Experience with Elasticsearch scalability in AWS

BÉLA BOROS, EPAM SYSTEMS, SZEGED

15-JUNE-2016

AGENDA

1. About
2. Goals
3. Achievements
4. Obstacles solved
5. Lessons learned
6. What we liked

ABOUT CLIENT

- Working with global weather data
- Historical / forecast weather data
- Needs a scalable platform for geo spatial queries

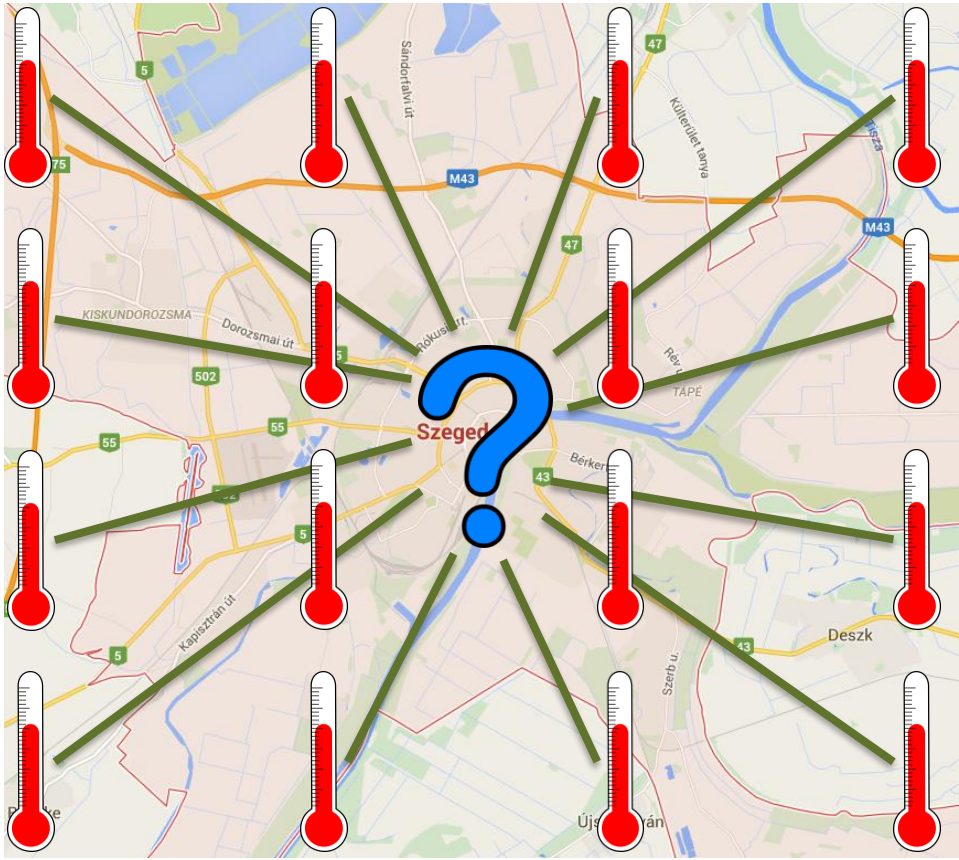
USE CASE: INTERPOLATED SPOT WEATHER

Goal: interpolated spot weather service

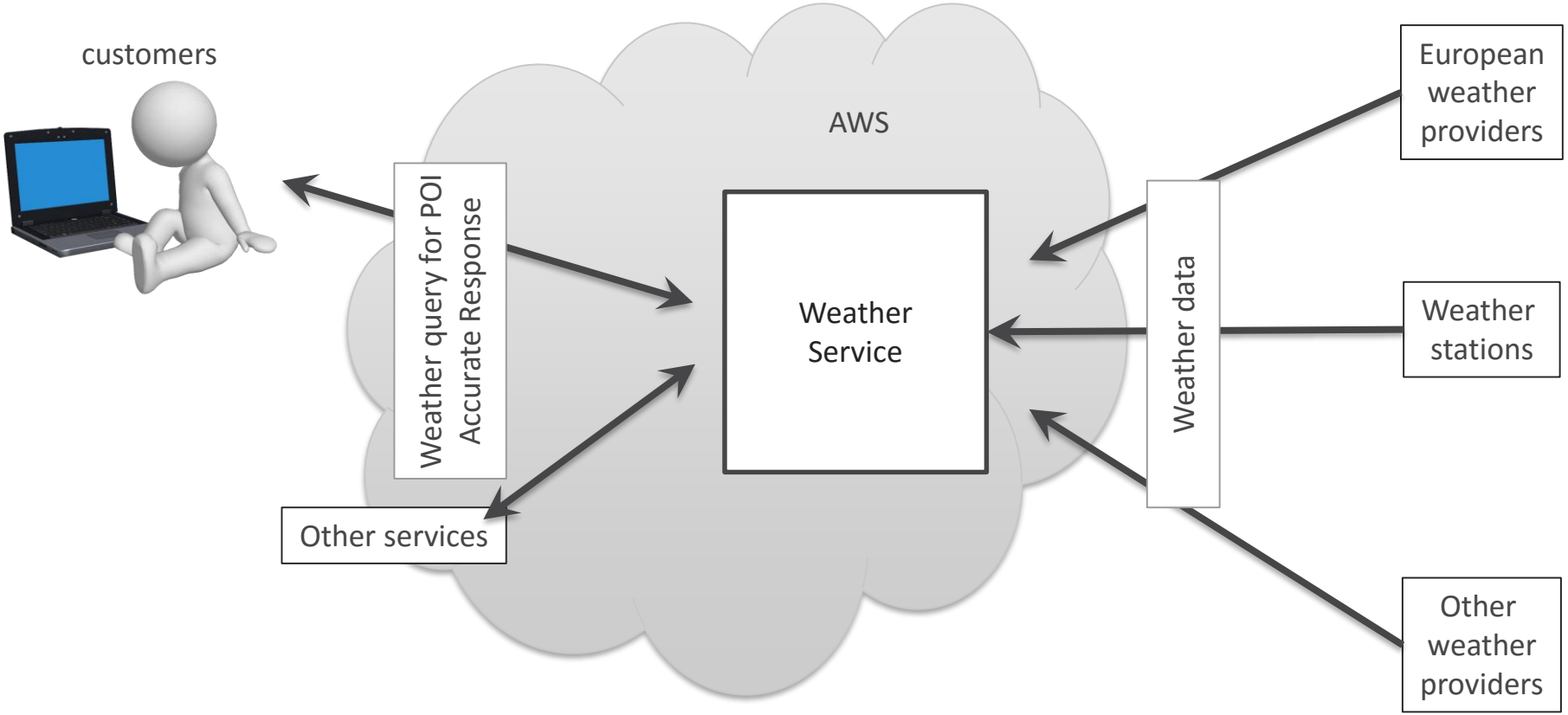
Return interpolated weather of N closest locations for a location for an altitude for a given time.

- Store 500GB daily input weather data
- Interpolate temperature
- Multiple calculations to combine input sources in next version

Time series, region based and other use cases in future.



INPUTS AND OUTPUTS



GOALS I.

Expandable & fast storage for micro-services, strategical platform for several future projects

- **Geo spatial search**
- **Elastic/linear-scalability (scale out/down)**
- **(Arbitrary) large data sets.**
  Current need:
  - ~500 GB daily
  - 3.3 million locations
  - 100 time forecast steps
  - 10 altitudes
  - Number of documents:
    - Primary goal: 1.5billion
    - Secondary goal: 10 billion

GOALS II.

- **Fast**
  - Ingestion
  - Queries: high throughput, low latency
- ingestion time:
  - Primary goal: 30 minutes (for 1.5 billion documents)
  - Secondary goal: 75 minutes (for 10 billion documents)
- Query speed:
  - Response time < 200 milliSec
  - Throughput 2000 req/sec
- Parallel ingestion & queries
- **Amazon AWS**
- **Cost efficiency**

ALTERNATIVES

Expectations:

- Expandable
- Low-latency random access
- Highly concurrent access
- Spherical geo spatial search

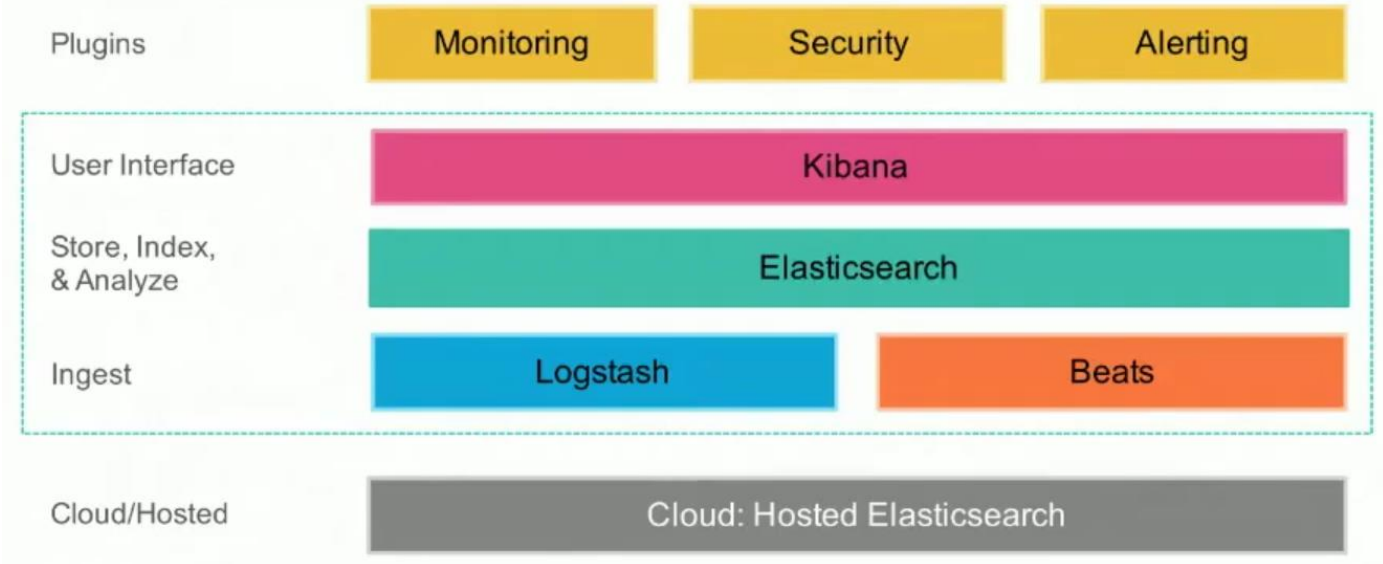
Customer insisted on Elasticsearch

Alternatives

Amazon DynamoDB

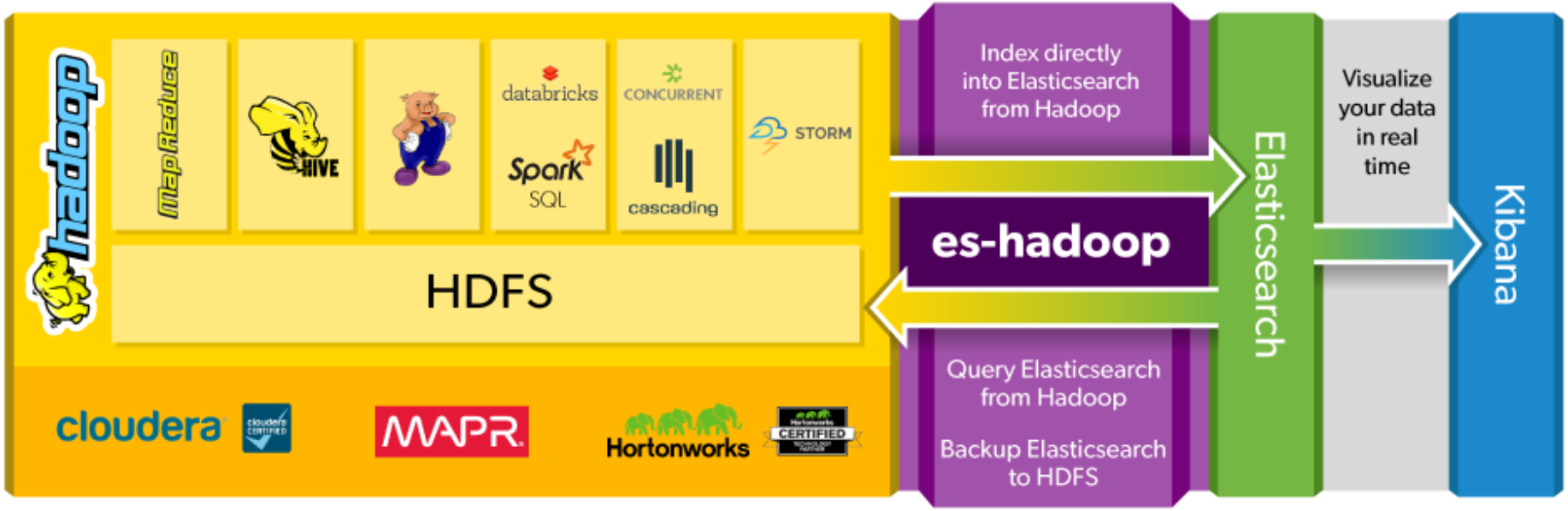
ABOUT ELASTICSEARCH

- Search engine based on Lucene
- Distributed, scalable, highly available
- Near real-time indexing and search
- Nice REST API



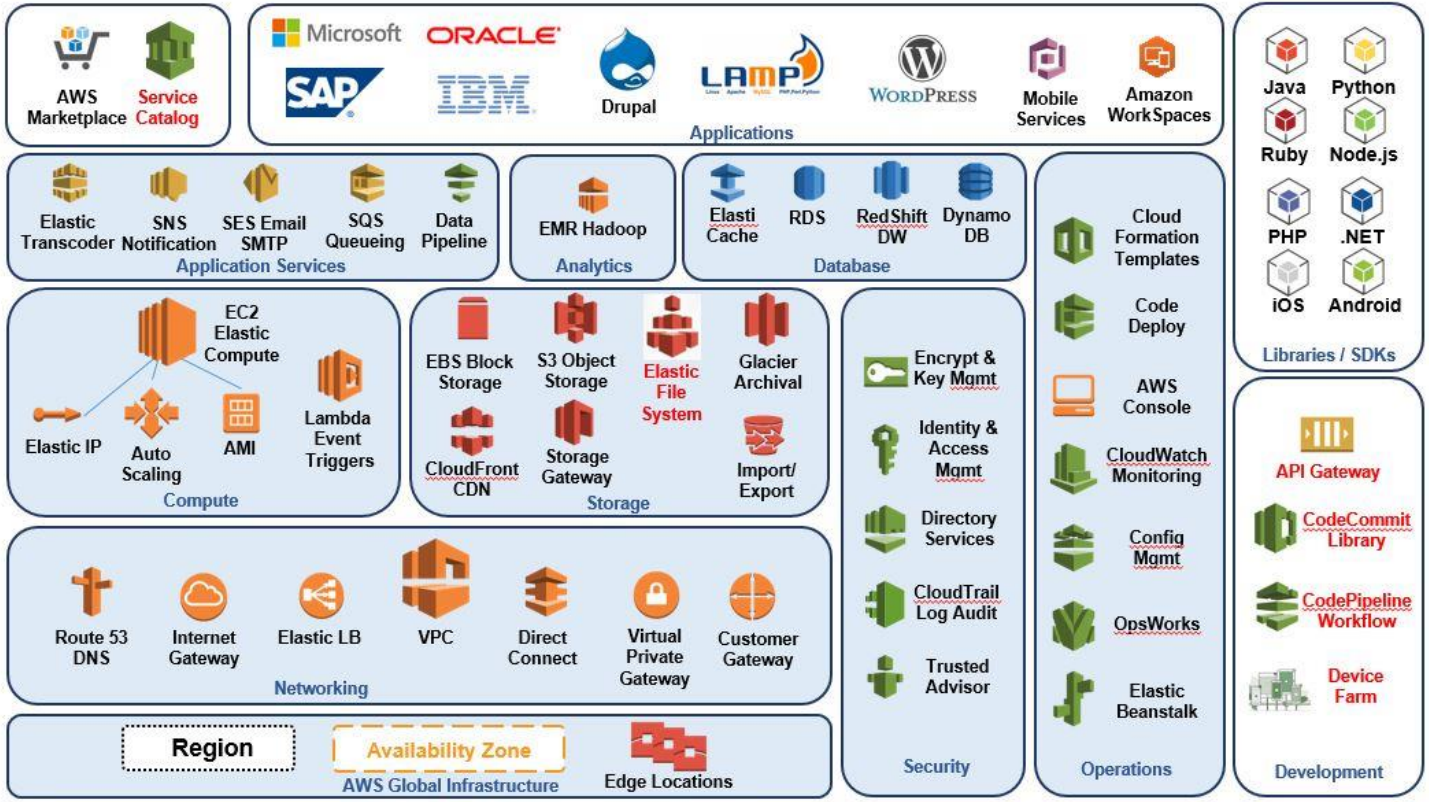
INTEGRATION

- Hadoop
- Spark / Spark SQL



AWS: AMAZON WEB SERVICES

- Biggest cloud provider
- easy to scale on AWS



TRADE-OFFs

Elasticsearch service of AWS	Elastic Cloud	Custom Elasticsearch on EC2
• Number of nodes < 10 • Less customizable • Easy to operate • Limited instance types • Slow • Older version	• Elasticsearch on AWS by elastic.co • Easy to scale and upgrade • Latest version	• Any number of nodes • Fully configurable • Requires (some) ES knowledge • Any instance type • For faster cases

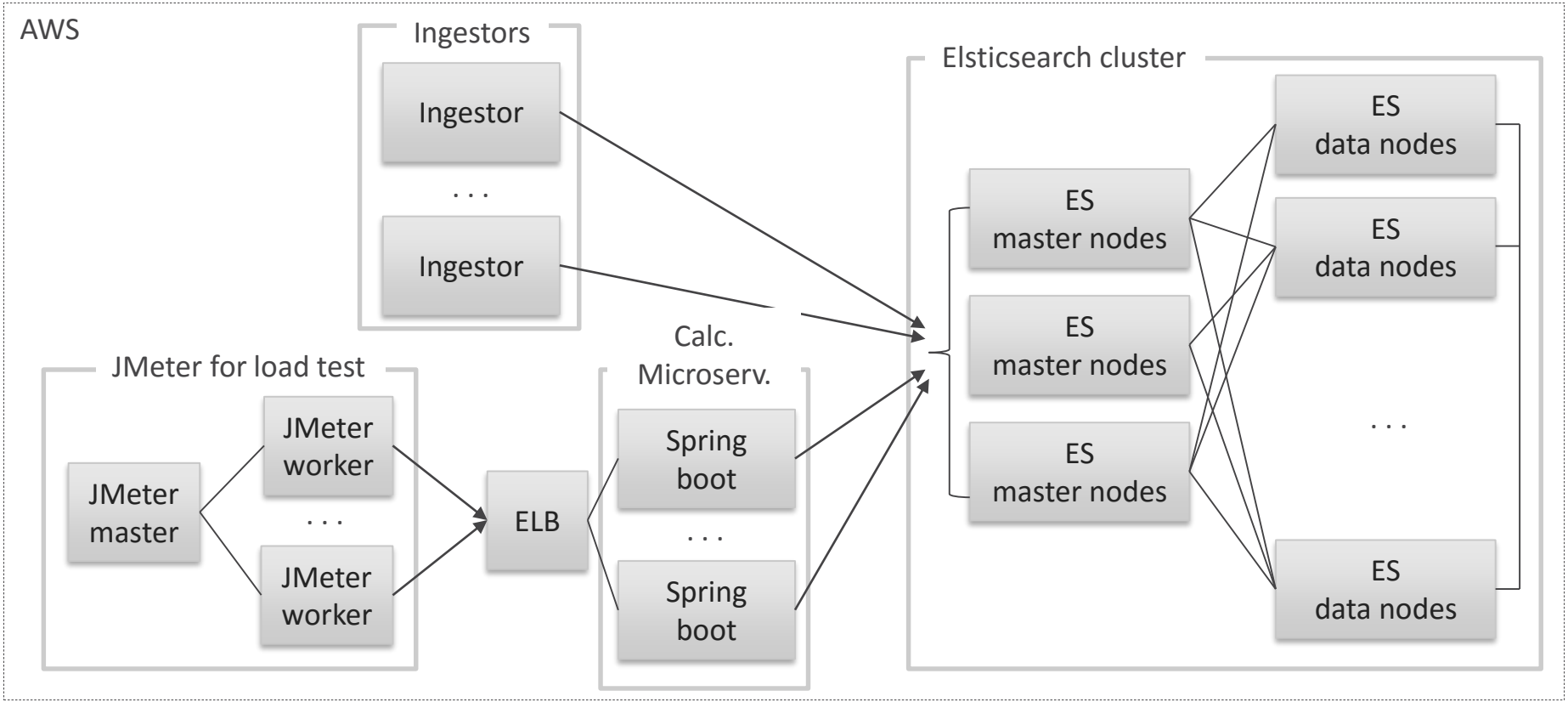
Shard by location		Shard by time
Hybrid sharding?		

How to organize documents: many small docs or few bigger ones?

Pre-calculate, cache or distribute storage and calculation?

ACHIEVEMENTS

ARCHITECTURE



SCALABLE INGESTION AND QUERIES

- “Full” control over performance
- Linear scalability
  - Store as many documents as we like
  - Ingest documents as fast as we like
  - Response time: as fast as we like
- Just add more nodes & shards
- Balanced parallel queries & ingestion
- Created a scalable/flexible/general distributed storage architecture that can be a strategic component for many current and future projects
- C3.4xlarge & locally-attached SSD
- ES with/without Docker

	10 nodes	30 nodes
Ingestion into SSD	• 185,823 docs/sec • 1 billion docs • 90 minutes	• 527,014 docs/sec • 1 billion docs, • 32 minutes
Ingestion into memory		• 797,576 docs/sec • 0.5 billion docs
Query	• 1,132 req/sec	• 3,239 req/sec

NEAR-LINEARLY SCALABLE INGESTION

$$C \approx \frac{Speed}{node \mid shard\ count}$$

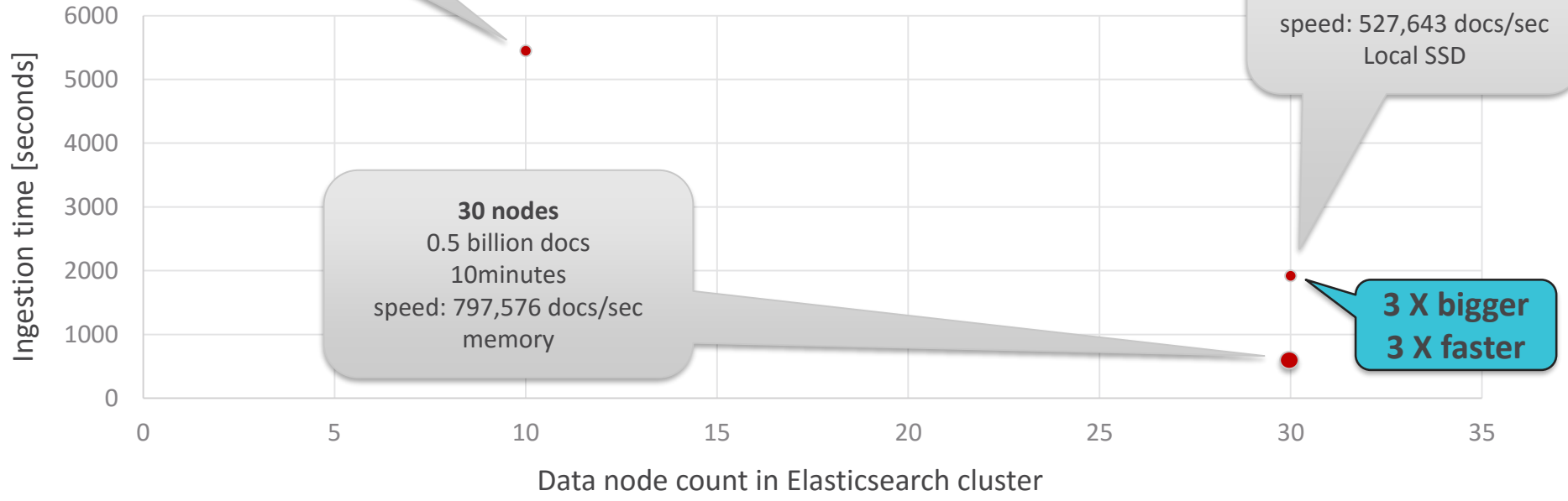
10 nodes
1billion docs
90minutes
speed: 185,823 docs/sec
Local SSD

Ingestion time decreases linearly with cluster size
(1 billion documents ingested)

30 nodes
1billion docs
32minutes
speed: 527,643 docs/sec
Local SSD

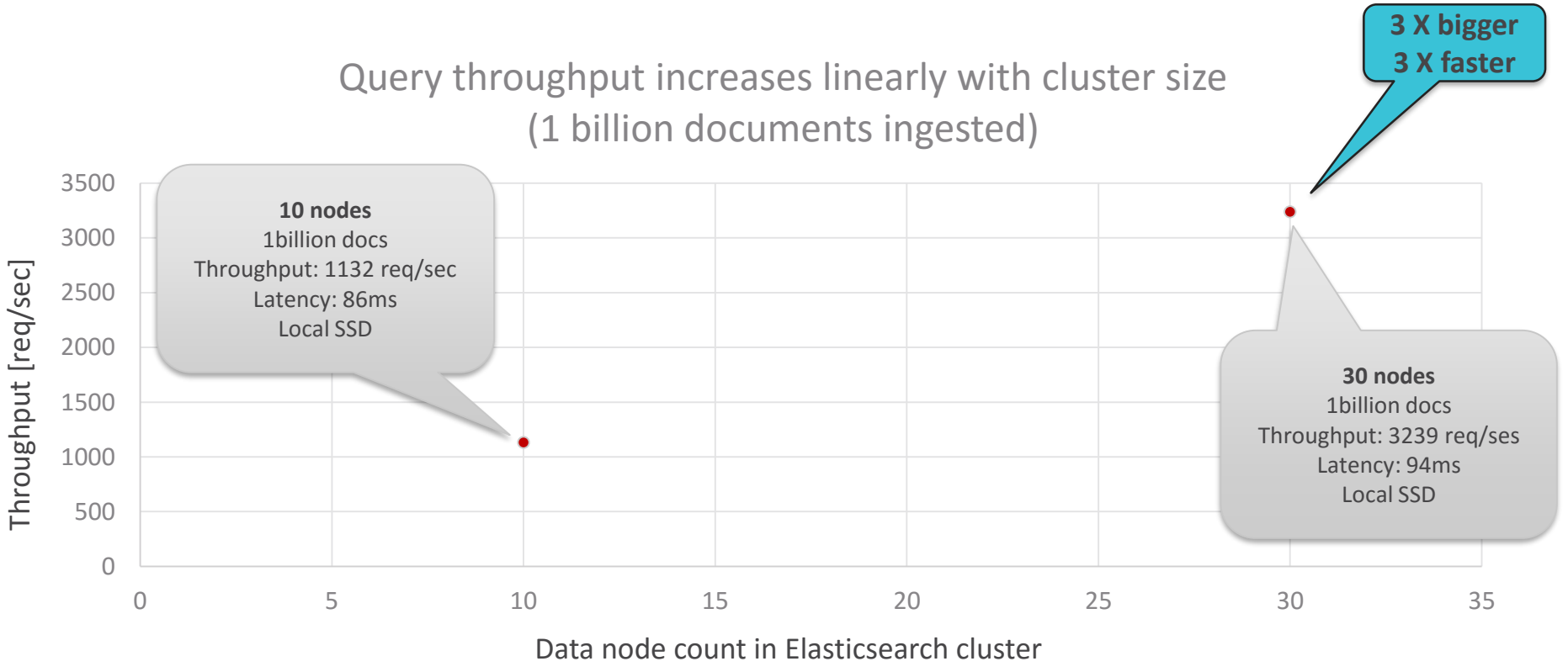
30 nodes
0.5 billion docs
10minutes
speed: 797,576 docs/sec
memory

3 X bigger
3 X faster



LINEARLY SCALABLE QUERIES

$$C \approx \frac{Throughput}{node \mid shard\ count}$$



PARALLEL INGESTION & QUERIES

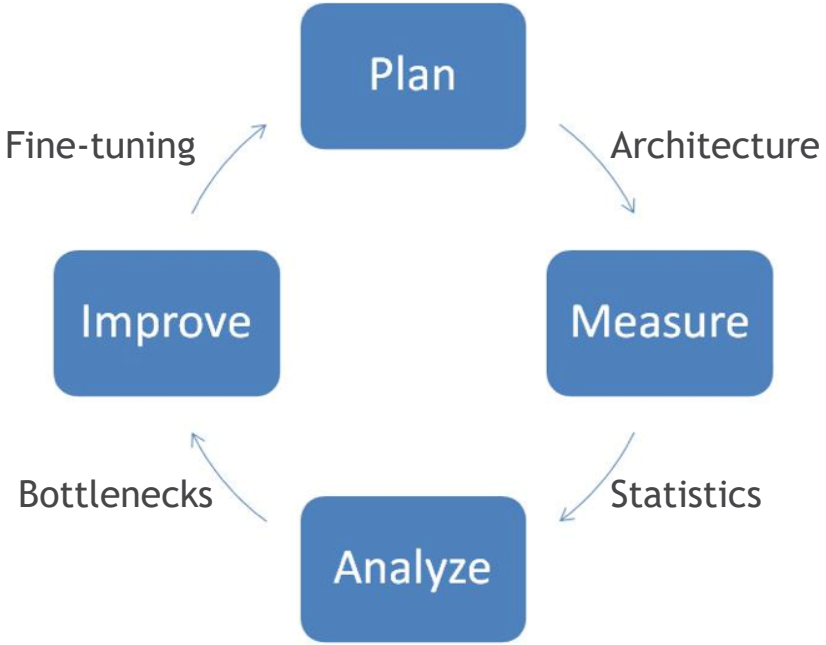
Ingest 500 million docs into the ES cluster with 30 (c3.4xlarge) data nodes,
while querying from 1 billion docs with 1 replica shard from another index.

Ingestion		Query		Load		
Elapsed time	speed [docs/sec]	Load	Avg query latency [milliSec]	ES data node CPU[%]	Disk IO read/write [MB/sec]	NET IN/OUT [Mbit/sec]
10 minutes 35 sec	797,576	0 req/sec	N/A			
12 minutes 8 sec	695,687	500 req/sec	76			
12 minutes 53 sec	655,188	2000 req/sec	96	1165	15/65	49/44

- Extra free capacity in 30-node ES cluster to server even higher query load
  or use a smaller cluster
- Tune free parameters to get optimal price/performance ratio (even dynamically for ingestion periods):
  - Node count
  - AWS VM instance type (CPU, memory, DISK size, EBS/SSD, throughput, latency)
  - Primary shard count, sharding type
  - Replica shard count
  - AWS instances on demand or reserved

OBSTACLES SOLVED

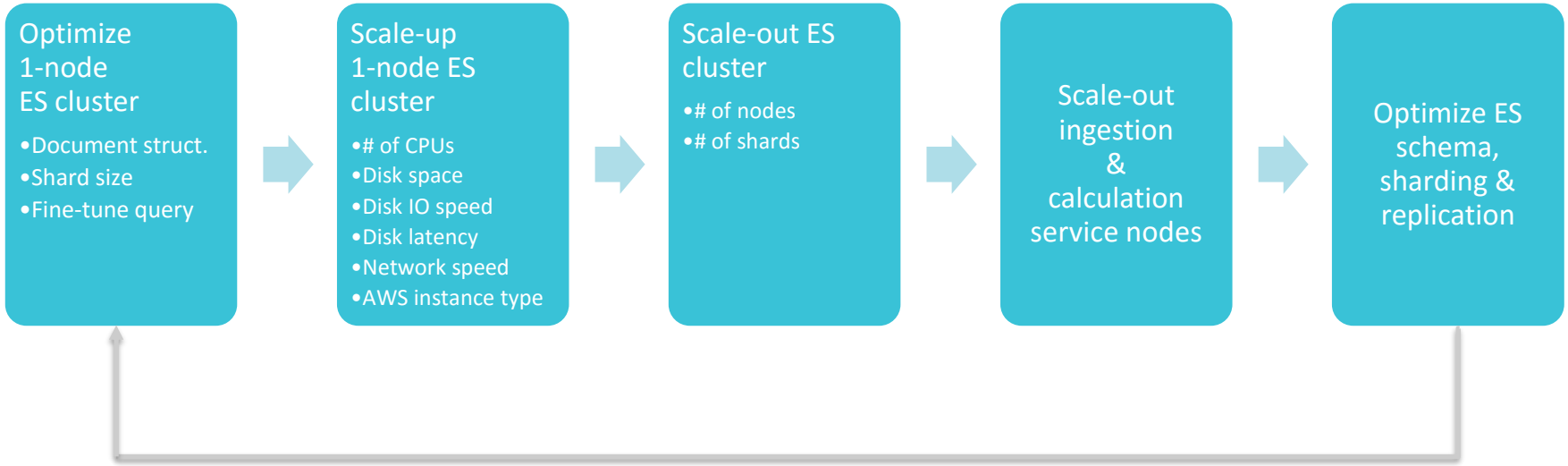
PERFORMANCE OPTIMIZATION PROCESS 1.



Innovate,
Forrest,
Innovate!

Based on measurements

PERFORMANCE OPTIMIZATION PROCESS 2.



MEASUREMENTS

- Raw system performance (in Docker container and in VM):
  - Disk IO throughput & latency (iostat, dd)
  - Network (nload)
  - CPU utilization (top)
  - memory
- ES performance:
  - Ingestion speed & time (ingestor app, time)
  - Query throughput & latency (JMeter)
  - Replica creation time (manually)
  - Cluster utilization (plugins: Marvel, HQ)
- Scenarios:
  - Ingestion only
  - Query only
  - Parallel ingestion and query of different indexes within the same cluster

INFRASTRUCTURE

- Distributed data ingestion on multiple machines & multiple threads
- Docker based virtualization
- Automated application deployment:
- Automated provisioning:
    - Dynamic & parallel infrastructure provisioning (AWS CloudFormation, Bash scripts)
    - Easy to scale Cluster configuration in CSV and some auxiliary files (all in a common directory)
    - Parallel application deployment (Bash scripts) using a deployment server in AWS

ES CLUSTER CONFIGURATIONS

- **AWS instance type scale-up**: m4.xlarge → c4.4xlarge → c3.4xlarge → i2.2xlarge
- **Cluster size scale-out**: 1 node → 7+3 → 10+3 nodes → 30+3 nodes
- (Remote) **EBS vs instance-store** SSD
- **Disk vs memory storage**

- **Document count**: 1M → 10M → 100M → 1B
- **Document schema**:
  - Small vs large document size: 10 vs 50 weather parameters
  - Mapping settings: `index: no`, `norms: disabled`, `dynamic: false`
- No upper limit for indexing (`throttle: none`)
- **# of ingestor instances**: 1 → 2 → 3 → 6
- **# of parallel threads**: 1 → 10 → 16 → 32
- Ingestor profiling with JVisualVM and AWS CloudWatch

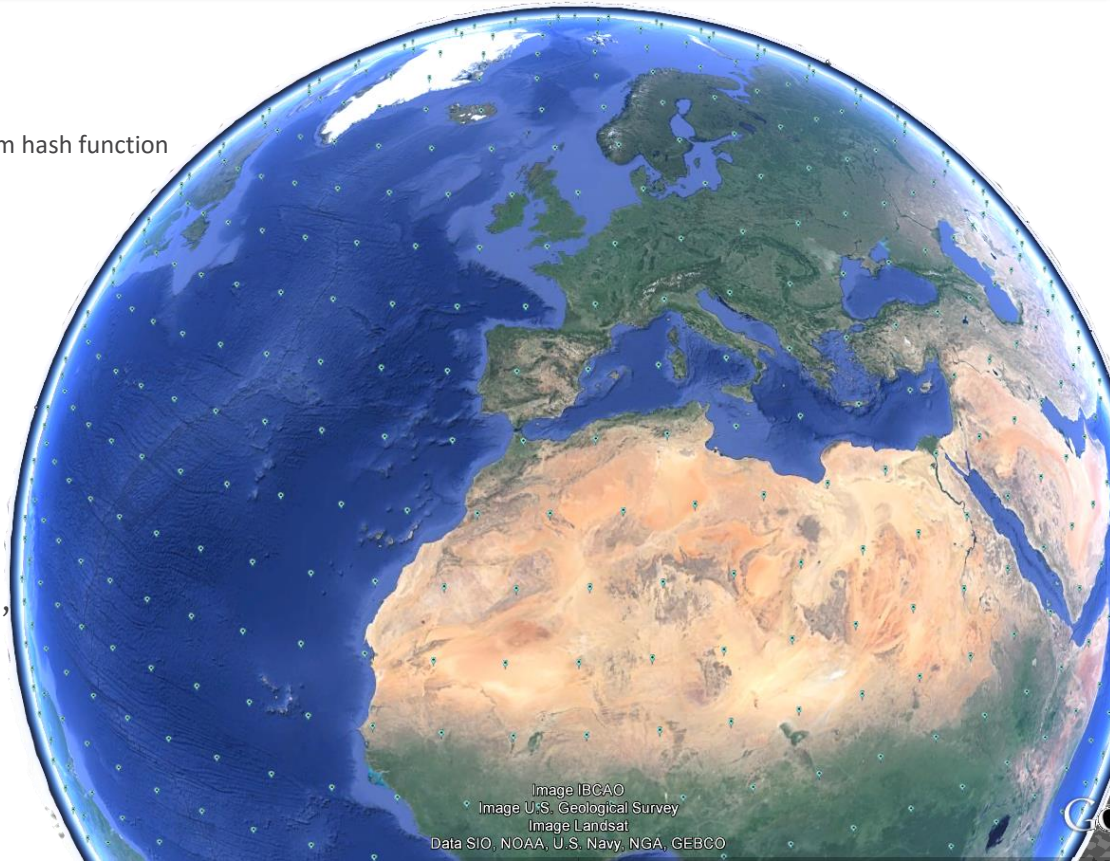
TECHNOLOGY STACK, TOOLS, PRINCIPLES

Technology stack	Tools	Methods, Principles
• Amazon EC2, EBS, ELB • ElasticSearch • Spring Boot • JMeter • Docker • CentOS Linux • TestNG, AssertJ, Mockito • Hystrix • Graphite/Graphana • ELK	• AWS CLI • Eclipse, IntelliJ • Maven • Git • Concourse CI, GoCD, Bamboo • Quay.io • Trello, Jira • Confluence • Zoom, Skype, HipChat, Slack • GIS tools (Google Earth) • sketchboard.me	• Agile, Kanban • Pair programming • Distributed teams • Test driven development • Infrastructure as code • Immutable infrastructure • Monitor, measure, improve, iterate

LESSONS LEARNED

OBSTACLES SOLVED & TECHNOLOGY DETAILS

- Balance cluster
  - Evenly distribute: locations, queries, shards, custom hash function
  - Remove hot shards, nodes, overloaded masters
  - Generated sample dataset
- Ingestion:
  - Multiple threads
  - Bulk ingestion API
  - Dedicated bulk for shards
- Find best sharding:
  - Geo-location-based
  - Time-based
  - Hybrid (model & time & altitude)
- Geo-spatial query speed up: geo-distance-sort, geo-distance, geo-bounding-box, geo-hash, bool-filters
- NodeClient vs TransportClient vs REST API



GENERAL

- Exact measurements driven development
- Extrapolate carefully!
- Ingestion should be optimized separately from query tuning
- Measure speed after each and every modification
    - Hard to realize in practice due to time pressure
- Measure the cumulative effect of multiple changes
    - Problematic when the number of options / combinations to try is large
- Long enough ingestion test with large enough data sets (but not too large) on a big enough cluster, but it should run fast enough ☺

AWS

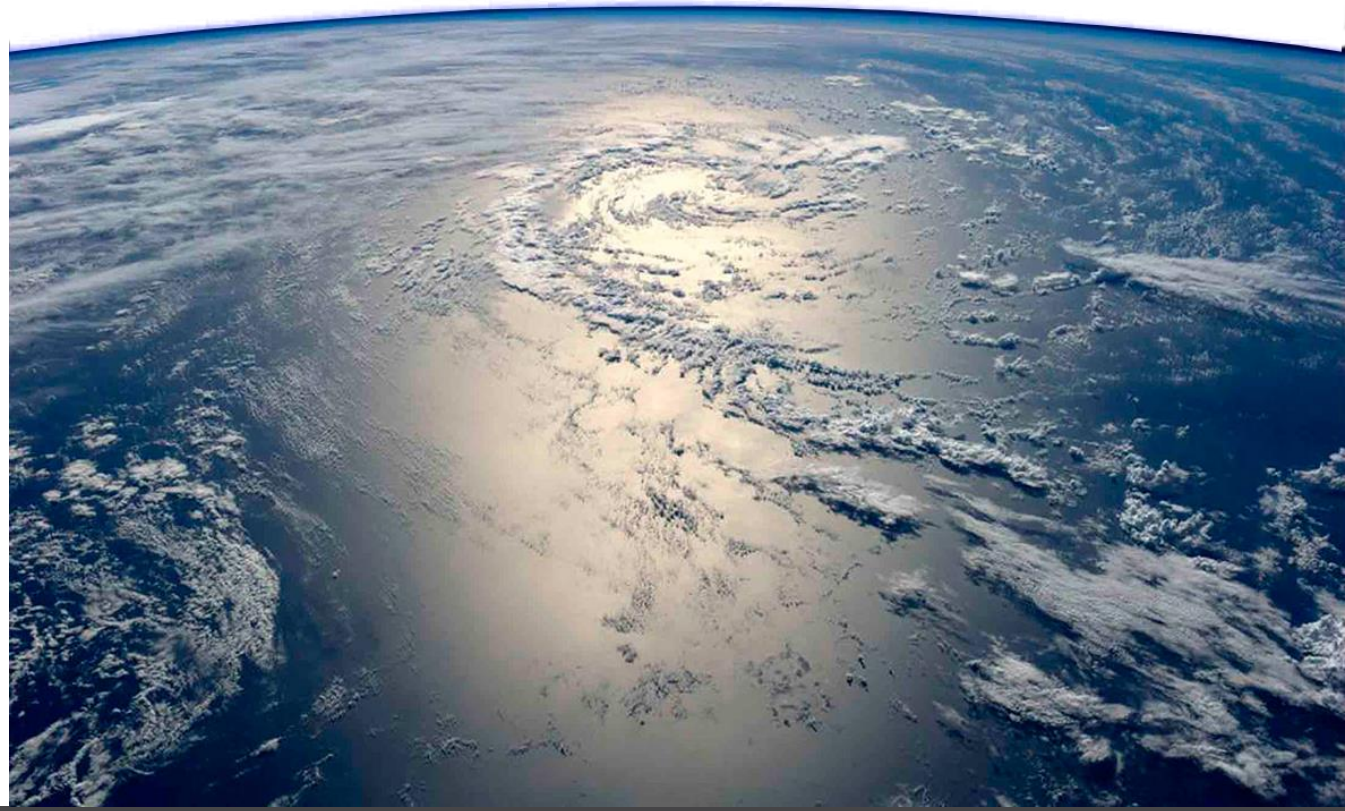
- EBS warmup
- High EBS latency → Significant performance impact on ES
- Ephemeral (local SSD) storage is much faster for random access
- Fluctuation instance-store SSD write latency with small files < 4096bytes

ELASTICSEARCH

- **More shards** → Better ES cluster utilization → Better scalability
- **Replica shards** (even with a single instance) → Higher query throughput during a parallel ingestion
- **Overloaded ES** becomes unreliable (due to internal timeouts and high disk latency)
  - Workaround: Catch exception, sleep, retry operation, abort after X attempts
  - Limit traffic to ES cluster in REST layer
- **Be cautious** with configuration tweaks; some may reduce performance!
- **Carefully with plugins:** Marvel monitoring plugin slows down the ingestion
- **Java client:**
  - 1-node cluster → Use TransportClient
  - Multi-node cluster → Use NodeClient
    (May not work for external clients connecting to an ES cluster behind a firewall!)
- **Docker overhead: less than** 5%
- **Bottleneck:**
  - Uneven ingestion speed (some threads finishing much earlier than the rest) → Lower throughput
  - Not enough time-based shards → Weaker scalability
  - Too large shards → High latency

WHAT WE LIKED

- Good documentation
- Excellent examples
- Frequent releases
- Large community / forum
- Easy to scale in cloud



THANK YOU!

THE END