The Happy Production Model

Maxim Noah Khailo

May 3, 2016

1 Purpose

The best production model is a happy one. Software is not some ephemeral fairy that lives in a world outside out own. It has a real physical nature. When you build a software system, you are building a machine, with moving parts.

However, machines tend to break down and need constant maintenance. This makes engineers unhappy. So we must resurrect the echo of Alan Kay to flip our idea of software running on a machine on it's head. There are other machines that are more resilient, that have maintenance built in as automatic systems. These are biological in nature. So we should look to nature for a system that is more resilient against entropy.

A happy production model is built on one that models more of a biological system than a mechanical one.

2 Properties of Biological systems

Biological systems have several properties we should emulate.

1. Redundancy
2. Communication
3. Decentralization with Centralization
4. Scale
5. Self Healing

Redundancy is necessary for resilience because we want to eliminate any single point of failure.

Biological systems are composed of trillions of self contained systems called cells. These cells are encapsulated in a membrane and achieve coordination via communication. True object oriented programming involves communication between components. In software systems, the communication is usually achieved via network protocols.

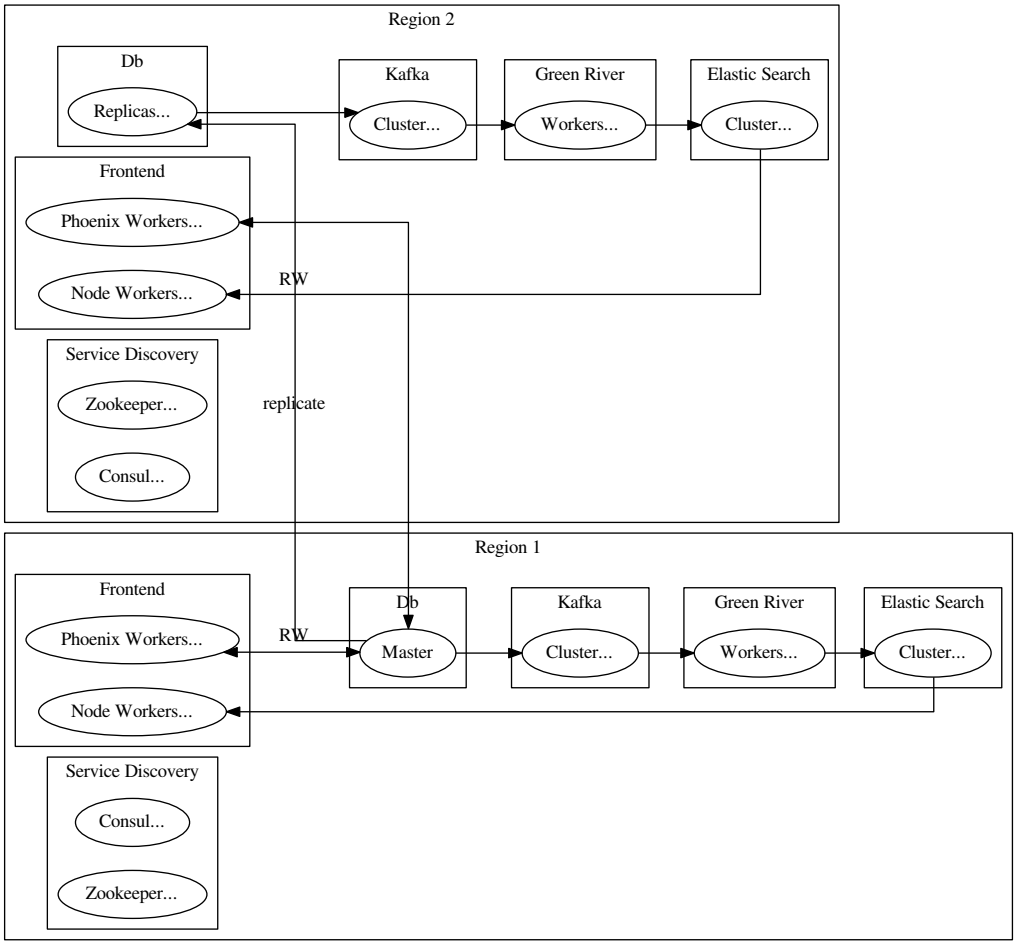
Higher level biological systems have a very interesting property. Despite being composed of trillions of decentralized communicating pieces, they also have central nervous systems helping coordinate functions. This is no accident. Performance increases for decentralized software systems when a small amount of centralization is introduced.

Biological systems are designed to grow. Different parts grow at different rates and at different times. For our production system, we will have to make it clear where the software can grow and by which stimuli.

Our bodies have agents that constantly clean the system. Our skin heals when you cut it. The fight for life is the fight against entropy. Similarly software systems need constant monitoring and healing. Machines and services that are not communicating anymore need to be automatically killed and replaced.

3 MVP Production Model

Given the properties outlined above, what would a minimal FoxCommerce production system look like?



3.1 Two Regions and DB Asymmetry

The first obvious feature of the architecture is that the system is essentially mirrored in two regions. Both regions will receive traffic to the frontend instances.

The system symmetry breaks at the DB level. There will be one master DB that both regions write two. This master will have replicas in both regions where changes are streamed using the write ahead log.

When one region goes down, the frontend instances should seamlessly switch over to the failover master in the other region. There is a mall danger of data loss if there is a large delta in time between replication. We can alleviate that by requiring writes to only succeed if the replication is successful.

3.2 Cluster Separation

Since each region is a copy of the other, we will keep them isolated except for the DB read/writes and replication.

This means we will not cluster Kafka, Green River, or ElasticSearch across the regions. Since all data comes from the DB, the Green River workers should process the same data and ElasticSearch should have the same index.

Given this, the front end machines can access their respective ES clusters in their region. This should help provide nice scalability and load distribution across regions.

4 Scalability of Components

4.0.1 DB

TBD - Sharding? Someone needs to do research here.

4.0.2 Kafka

We will create the appropriate amount of partitions per topic depending on the topic size. If the topic is large and the messages independent then we can partition the topic into N partitions and have N Green River workers processing that topic.

Unfortunately we cannot autoscale Kafka and need to be thoughtful in configuring the topics and partitions. Should we need to add brokers, we would need tell Kafka which partitions to move over to the new machines.

4.0.3 ElasticSearch

Elastic Search has a master/replica setup and does not autoscale well. Similarly to Kafka, to load balance the data, re-sharding can be done but it is an expensive operation and requires re-indexing.

4.0.4 Phoenix and Ashes/Firebird

Since both Phoenix and Ashes/Firebird are stateless services, autoscaling them and putting them behind a load balancer is fairly trivial.

We can decide to separately scale Phoenix and Ashes/Firebird or we can put them on the same machine and scale them together.

The advantage of putting them on the same machine is reduced latency. The disadvantage is we would require more DB connections than necessary.

5 AWS System Cost Estimate

The cost described below is a range and uses AWS EC2 prices. I used the standard hour prices here for instance types. I have also not factored in storage costs like S3, snapshots, and other backups. Typically storage is the cheapest part of the system.

The total prices below are per month for one region The 2xRegion cost is also listed.

5.1 Medium

System	Machine	Count	1xPrice	Total/Month	Notes
Consul/Zookeeper	t2.small	3	$18.72	$56.16	
DB	i2.2xlarge	1	$1227.60	$1227.60	The other region will be replica
Kafka	d2.xlarge	2	$496.80	$993.60	Not using SSD here since access is linear.
ElasticSearch	r3.xlarge	3	$239.76	$719.28	1 Master, 2 replicas
Green River	m3.xlarge	1	$191.52	$191.52	Multi workers on one instance.
Phoenix/Ashes	m3.xlarge	1-4	$191.52	$191.52-$766.08	Scales based on traffic.
			total	**$3378.68-$3953.24**	
			2xRegion	**$6757.68-$7906.48**	

5.2 Large

Notes are based on changes from the medium install.

System	Machine	Count	1xPrice	Total/Month	Notes
Consul/Zookeeper	t2.medium	3	$37.44	$112.32	Slightly better machines.
DB	i2.4xlarge	2	$1227.60	$2455.20	One replica per region.
Kafka	d2.xlarge	3	$496.80	$1490.40	More brokers.
ElasticSearch	r3.2xlarge	3	$478.80	$1436.40	More RAM.
Green River	m3.2xlarge	1	$383.04	$383.04	More CPU.
Phoenix/Ashes	m3.2xlarge	1-4	$383.04	$383.04-$1532.16	More CPU.
			total	**$6260.40-$7409.52**	
			2xRegion	**$12520.80-$14819.04**	

5.3 Small

Notes are based on changes from the medium install.

System	Machine	Count	1xPrice	Total/Month	Notes
Consul/Zookeeper	t2.small	3	$18.72	$56.16	No Change
DB	d2.xlarge	1	$496.80	$496.80	Don't use SSD
Kafka	d2.xlarge	1	$496.80	$496.80	One Machine.
ElasticSearch	r3.large	1	$119.52	$119.52	One Machine.
Green River	m3.large	1	$95.76	$95.76	Smaller Machine.
Phoenix/Ashes	m3.large	1-4	$95.76	$95.76-$383.04	Smaller Machine.
			total	**$1360.80-$1648.08**	
			2xRegion	**$2721.60-$3296.16**	

6 AWS Cost Optimization

The minimum cost described above can actually be smaller if we decide to save in a couple of ways. We can save in EC2 costs by purchasing reserved instances. Typically discounts can be up to 50% depending on the license terms we choose and how much we pay up front. We should investigate if we can use the AWS credits we have towards buying reserved instances.

Another cost savings approach we can take is to use spot instances for our stateless services. AWS has a market for purchasing unused EC2 capacity at a steep discount. The prices are pretty volatile but the discounts can be steep, up to 80% off.

The downside of spot instances is that they can be killed at any time should should the spot price rise above the price you paid. I think spot instances could be appropriate for scaling Ashes and Phoenix instances.