Lessons from an AWS Migration

Chris Mills
The Meet Group
The Short Version

- Migration will cost more and take longer than planned.
- Production operation may be cheaper. It may not.
- Get it working first, then scale up and optimize iteratively.
- Instances fail all the time. Plan for it.
- If dynamism was a key goal, you’ll be happy.
Story Time
Once Upon a Time...

- Years ago, we had a basic event pipeline.
  - We logged events in plain text strings.
  - ETLs parsed the strings and inserted rows to GreenPlum.
  - Daily GP batch jobs inserted metrics to an Oracle warehouse.
  - New events required new logging code, ETLs, SQL queries, and dashboards.

- It met minimum requirements, but...
  - Key components were reaching EOL, and scale was exceeding capacity.
  - Event structure and contents varied, enforced only by tribal convention.
  - ETL maintenance was a nightmare, and got worse with every change.
  - Adding events involved coordinating multiple teams and weeks of delay.
Logging Shouldn’t Be Hard

▪ **GOAL: Encourage event logging**
  - Storage is cheap. Not knowing what happened can be (very) expensive.
  - Powerful, mature open source tools for distributed processing are available.
  - We don’t always know what’s important up front. Err on the side of more logs.
  - A PM and an engineer should be able to log something new in about an hour.

▪ **GOAL: Keep complexity under control**
  - Standardize event serialization.
  - One bus for all events, organized by subject.
  - Set processing options in subject-level metadata.
  - Automate processing to rely on schemas+subject metadata+events.
Let the Users Log

- **Self service event processing (Ramblas)**
  - Avro serialized events with schemas extending a common skeleton
  - A schema repository that enforces back-compatibility for subjects (TASR)
  - Metadata in TASR controls event processing at subject level
  - Libs, plugins and sample code for engineers adding event logging
  - A stack (Kafka, HDFS, Hive, HBase) that can handle the scale

- **Goals achieved**
  - A PM and an engineer can add new subjects in about an hour.
  - Newly added events are consumable from Kafka right away.
  - Aggregates for new events automatically generated in next batch run.
Ramblas in the Data Center

Diagram showing the integration of various technologies including Apache, PHP, Bruce, Kafka, ZooKeeper, Camus, TASR, Hadoop Cluster, Spark, Hive, HBase, HoneyBadger, BottleFly, FlyTrap, and Airflow.
So, why leave the DC?

- **The data center is stable**
  - >1 TB/day raw event volume across >200 subjects
  - 30 node cluster (~2.5PB storage, ~6TB mem, ~1050 containers)
  - 5-7 hour daily batch runs met our SLAs
  - Cost efficient compared to previous system

- **But, it’s not dynamic**
  - A failed batch plus a rerun with a degraded cluster can take 18 hours
  - Significant backfill with new models isn’t really feasible
  - No way to “power up” temporarily for infrequent deep dives
  - Too little headroom for experimentation chills new development
Migration Goals

- Do what the DC already does as well or better.
- Enable experimentation & backfill in parallel with low cost & risk.
- Keep cost & complexity the same or lower for the same outputs.
Why AWS?

- Dynamic provisioning
  - Batch cluster scaled and spun up as needed with spot instances
  - Spin up instances for experiments, then spin them down
  - Deep dives get dedicated, temporary clusters, sized to fit
  - Backfills get dedicated, temporary clusters, sized to fit
  - The ephemeral tasks don’t steal resources and cripple the core pipeline

- Existing ecosystem
  - Extensive service offerings
  - Good tools, libraries and examples for working with AWS were easily available
  - Many of our engineering candidates have some experience with AWS already

- Subsidized migration support
Ramblas in AWS
Key Differences

- Multiple Clusters
- Kafka Mirroring
- S3 and Glacier mostly replace HDFS (even for HBase)
- User facing applications containerized with ECS
- RedShift for ad hocs and historical SQL queries
The Good

▪ RedShift
  - We had more than a thousand legacy SQL queries.
  - We needed ~20% to work for key BI metrics.
  - Better than 90% of them worked with minor changes.
  - PostgreSQL drivers worked out of the box for our ETLs.

▪ S3 and Glacier
  - Lifecycle rules eliminated the need for archiving ETLs and DAGs
  - Parallel installations possible with separate buckets, making dev easier
  - Console tools, boto3 and compatibility with distcp made data migration easier
  - Storing HBase data directly in S3 made the persistent cluster health less critical
The Really Good

▪ Dynamic Provisioning
  - Adding or changing hardware took weeks. EC2 changes take minutes.
  - Everything is scriptable. Console tools and boto3 are solid.
  - Using spot instances for temporary needs routinely cuts cost by half.

▪ Disposable Clusters
  - EMR makes spinning up a cluster fast, easy, and scriptable.
  - Many DC maintenance issues disappear with an ephemeral cluster.
  - Deep dive results in hours to days, not weeks.
  - Less debate on experiments. When it’s cheap and quick, just run them.
The Bad

▪ Instability / Availability
  - Spot instances get outbid routinely.
  - EC2 instances just fail.
  - ECS containers die all the time.
  - Sometimes there just aren’t nodes available.
  - Monitoring and automated repair is much more important.

▪ Mistakes Can Be Costly
  - Zombie clusters and a weekend can run up a serious bill.
  - Review S3 lifecycle rules regularly, especially during a data migration.
The Ugly

- **Kafka**
  - For us, Kafka was faster and cheaper than using Kinesis.
  - Kafka on EC2 worked best for us with brokers in the same AZ.
  - You want to use instances with their own disks (HDD or SSD).
  - Handling instance failures well required reconfiguration and ordered restarts.
  - Ansible worked better than Puppet or Terraform handling broker instance failure.

- **MirrorMaker**
  - You can bridge, consuming from 0.8 and producing to 0.10, with MirrorMaker.
  - Run the MirrorMaker distributed with Kafka 0.8 to produce to your 0.10 brokers.
More Ugliness

- ECS
  - Containers fail way more often than EC2 instances.
  - Running your scheduler in a container was an error. We don’t anymore.
  - Moving applications to containers is a significant project on its own.

- Tez
  - AWS is phasing out support for MapReduce, so you should use Tez.
  - Tez is definitely faster than MapReduce for our pipeline.
  - Previously stable Hive jobs with UDFs & UDAFs began running out of memory.
  - Debugging on EMR is more difficult than on our DC cluster.
The Really Ugly

- **HBase on S3**
  - Addresses the problem of “What happens to my HBase data if the cluster fails?”
  - Great fit for the “large historical data sets” use case.
  - Still has some bugs. Which can cause region overlaps and corrupt store files.
  - Snapshots are not enough protection.
  - Storing a full copy of the HBase root in a known stable state is a good idea.

- **Data Migration**
  - Make directory structure changes in HDFS *before* going to a Snowball or S3.
  - Once in S3, modifications are made to each object individually. It’s really slow.
Lessons Learned

- Instances fail all the time. Plan for it.
- Day to day costs depend on timing.
- Get it working first, *then* optimize and lock it down.
- Containerization is a migration in and of itself.
- Experimentation will increase, as will new apps, costs, and value.
Thanks

- Learn from our mistakes (and successes)
  - Ramblas is being open sourced. Check out github.com/ifwe
  - Bruce and TASR are available now.
  - AWS deployment scripts and pipeline code in October in the Ramblas repo.

- Tagged, if(we), and The Meet Group
  - The project wouldn’t exist without Tagged. Thanks, Johann.
  - The migration wouldn’t have happened without if(we). Thanks, Gene.
  - Today the project continues, courtesy of The Meet Group. Thanks, Nik & Rich.
  - Thanks to my team. Without you, there wouldn’t be a project to talk about.

- Chris Mills
  - cmills@themeetgroup.com