SILICON VALLEY

In-Memory Computing | SUMMIT 2017

AN IGNITE COMPUTE GRID IN THE CLOUD

CHRIS BERRY

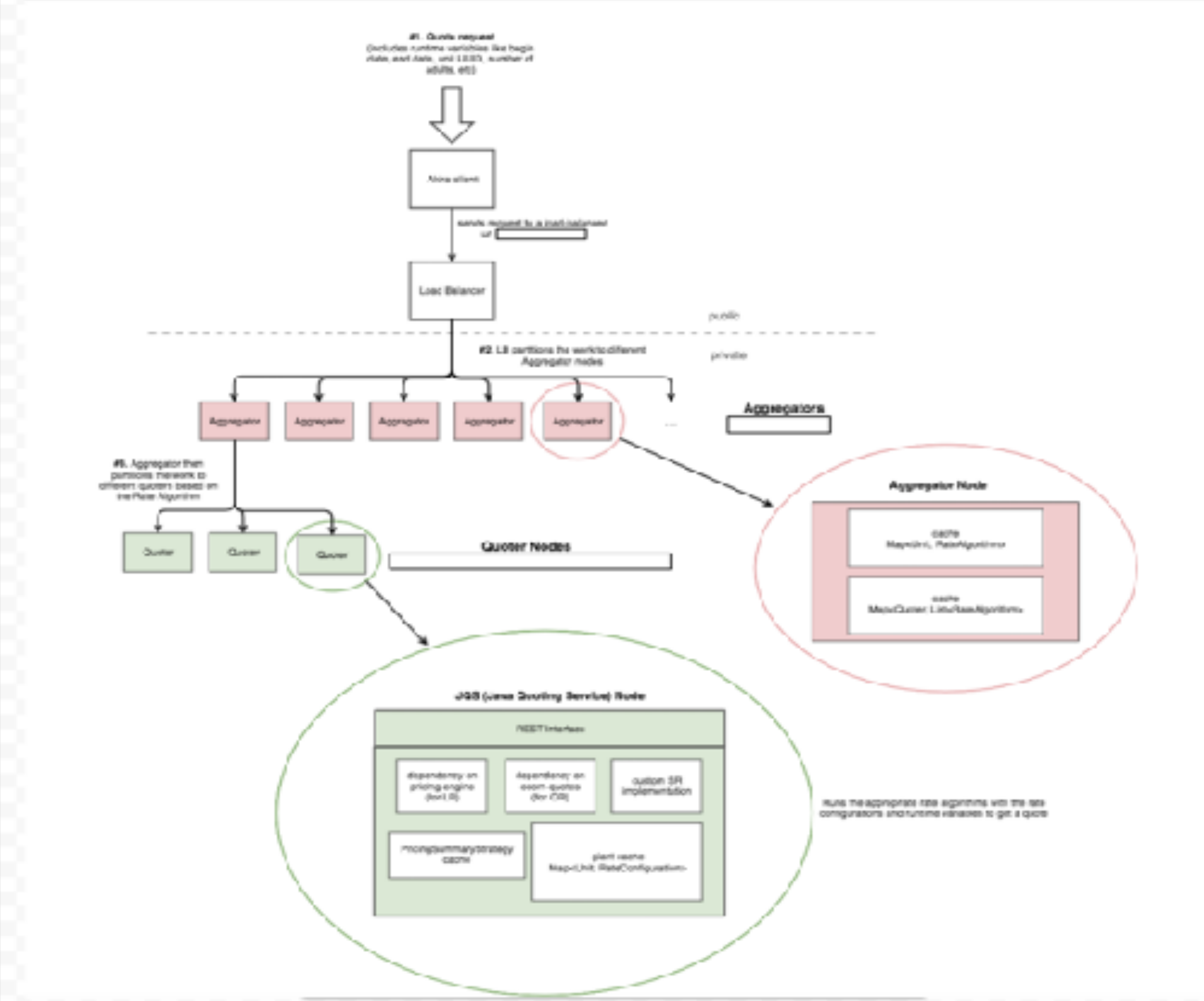
SO. WHAT'S THE APPLICATION?

- "Akira" -- the "Rates Engine" for HomeAway
  - Computes quotes based on cached "Rate Data"
- High volume/low latency application
  - Average: 12B quotes per day @ ~40ms p95
  - Computed in batches of ~200 avg. (~2300batches/ sec)
  - Over ½ TB of Rate Data

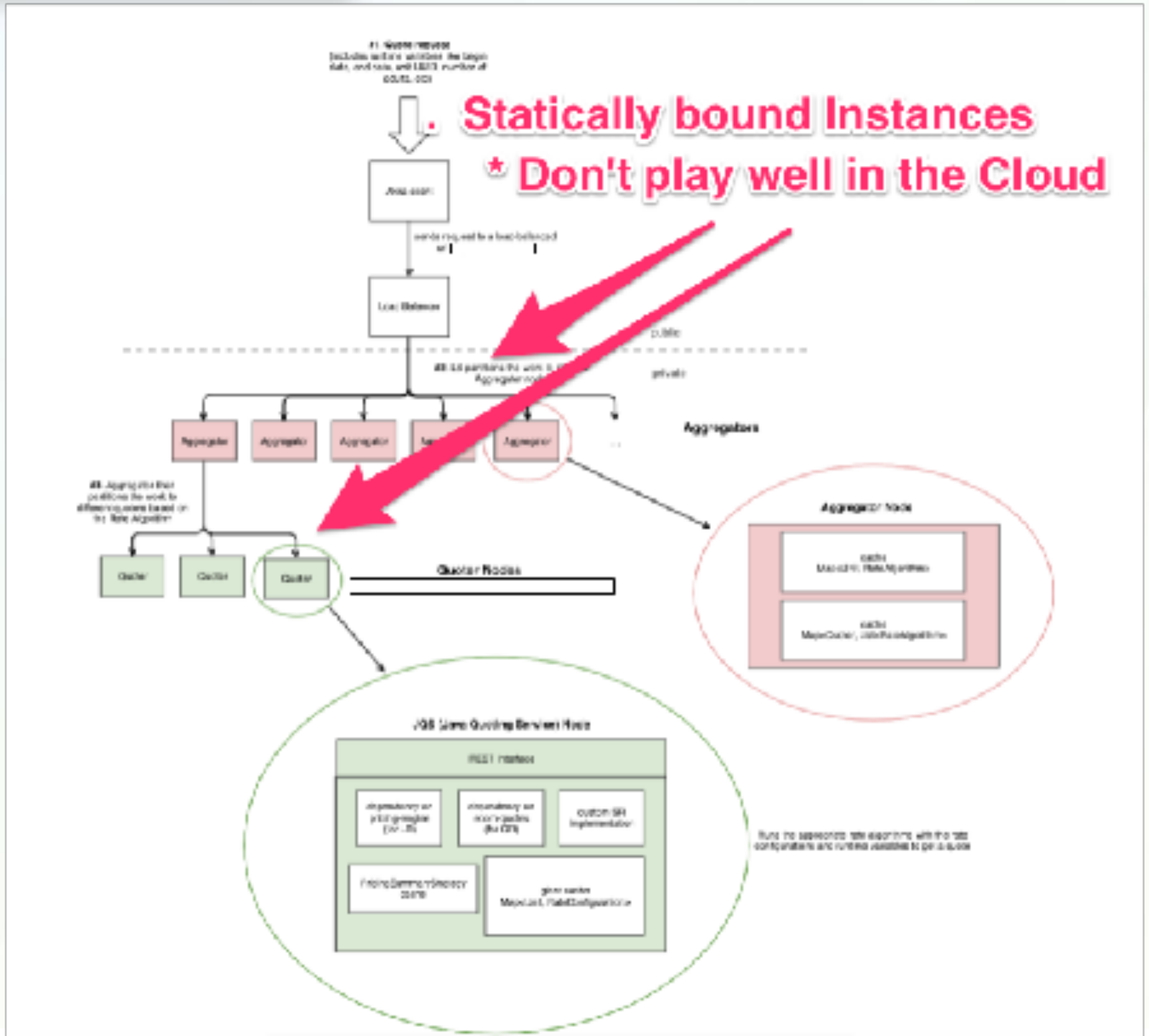
AND WHAT'S THE JOB?

- Migrate from old-school, terrestrial deployment to the Cloud.
- Simplify whatever we can in the process.
- Modernize the stack while we're at it.
- And don’t break things!

THE OLD-SCHOOL, TERRESTRIAL AKIRA.


#1. Quote request
(includes runtime variables like begin date, end date, unit ID, number of adults, etc)
Akira client
sends request to a load-balanced url
Load Balancer
public
private
#2. LB partitions the work to different Aggregator nodes
Aggregator
Aggregator
Aggregator
Aggregator
Aggregator
...
Aggregators
#3. Aggregator then partitions the work to different quoters based on the Rate Algorithm
Quoter
Quoter
Quoter
Quoter Nodes
Aggregator Node
cache
Map<Unit, RateAlgorithm>
cache
Map<Quoter, List<RateAlgorithm>>
JQS (Java Quoting Service) Node
REST Interface
dependency on pricing engine (for LR)
dependency on room quotes (for OR)
custom SR implementation
Pricing/summary/strategy cache
plan cache
Map<Unit, RateConfiguration>
Runs the appropriate rate algorithms with the rate configurations and runtime variables to get a quote

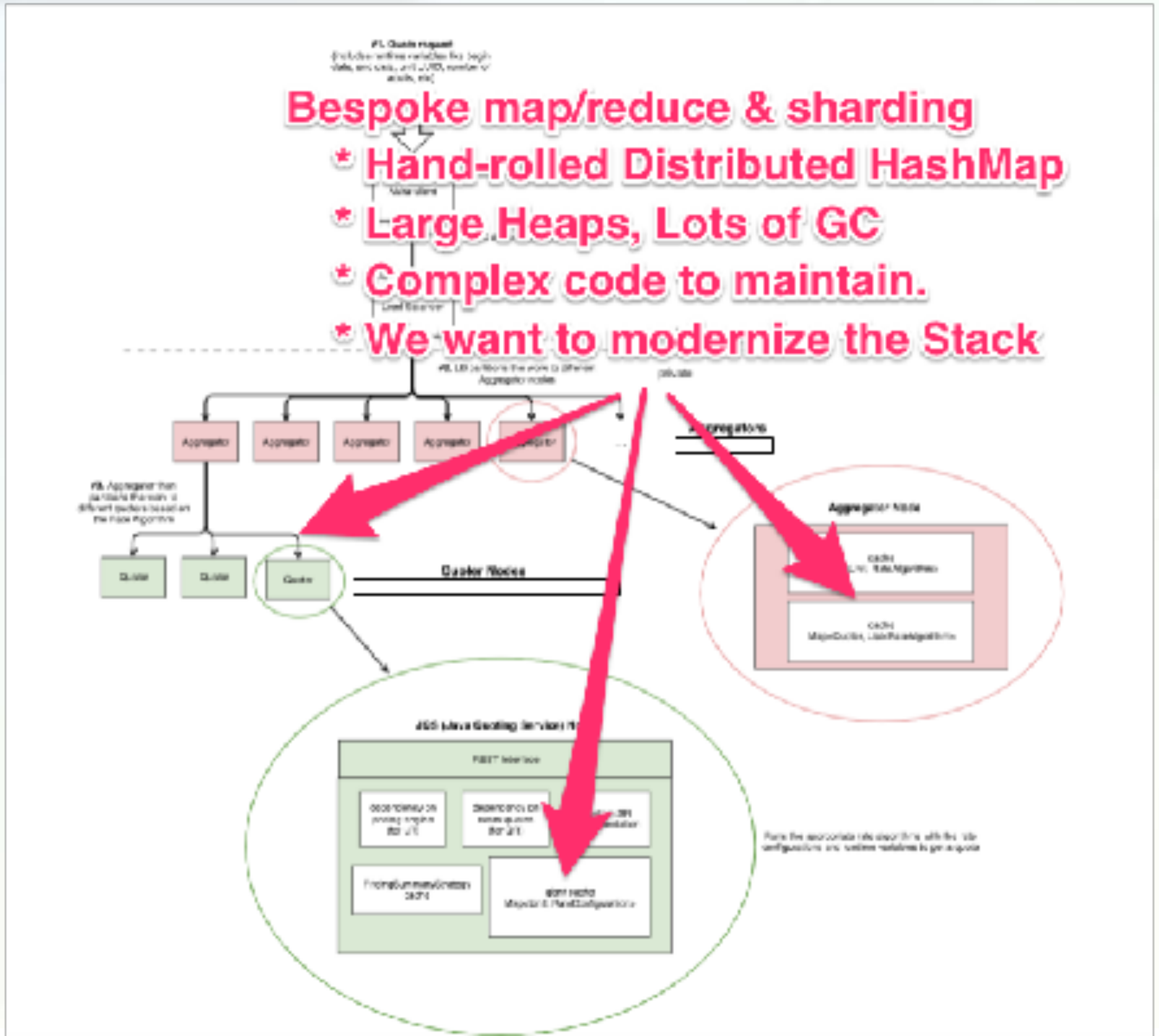
SO. WHY NOT PORT IT TO THE CLOUD AS IS?



Statically bound Instances

* Don't play well in the Cloud

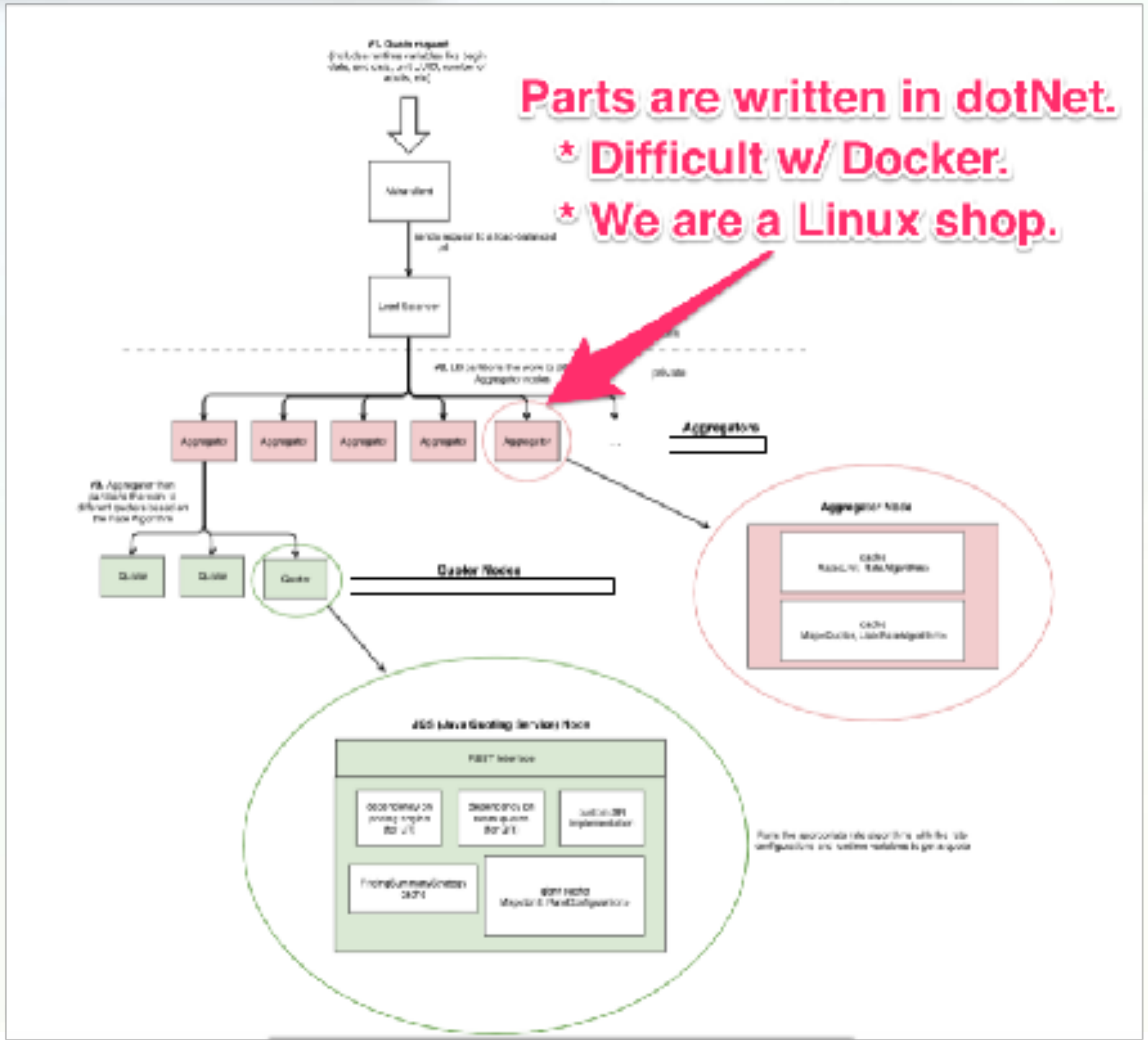
SO. WHY NOT PORT IT TO THE CLOUD AS IS?



Bespoke map/reduce & sharding

- * Hand-rolled Distributed HashMap
- * Large Heaps, Lots of GC
- * Complex code to maintain.
- * We want to modernize the Stack

SO. WHY NOT PORT IT TO THE CLOUD AS IS?



Parts are written in dotNet.

* Difficult w/ Docker.

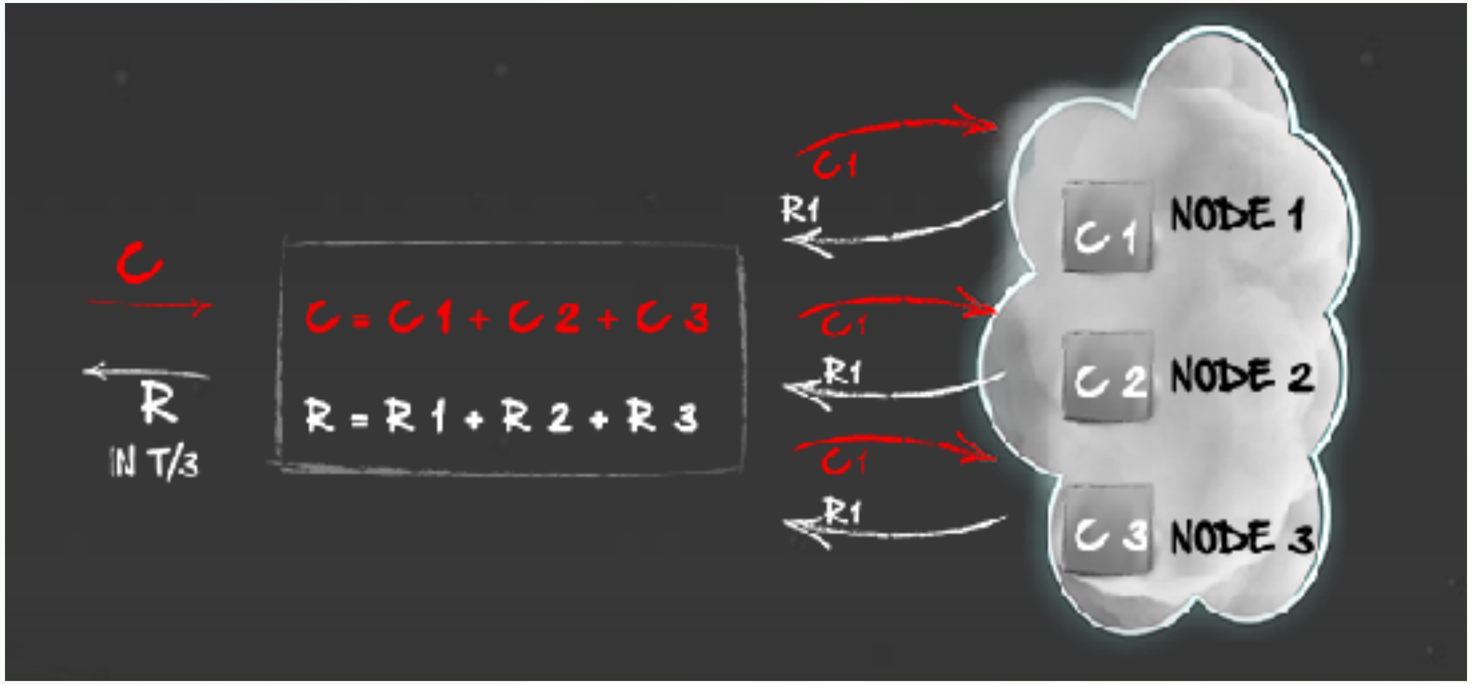
* We are a Linux shop.

OK. WHY NOT USE REDIS / MEMCACHED /... ?

- Those pesky Laws of Physics intervene.
- The Rate Data to compute a quote is large (~0.25MB/unit)
  - 0.25M * 200/batch over 1Gbit network ~= 500ms
  - Before you even start computing!

SO. HOW DO WE SOLVE THIS?
IN-MEMORY COMPUTING. OF COURSE.

- The Apache Ignite Compute Grid
  - Provides a simple map/reduce
  - Distributed Caches w/ affinity
  - Collocated compute & data.



LET'S START AT THE BEGINNING.

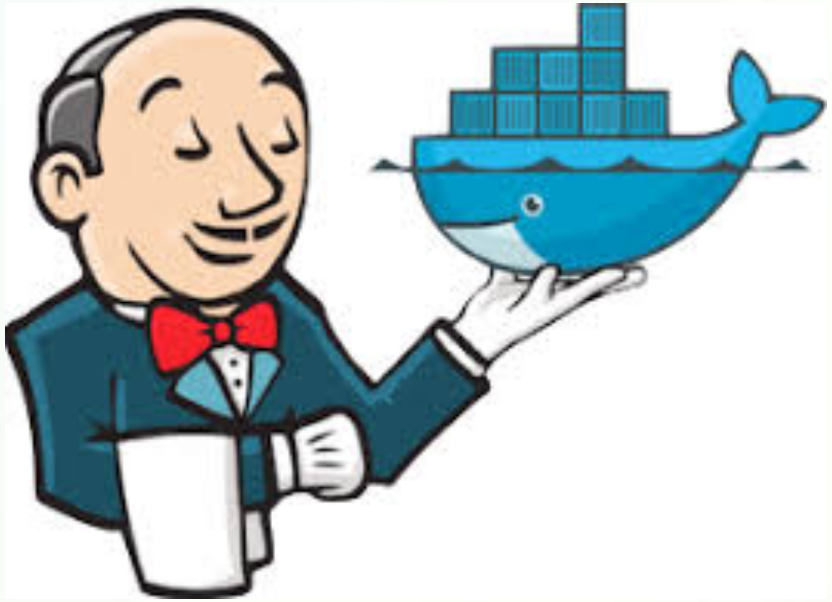
- Already had a well written, well tested code base
  - So. Don't poke the bear
  - Perturb things as little as possible
  - Allow for continued feature work
- Ignite is almost a drop-in replacement
  - HashMap => IgniteCache

SO. HOW EXACTLY DID THAT WORK?

1. Added "hooks" to allow extensibility where there wasn’t.
   - Allowed for an alternative store (HashMap), etc.
2. Extracted the existing Server as a standalone JAR
   - Excluding all the old school bits (like Jersey1).
3. Dropped this JAR into a modern, Dropwizard application
4. Rolled that all up into an executable JAR
5. And saved the artifacts off into a Maven repo.

AND THEN WHAT?

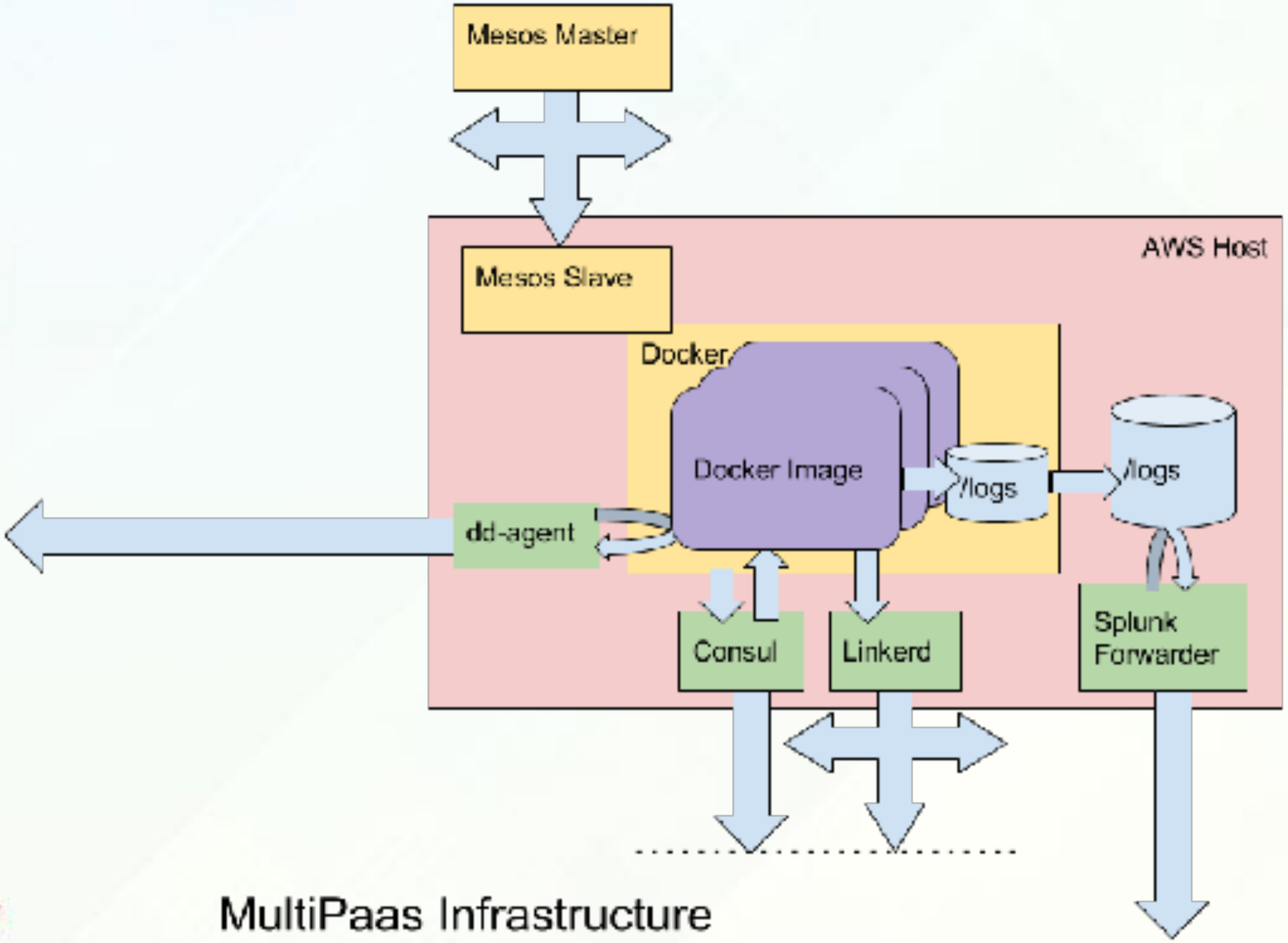
- Created a reusable Jenkins job for Dropwizard apps
  - Takes the Dropwizard, executable JAR from the Maven repo.
    - Plus all of the necessary configuration.
  - Wraps it up with some reusable , bash “control scripts”
- Rolls all this into a Docker Image
  - Using Docker ONBUILD & build-args to parameterize.
  - On top of “blessed Images”; Ubuntu & Java8
- And drops it into a Docker repo.



SO. NOW WE HAVE AN DOCKER IMAGE.
LET'S DEPLOY IT TO AWS.

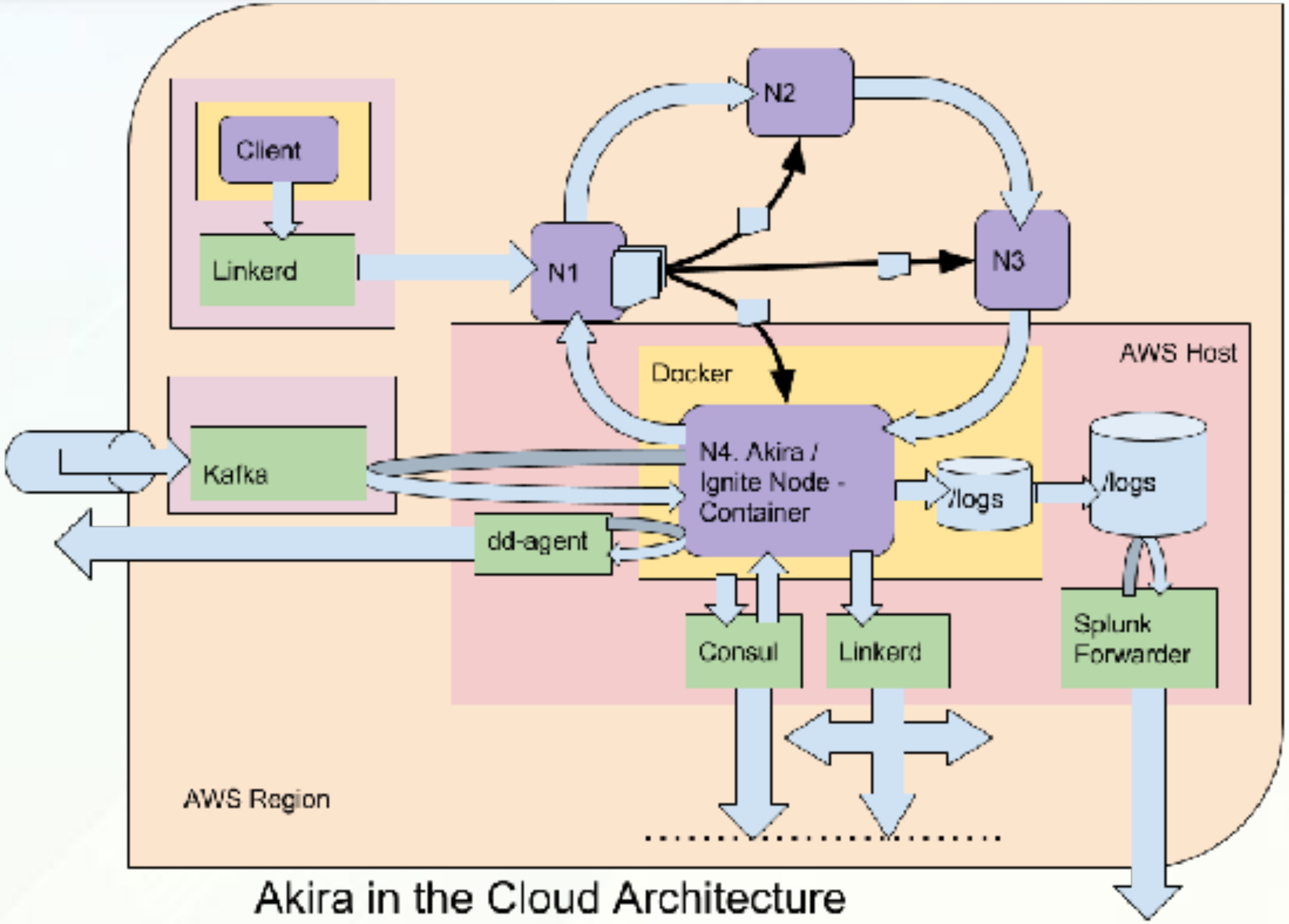
- MultiPaas – HomeAway's Platform as a Service
  - Uses Mesos/Marathon for the orchestration of Containers.
  - Employs Docker to run Containers.
  - On a base infrastructure (AMIs), Terraform-ed onto AWS.
  - Consul for Service Discovery.
  - ContainerPilot to make Service registration easier.
  - Splunk for log forwarding.
  - Datadog for metrics. (Hacked for Consul Service Discovery)
  - Linkerd, Styx, & dnsmasq to create a Service Mesh.
  - Vault for secrets

PUTTING ALL THAT TOGETHER, THE BASE INFRASTRUCTURE LOOKS LIKE THIS...



MultiPaas Infrastructure

AND LAYERING AKIRA ON TOP OF THIS INFRASTRUCTURE.



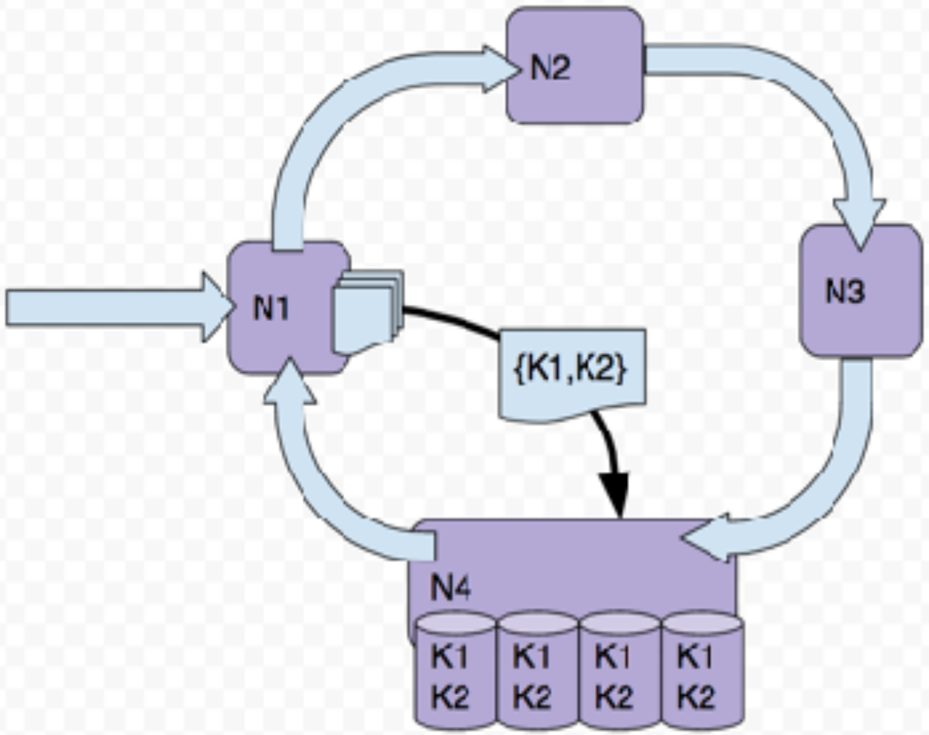
Akira in the Cloud Architecture

OK. SO, LET'S GET SPECIFIC.

- How does all this fit with Apache Ignite?
  - How did you tie the room together?

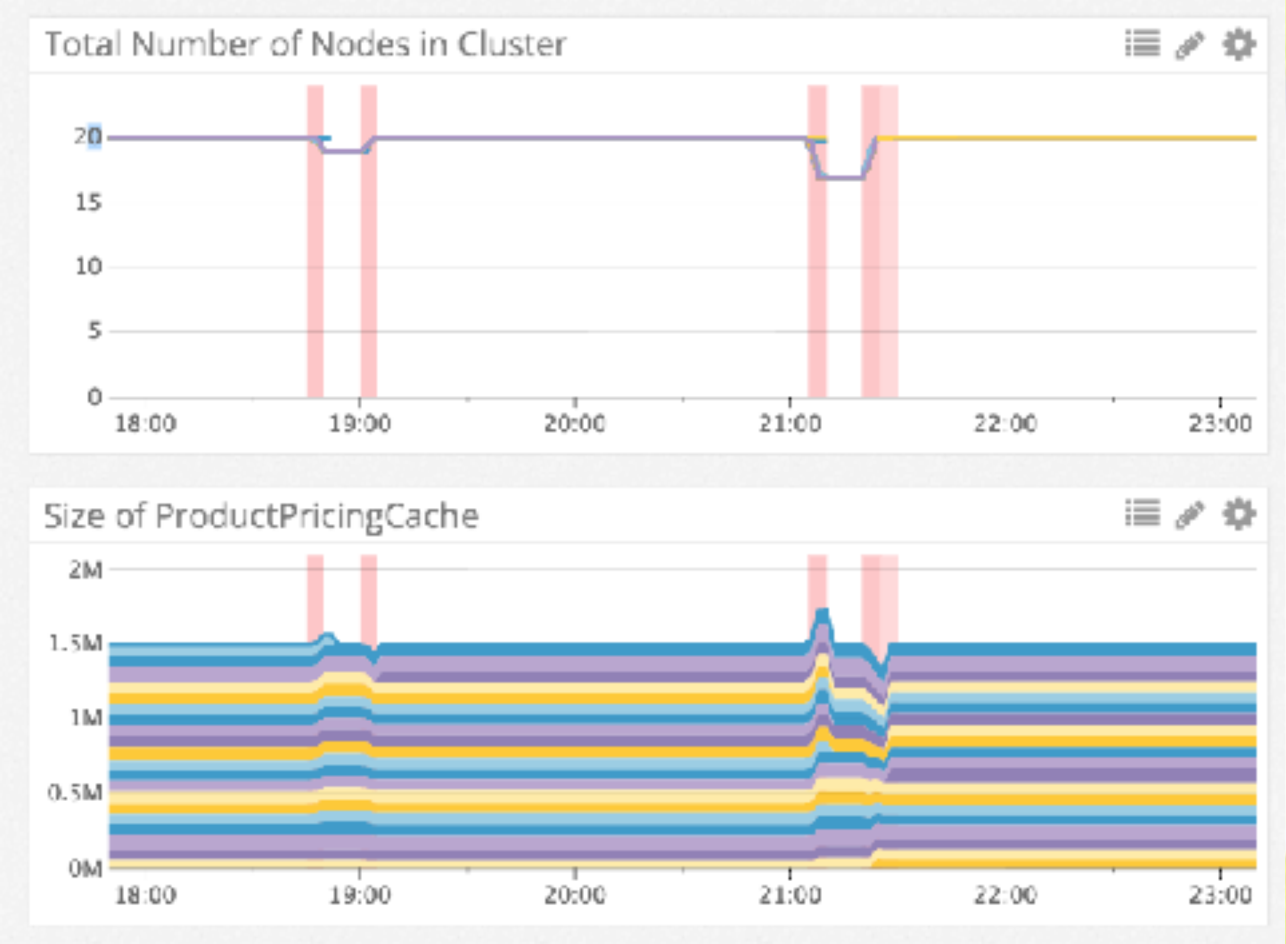
THE COMPUTE GRID & CACHE AFFINITRY.

- Rely heavily on the collocation of data and computation.
  - **Computation:** Deploy the same code on every Node
    - Within a Docker Image using MultiPaas.
  - **Data**: Every cache uses the same Key set.
    - Several PARTITIONED caches
    - Ignite's Affinity functions ensure all data for a given Key is on the same Node.
    - We use setLocal to ensure that we either stay local or throw an Exception.

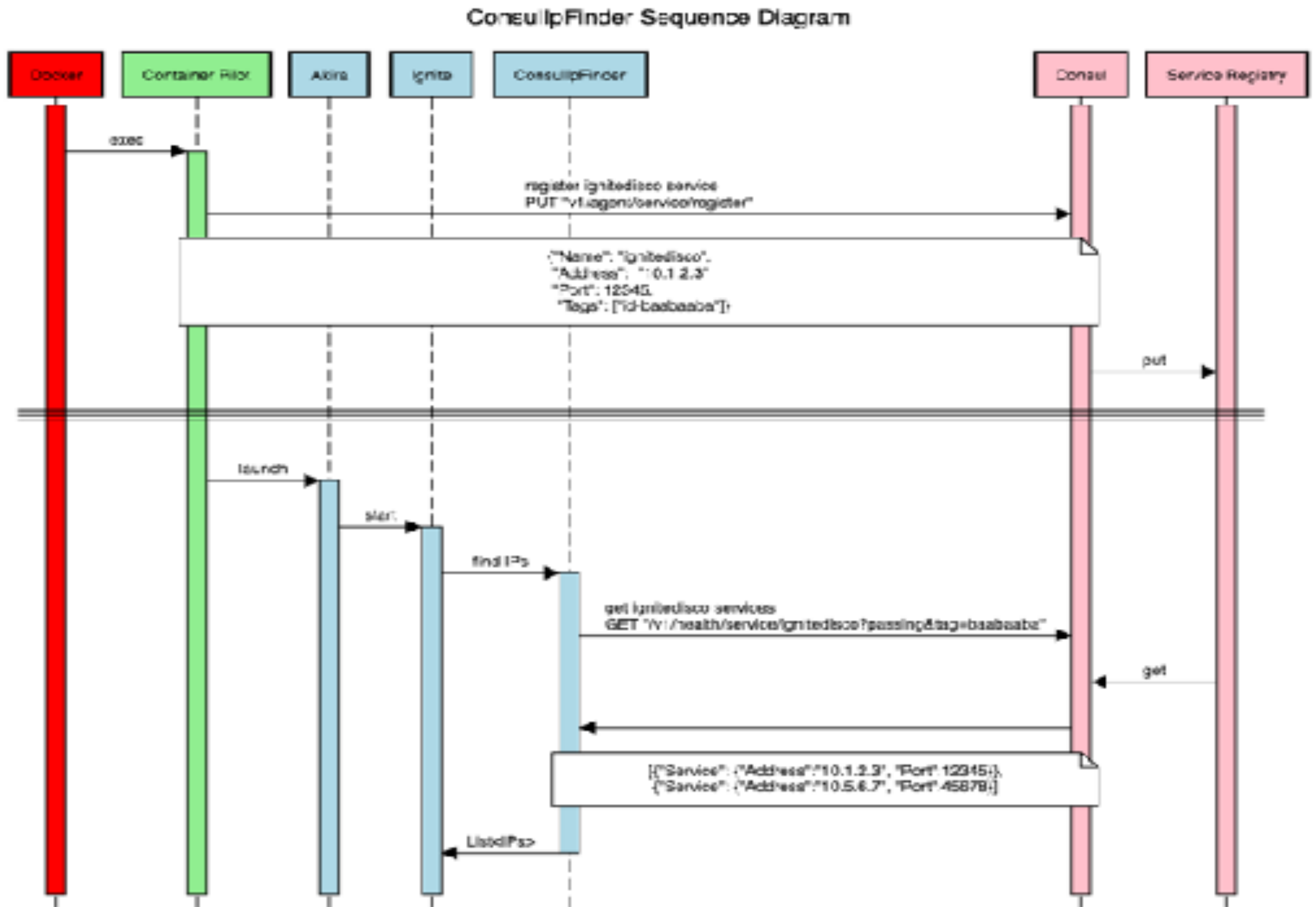


CACHE RESILIENCY.

- Keep several Backups, and allow reads from all.
  - 2TB of data total, in Primary and Backups
- Thus, we can withstand the ephemeral nature of Nodes in the Cloud.
- Yields, overall, a resilient, well-performing system.

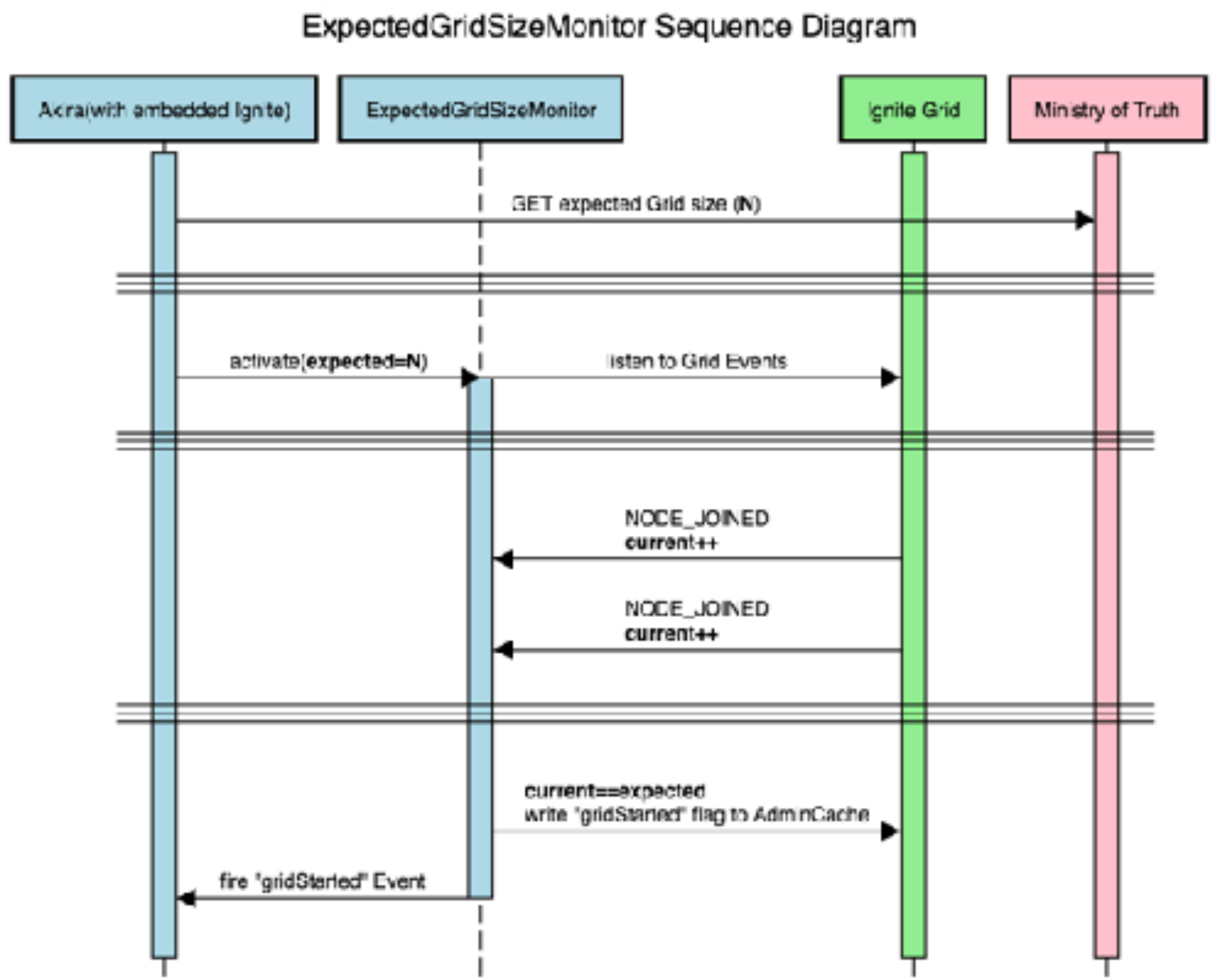


THE CONSUL IP FINDER.
NODE DISCOVERY BASED ON CONSUL.


ConsulIpFinder Sequence Diagram
Docker
Container Rlicr
Akira
Ignite
ConsulIpFinder
Consul
Service Registry
exec
register ignitedisco service
PUT "v1/agent/service/register"
{"Name": "ignitedisco",
"Address": "10.1.2.3"
"Port": 12345,
"Tags": ["id-baabaaba"]}
put
launch
start
find IPs
get ignitedisco services
GET "/v1/health/service/ignitedisco?passing&tag=baabaaba"
get
[{"Service": {"Address":"10.1.2.3", "Port":12345}},
{"Service": {"Address":"10.5.6.7", "Port":45678}}]
ListofIPs>

In-Memory Computing SUMMIT 2017

THE EXPECTED GRID SIZE MONITOR.
HOW DO WE KNOW WHEN THE GRID IS WHOLE?



ExpectedGridSizeMonitor Sequence Diagram

Akka(with embedded Ignite)

ExpectedGridSizeMonitor

Ignite Grid

Ministry of Truth

GET expected Grid size (N)

activate(**expected=N**)

listen to Grid Events

NODE_JOINED
current++

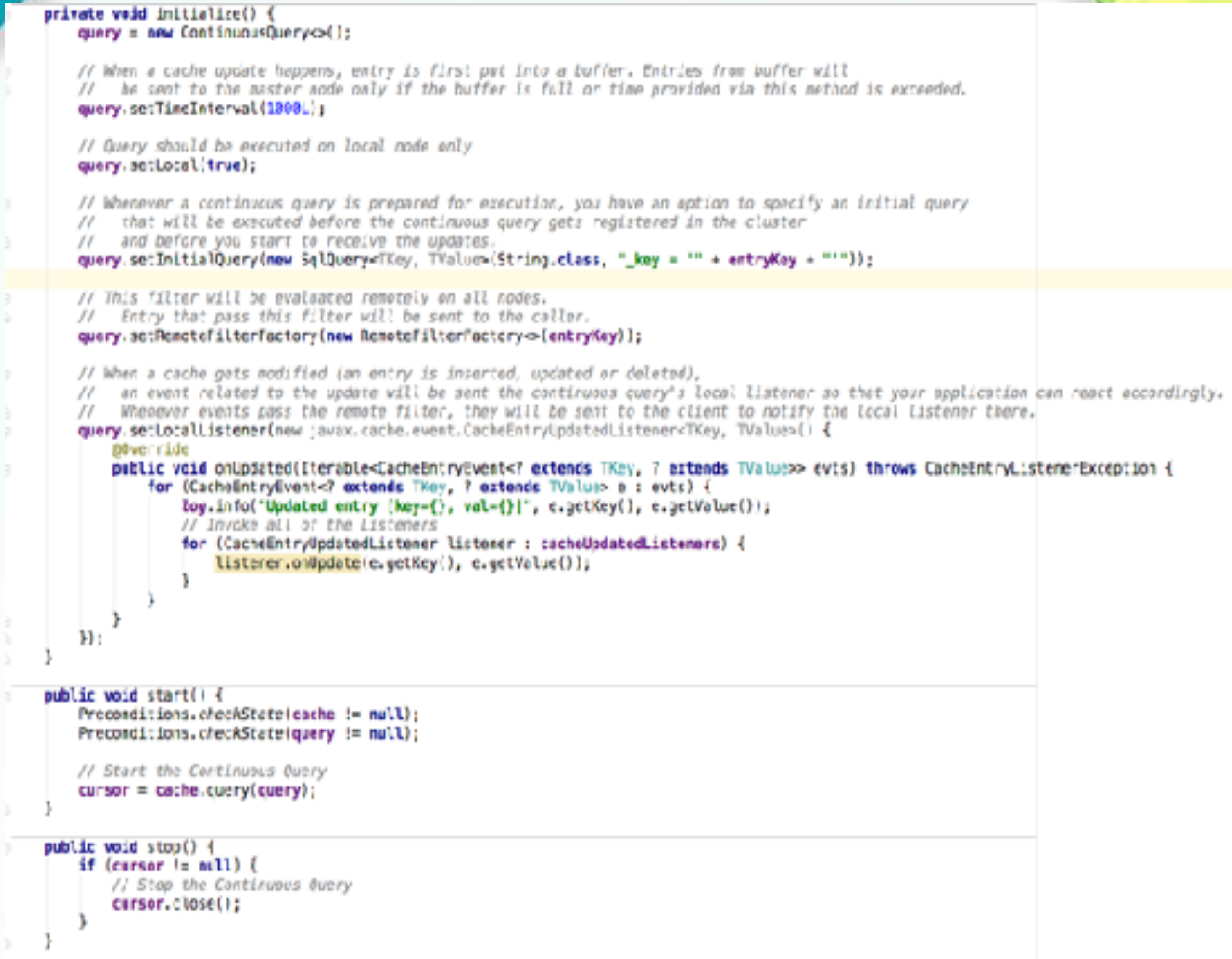
NODE_JOINED
current++

current==expected
write 'gridStarted' flag to AdminCache

fire 'gridStarted' Event

THE CACHE ENTRY UPDATE MONITOR.
WATCHING FOR ADMIN EVENTS.

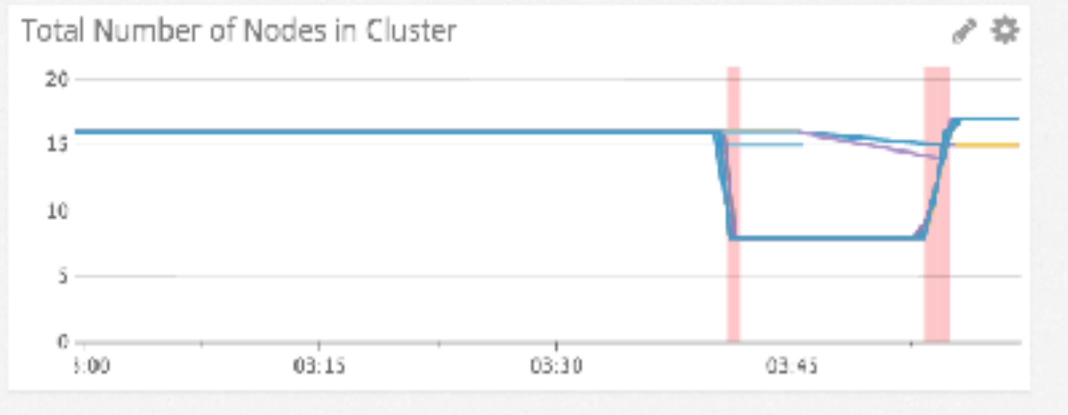
- CacheEntryUpdateMonitor
  - Watch for updates to a single cache entry
  - Uses a Continuous Query
  - Used for various entries in the AdminCache
  - Use only for low volume entries



```java
private void initialize() {
    query = new ContinuousQuery<>();

    // When a cache update happens, entry is first put into a buffer. Entries from buffer will
    //    be sent to the master node only if the buffer is full or time provided via this method is exceeded.
    query.setTimeInterval(1000L);

    // Query should be executed on local node only
    query.setLocal(true);

    // Whenever a continuous query is prepared for execution, you have an option to specify an initial query
    //    that will be executed before the continuous query gets registered in the cluster
    //    and before you start to receive the updates.
    query.setInitialQuery(new SqlQuery<TKey, TValue>(String.class, "_key = '" + entryKey + "'"));

    // This filter will be evaluated remotely on all nodes.
    //    Entry that pass this filter will be sent to the caller.
    query.setRemoteFilterFactory(new RemoteFilterFactory<>(entryKey));

    // When a cache gets modified (an entry is inserted, updated or deleted),
    //    an event related to the update will be sent the continuous query's local listener so that your application can react accordingly.
    //    Whenever events pass the remote filter, they will be sent to the client to notify the local listener there.
    query.setLocalListener(new javax.cache.event.CacheEntryUpdatedListener<TKey, TValue>() {
        @Override
        public void onUpdated(Iterable<CacheEntryEvent<? extends TKey, ? extends TValue>> evts) throws CacheEntryListenerException {
            for (CacheEntryEvent<? extends TKey, ? extends TValue> e : evts) {
                log.info("Updated entry [key={}, val={}]", e.getKey(), e.getValue());
                // Invoke all of the Listeners
                for (CacheEntryUpdatedListener listener : cacheUpdatedListeners) {
                    listener.onUpdate(e.getKey(), e.getValue());
                }
            }
        }
    });
}

public void start() {
    Preconditions.checkState(cache != null);
    Preconditions.checkState(query != null);

    // Start the Continuous Query
    cursor = cache.query(query);
}

public void stop() {
    if (cursor != null) {
        // Stop the Continuous Query
        cursor.close();
    }
}
```

THE AVAILABILITY ZONE (AZ) AWARE BACKUP FILTER. ENSURES WE CAN LOSE A WHOLE AZ.

- The AzAwareBackupFilter
  - Plugs into the RendevouzAffinityFunction in Ignite
  - Balances the Backups and the Primary evenly across the AZs
  - Allows the full Grid to span AZs
    - All partitions are in all AZs, twice
    - Ensures we can lose an entire AZ.
  - Uses knowledge encoded into the ConsistentNodeId.
    - To determine Env, Region, and AZ

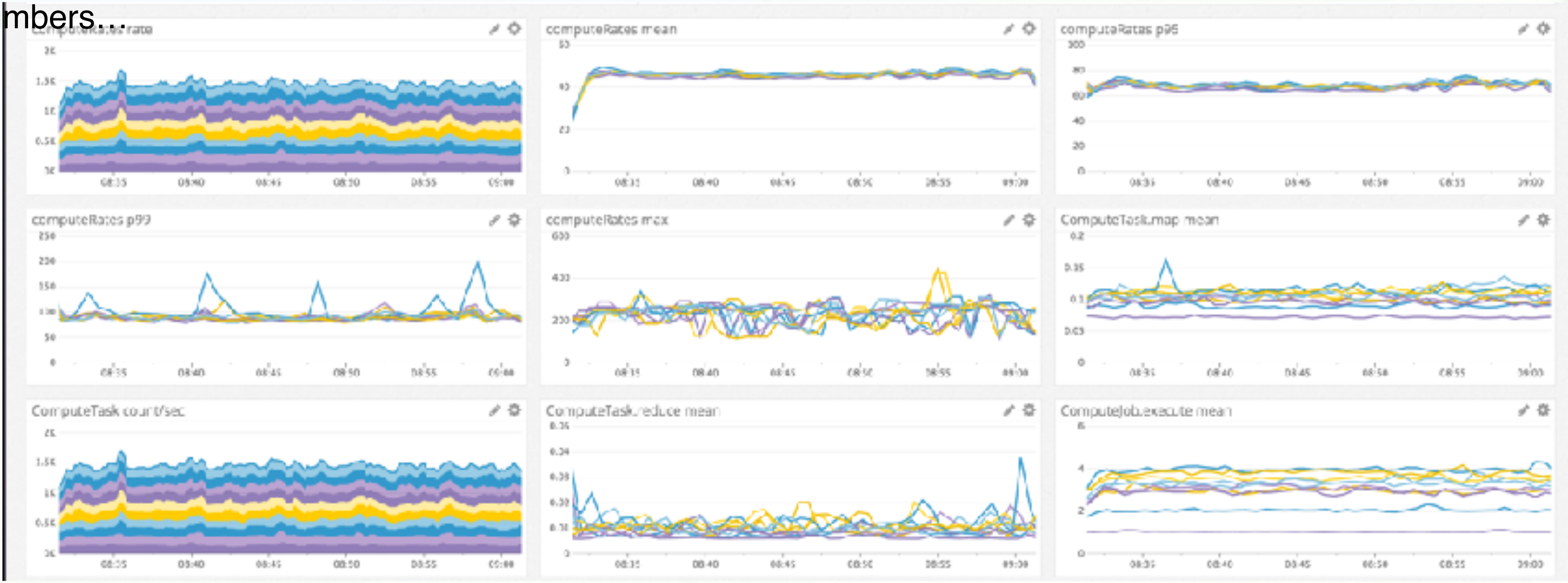


REQUEST-SCOPED TRACING.
DEBUGGING IN A DISTRIBUTED WORLD IS HARD.

- Enables debugging of complex operations where verbose logging is impractical
  - Registers a Jersey ContainerFilter
  - And a custom Logback MDCFilter
  - Dynamically toggles the logging level to TRACE by sending in a special Request Header.
  - Eventually, passes these ThreadLocal MDC flags into the ComputeJob, so we can track the Request across the Grid
  - And finally, into the logs
    - logFormat: '[%date{''yyyy-MM-dd''T''HH:mm:ss,SSS'',UTC}]\(%t\)\([%X{requestMarker}]\) %p - %logger{0} - %m%n%r'

SO. HOW'S THAT GOING FOR YOU? CURRENT PERFORMANCE

We are nearing our prescribed perf numbers…



THE END.

- Any questions??
  - (BTW: we're hiring :~)