Preview/Thumbnail
- Rsync
- API gateway
- Billing
- FTP/SFTP
- and many more ….



How are the servers provisioned?

Most of the services are puppetized and run on VM, we run physical for only a few of the things like MySQL, Memcached, and storage nodes. We use a third-party that provision the servers based on a template and put it in the data center and make it available for use to use. But we have started work to migrate everything to the public cloud so eventually, everything would work in Kubernetes. The challenge, however, is how do you change the engine of a race car while you are in the race without downtime.

What operating systems do you use?

CentOS7

Which web server do you use?

Nginx, HAproxy. Apache is used in some old flows and will get deprecated over time.

Which database do you use?

MySQL and Redis. We had used other databases like Berkeley DB, Lucene, Cassandra in past but we migrated over time all of them to MySQL because of its engineer/ops familiarity and scalability. More on this can be found at MySQL at Egnyte.

We also use OpenTSDB, BigTable, Elasticsearch for some of the flows.

Do you use a reverse proxy?

Yes Nginx and HAProxy

Do you collocate, use a grid service, use a hosting service, etc?

We collocate and we also use a public cloud.

What is your storage strategy?

We started by creating our own servers and packing as many hard drives as possible in a machine, we used to call them as DRBD Filers. We did this as AWS was cost-prohibitive. We had evaluated GlusterFS but it wasn't scaling to meet our needs at that time so we built our own. Overtime S3 became cheap and GCS/Azure was born and we had architected the storage layer to be pluggable so now customers can decide which storage engine they want to use (Egnyte, S3, GCS, Azure, ….). At this point, we store 1 DR copy in public cloud and 1 copy with us but eventually we will use our data center as a pass-through cache as compute is cheaper in the cloud but bandwidth is expensive.

How do you grow capacity?

We have semi-automated a capacity planning tool based on data from Newrelic, Grafana and other stats and we do regular capacity planning sessions, based on those we watch the key indicators in our monitoring reports and pre-order some extra capacity. Some services are now cloud-enabled and we just autoscale them based on queue size.

Do you use a storage service?

Yes Egnyte, S3, GCS, Azure,

How do you handle session management?

We rewrote our architecture many times and currently, 99% of the services are stateless. Only the service serving web UI uses session, we use sticky sessions in tomcat backed by memcached-session-manager but eventually, my plan is to make this also stateless using JWT or something like that.

How is your database architected? Master/slave? Shard? Other?

We use Master-Master replication for almost all the databases with automatic failover, but switchover on some of the heavily mutating databases are manually done, we had encountered some issues where automatic switch would cause application data inconsistency due to replication lags and we need to re-architect some of core filesystem logic to fix this, we would eventually get this done. More details at length on database architecture are answered below in question about handling database upgrades.

How do you handle load balancing?

We geo balance customers based on the IP they are accessing the system using DNS and within a data center they are routed to their corresponding POD using HAProxy and inside POD they are again routed using HAProxy

Which web framework/AJAX Library do you use?

We have changed UI many times and this is one thing that is always in flux. In the past, we had to use ExtJS, YUI, JQuery and what not. The latest iteration is based on ReactJS/Redux and some legacy code on Backbone/Marionette.

Which real-time messaging frameworks do you use?

We use Atmosphere but eventually, we would replace it with NodeJS

Which distributed job management system do you use?

We use Google Pubsub, RabbitMQ and Java/Python based consumer services for this.

Do you have a standard API to your website? If so, how do you implement it?

Our API is classified into 3 types:-

1. Public API: This is the API we expose to third party app engineers and integration team and our Mobile app. We deprecate/upgrade API signature following proper deprecation workflow and changes are always backward compatible. We use Mashery as a gateway, the API is documented at https://developers.egnyte.com/docs
2. API for our clients: This API is internal to our clients and we don't guarantee backward compatibility if someone other than us uses this.

3. Internal protected API between services: This is the API used internally within our data centers by services to talk to each other and this can’t be called from the outside firewall.

What is your object and content caching strategy?

We store petabytes of data and we can’t cache all of it but if a customer has 50 million files on a given 15 day period he might be using only 1 million of them. We have cache filers nodes based on tomcat/Nginx/local file system and it acts in LRU fashion. We can elastically increase decrease the no of cache filer servers. One of our biggest problems is upload speeds, how do you upload data as fast as possible to Egnyte from any part of the world, for this we built special Network pops, if you are curious you can read more on it at Speeding Up Data Access for Egnyte Customers

Memcached/Redis is used for caching metadata, we use separate Memcached pool for caching long-lived static data and file system metadata. The core file system metadata is huge, won’t fit in current Memcached nodes and would evict the recently used other kinds of data. to prevent this we use 3 kinds of pools and application code decides where to look for what kind of data. We allow evictions in filesystem Memcached cache and strive for zero evictions in other kinds of Memcached pools. We also use different object expiry for different kinds of data. For some of our highly used data like customer information or shard mapping even going to Memcache for every request would slow us down for some requests like listing of folders, so we do in-memory caching of this data on each JVM and the data is flushed based on a custom TTL or we use some pub-sub mechanism to flush it.

Two of the biggest pain in caching are permissions and events. For permissions data, we have rearchitected that layer many times and recently we wrote a TRIE to cache this efficiently.

For events, we cache them in Memcache but it can happen that during the night some 100K events were published for a customer and in the morning suddenly at 9:00 AM 30K people opened their laptop and now everyone wants those 100K events to make their system consistent. This is an interesting scale problem as this would require you to process 30B events in a short duration like 15 min and only sending events that the users have permissions to them. As events are immutable we were caching them in Memcache for 12 hours but even them downloading the same events so many times from Memcache was causing network issues. Eventually, we resorted to caching the events in memory for a short duration and also tuning the GC settings on those nodes as we are doing a lot of young generation collections. We also put these nodes on a faster network compared to other nodes and we still aren't done with this problem :).

What is your client-side caching strategy?

For our web UI, we use requireJS and various other ways to download only the required modules. Our Mobile and Desktop clients are rich use the local filesystem as a cache.

Which third party services do you use to help build your system?

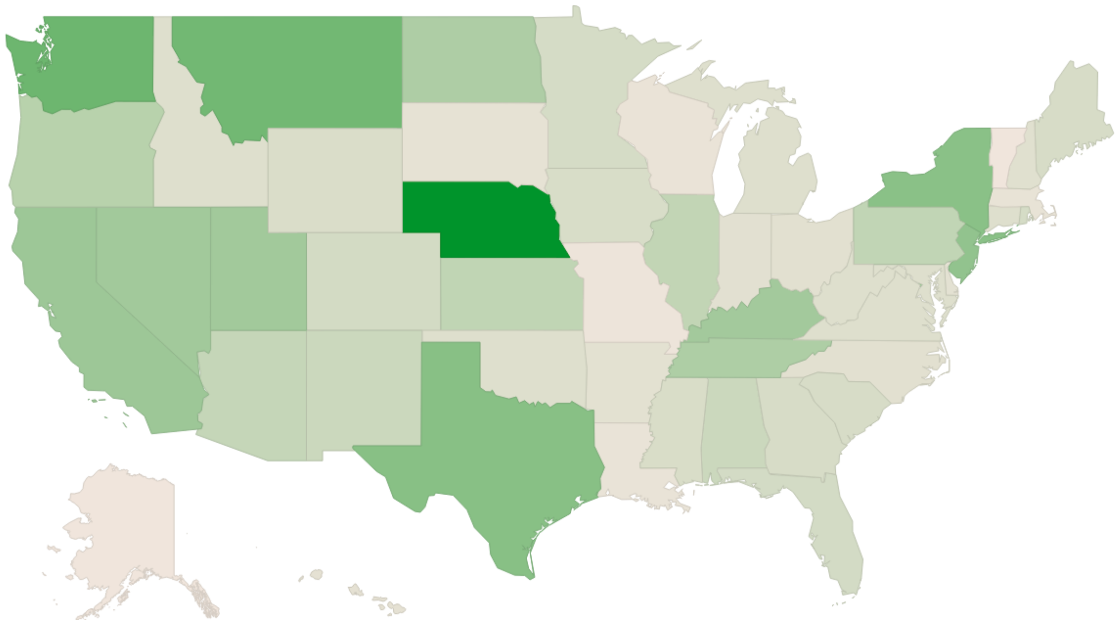
Google compute, Azure, New Relic, Authy, MixPanel, Flurry, Tableau are some services we use but most of the core components are built by us.

How Do You Manage Your System?

How do you check global availability and simulate

end-user performance?

We use nodes in different AWS regions to test bandwidth performance consistently. We also use internal haproxy reports to plot upload/download speeds observed by the customer and proactively hunt them and use network pops and other strategies to accelerate packets.



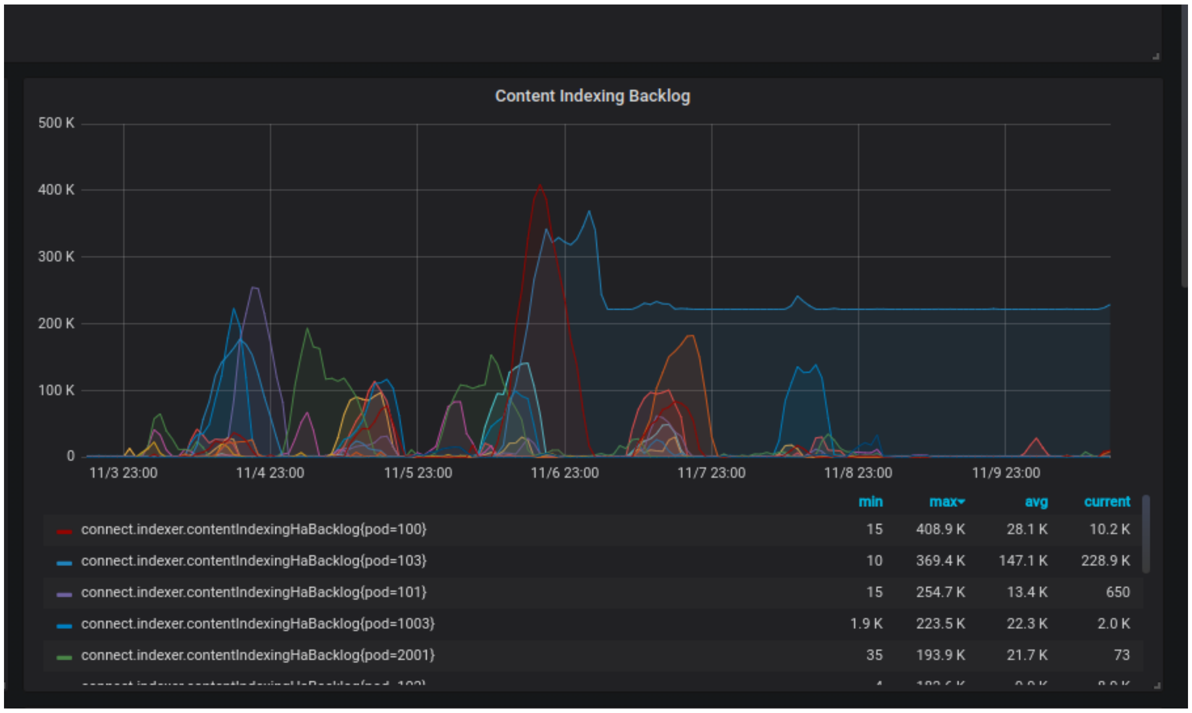
How do you health check your server and networks?

Nagios, Grafana and New Relic and some internal proactive exception analysis are used. More details on it are in this blog post

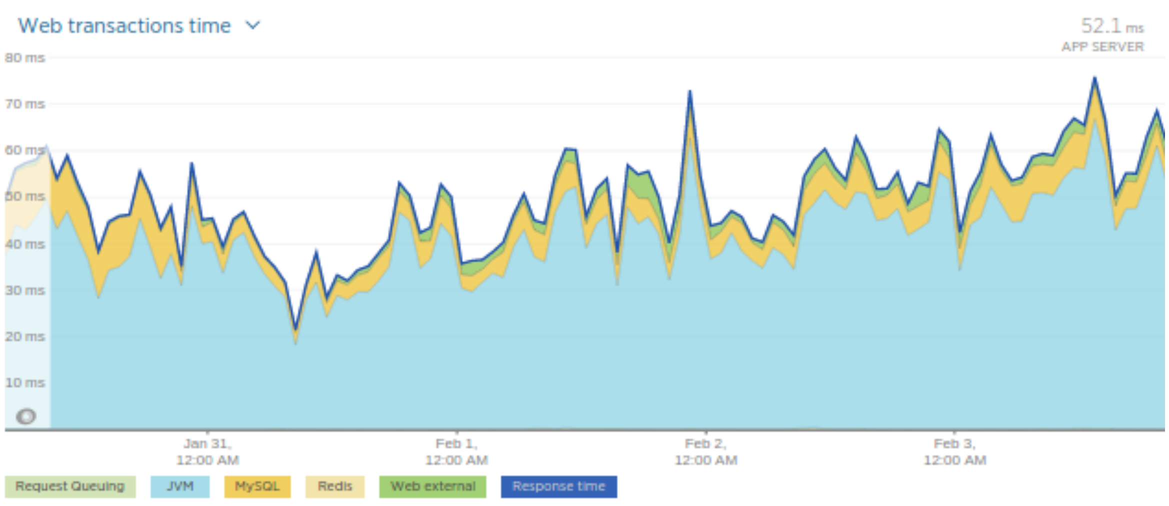
How you do graph network and server statistics

and trends?

We use Grafana, Kibana, Nagios and New Relic.



	min	max	avg	current
connect.indexer.contentIndexingHaBacklog{pod=100}	15	408.9 K	28.1 K	10.2 K
connect.indexer.contentIndexingHaBacklog{pod=103}	10	369.4 K	147.1 K	228.9 K
connect.indexer.contentIndexingHaBacklog{pod=101}	15	254.7 K	13.4 K	650
connect.indexer.contentIndexingHaBacklog{pod=1003}	1.9 K	223.5 K	22.3 K	2.0 K
connect.indexer.contentIndexingHaBacklog{pod=2001}	35	193.9 K	21.7 K	73



How do you test your system?

Selenium, Junit, Nose, Nightwatch and manual testing. Combination of unit, functional, integration and performance tests.

How you analyze performance?

New Relic is used to monitor the Application performance. We also generate quite a bit of internal application metrics using a home grown framework. We use Grafana/Nagios/Kibana, internal tools and other tools to monitor performance for other parts of the system. More details on this are in this blog post Debugging Performance Issues in Distributed Systems

How do you handle security?

The dedicated Security team runs automated security benchmark tests before every release. Continuous automation pen tests are running in production. We also use bug bounty programs and engage in whitehat testing companies. Some customers do their own security testing using third parties.

How Do You Handle Customer Support?

We have a dedicated 24X7 distributed Customer success team, we use Zendesk and JIRA

How Do You Decide What Features To Add/Keep?

Do you implement web analytics?

We use Google Analytics, Mixpanel, Flurry to measure feature usage

Do you do A/B testing?

Yes, we use feature flags to do A/B testing. More on this is Using feature flags at Egnyte

How many data centers do you run in?

3 primary data centers, including one in Europe (due to safe harbor rules) and network-pops all around the world.

How is your system deployed in data centers?

Puppet/Ansible is used for deploying most of the new code.

Which firewall product do you use?

Palo Alto networks

Which DNS service do you use?

NS1

Which routers do you use?

Cisco

Which switches do you use?

Arista

Which email system do you use?

We use a combination of SendGrid and our own SMTP servers.

How do you backup and restore your system?

For MySQL, we use Percona XTraBackup , for Elasticsearch the data is replicated 3 times. For customer files, we replicate them 3 times and 1 copy is stored in a DR public cloud. If a storage Filer fails to recover, we discard it, add a new Filer and replicate the copies again. For some customers, we additionally replicate their data to the provider they choose. For customers using S3, Azure or GCS as a pluggable storage layer it will ensure replication to prevent data loss.

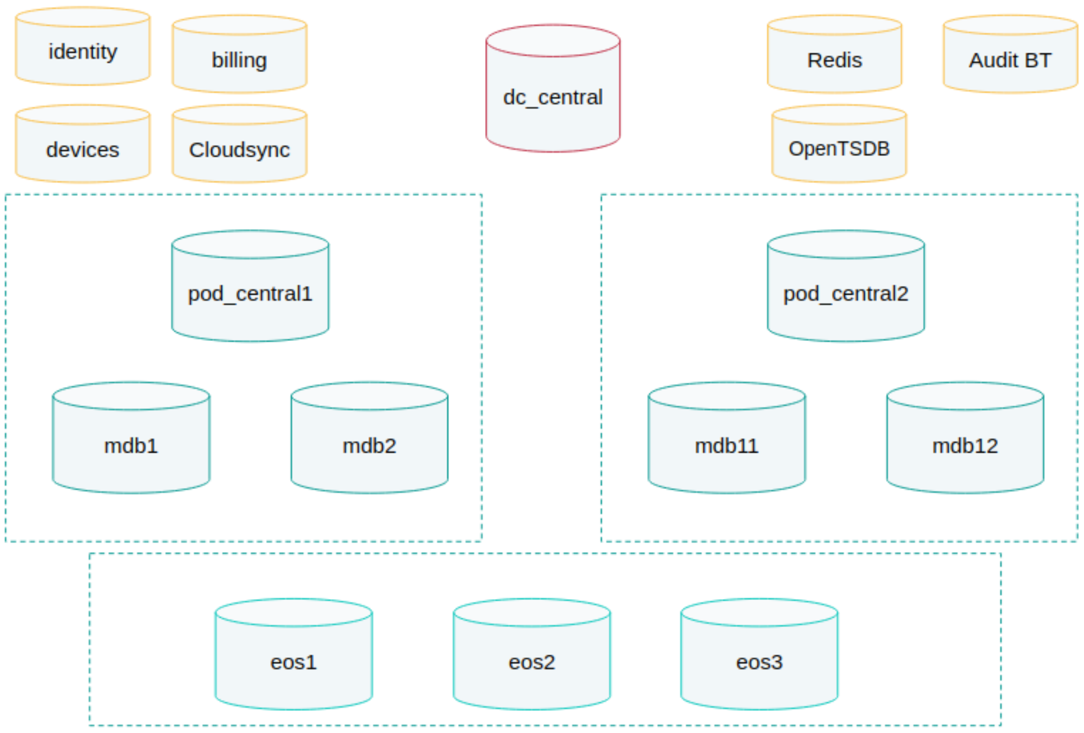
How are software and hardware upgrades rolled

out?

Most of the nodes are stateless and stateful component has an active-active failover. Upgrades are handled by taking the node out of the pool and upgrading and putting it back in the pool. We use jenkins+Ansible+puppet and custom automation for it.

How do you handle major changes in database schemas on upgrades?

Different services use different types of databases and they are upgraded in a different manner. At a 10000 ft they look like below screenshot :



1. EOS DB stores object metadata and grows very fast,

it’s sharded and we keep adding more of these.

2. MDB grows even faster, it’s sharded and we keep adding more of these.
3. DC_central is a DNS database and remains fairly static. We run many replicas of this for scalability.
4. Pod_central has fast mutating data but does not grow beyond 20M rows per table. We run many replicas of this for scalability.

- Every database schema is always forward and backward compatible i.e. we never drop columns and code in the same release, we first deploy the code in release-1 that stops using the column and in release-2 we drop the column.
- The non-sharded Databases gets upgraded as often as every week. They are the ones storing all kinds of feature-driven tables. We currently upgrade them using a script in production but we use Liquibase in QA and this is gradually moving to production
- Sharded DB new column alter happens using an automated script
- Sharded DB migration is a pain as we have 12000+ shards and growing, you can't do it in the 1-hour upgrade window. The way to do is:

- Live Code migrates the row as they need it. This means migration can happen over the months.
- Migrate using feature flags, you have both old/new code live at the same time and you migrate customer in the background and then flip a flag to switch them to go to new code path without downtime, more on this is here and here
- When we migrated from Lucene to ElasticSearch we had no option than to reindex all the content and we did it using feature flags and it took some 3-4 months to finish.
- Schema consistency checker reports ensure that all schemas are the same in all data centers after the upgrade.

Do you have a separate operations team managing your website?

Yes, we have a dedicated Production engineering, SRE and an IT/Ops team responsible for monitoring and managing the production. But as I said before Engineers who built the feature are responsible for making the decisions so they are deeply involved in monitoring the metrics and resolving production issues.

Miscellaneous

Who do you admire?

AWS: Their pace of innovation is admiring.

Google: Their tools like BigQuery, Kubernetes are awesome.

Elasticsearch: The rest API simplicity and architecture is awesome. We manage a 100+node cluster with terabytes of data and just 1 engineer.

MySQL/Memcache: they are simple, fast and awesome.

Eclipse/Jenkins: The plugin architecture is nice.

Have you patterned your company/approach on someone else?

We are a regular reader of http://highscalability.com/ , many designs are inspired by it.

- The POD architecture was inspired by Cell architecture at Tumblr. It is not an exact match but the concept of isolating failures is the same.
- The architecture to have jitter in Memcached and flush keys after 12 hours was inspired by facebook.

- Adding fingerprint to each database query by inspired by some article at http://highscalability.com/

  We are hiring, check us out at Jobs Page and contact us at jobs@egnyte.com if you are interested in being a part of our amazing team at Egnyte.