Temporary Thesis

**Introduction**

Starting a web-based company is a very risky endeavor. Recent statistics place the startup failure rate above 90% for varying reasons [1]. This figure should be frightening to many who are trying to begin their own business, especially since the cost of hardware for such an endeavor has traditionally resided in the thousands of dollars before the service can even launch. This, however, is the problem that the Cloud market specifically aims to rectify.

The Cloud, as it is today, is a collection of hardware resources which are owned by several large, established companies and rented out for use by other companies. This market takes many forms, such Google’s Compute Engine and Amazon AWS, but all essentially the same: users who wish to use these resources as a platform for their own service, or for whatever needs they may have, pay for the resources they want and are charged by the amount of time they use these resources. As with any resource someone wishes to buy, when they rent this hardware they are guaranteed to receive it and it cannot be taken away from them until they give it up freely. However, since users of Cloud services ask for the exact resource which they want, the providers of said services have very little control over which of their own servers they need to turn on and how much of these servers is actually being utilized for a given user.

For example, consider the case where only one user among thousands asks Google for their highest speed server type, but they only wish to utilize 10% of this server. Google must now turn on this server, and while it is only at 10% utilization, its power consumption is **[NEED SOURCE]** very high, most likely well over 50%. This leads Google to a situation where they are paying more to sell this resource than they are to receiving from its use, or at very least a situation where they are able to make more efficient use out of a product at almost no cost to them. Seeing this very clear issue, Amazon came up with a solution they called Amazon EC2 Spot Instances [2]. Spot Instances are Cloud computing resources that make up the unutilized space on currently active Amazon servers and are sold to the highest bidders attempting to utilize this resource for less than the standard rate of a Cloud resource. The catch to these is that they are not guaranteed as available to any user for any amount of time and thus can be taken away as soon as someone is willing to pay more for the resource or when the server is no longer needed for on-demand customers.

These spot instances, and their equivalents at other companies such as Google, have provided a unique opportunity. Companies seeking to minimize costs have the option to “take a chance” on these unreliable resources and utilize them as opposed to on-demand servers at full price. While on the surface this seems like a risky endeavor, especially for new companies who can’t risk intermittent service outages while trying to establish themselves, there would appear to be ways to avoid losing service while still benefitting from these affordable prices. **The problem we now face is how do we implement a service with 99.99% reliability for the minimum cost available?**

I believe the solution to this is to utilize pre-emptible VMs or spot instances in such a manner that their inherently failure-prone nature is hidden by having more available at a given time. By having more nodes available, you reduce the risk that some are taken away from you at any given time, as the traffic that was once there is just placed on the other available nodes. Currently, Google’s pre-emptible VMs cost 40% of their typical per hour costs for their smallest, most basic compute instance [6]. Given this, we know that if we can create a service that utilizes this type of instance, we can afford two and a half pre-emptible instances for every one on-demand instance we would need. That means that if we can implement a 99.99% reliable service using less than five pre-emptible instances for a service that would take two reliable instances, we have saved money.

The real solution will not be quite so simple though. The numbers and pricing above relate only to the smallest available standard compute engine instance. The final solution will first determine how big an instance will need to be to handle the pods we have designed. This instance size will be considered the baseline. Additionally, depending on the variety of pods needed for the entire service, there may be two or more “baseline” instance sizes to consider, considering that pods will be performing different functions and each type may require a different amount of computing resources. After the baselines are determined, we will need to determine how many on-demand baseline instances are needed to provide 99.99% reliability to some arbitrary **(SHOULD THIS NOT BE ARBITRARY?)** number of users. We will attempt to determine the lowest price possible for on demand instances by utilizing bigger instances with more pods (i.e. creating two pods on an instance that is two times the size of our baseline instance) until we are satisfied that we have found the lowest on-demand price possible.

After this testing is complete for the on-demand instances, we will begin working with pre-emptible VMs. We will have our maximum cost allowable for Google Compute Engine, and from here we will attempt the lowest cost possibilities using pre-emptible VMs. Determining what these will be should be simple, as Google has guaranteed pricing on their pre-emptible instances. Unlike Amazon Spot Instances, these costs will always remain the same, so determining the lowest cost possible after having determined the computing power needed will be a matter of simple math. However, we will then need to implement this and determine the reliability and revocation rate of the types of servers we have chosen. Unlike Spot Instances, pre-emptible VMs are still very new and lack the revocation history data needed to make accurate predictions about which instances are likely to fail before it happens. This means we will need to take reliability data as we test. As we gather reliability data, we can then make models to determine how many extra of every instance type should be made in order to keep service at 99.99% reliability. Once we have found the lowest cost solution as a function of per hour cost and reliability of server types, we will finally be able to determine if the pre-emptible VM instances have offered us the lowest possible cost solution on the Google Compute Engine.

To demonstrate this, I intend to create a basic web service and implement it on the cloud using Amazon Spot Instances or Google Pre-emptible Virtual Machines, which is Google’s unguaranteed cloud service created in response to Amazon Spot Instances. This decision is being influenced by numerous factors. Amazon has a much more established spot market, providing very useful past data about their various spot markets that can be used to accurately predict the likely future prices and make the most informed purchasing decisions. However, the Amazon Spot Market is far more complex and utilizes a bidding system which adds to the difficulty in making a perfect algorithm. Google’s Pre-emptible VMs offer a fixed price and a much more fluid integration with Kubernetes, a service that will be utilized to manage server instances. They are also, however, newer and lack the history data of revocations that would be useful in making confident purchasing decisions and they put an absolute limit of 24 hours on any pre-emptible instance they sell. One major factor in deciding is determining how well Kubernetes will be able to manage Amazon Spot Instances across many markets simultaneously. Some claim that “Kubernetes was designed to make working with containers on Google Compute Engine easier” [3]. Using the Google Cloud was the intention for Kubernetes very early on, and it makes sense that this fact would make Google the much more intuitive choice for service.

This web service will be managed by a 2-level management algorithm. The first level acts as a traffic analyzer to estimate the current load on each of the available servers and determine when it is necessary to obtain more resources to maintain reliability and when it is acceptable to drop superfluous resources. When it is decided that the number of resources must change, it advises the second level to respond accordingly. The second level management is the implementation of Kubernetes to manage the Cloud resources. Kubernetes is a very new, open-source project managed by Google which is used to manage server resources across many servers in as efficient a manner as possible.

Kubernetes creates containers, which are sealed application packages that exist as seemingly separated portions of a server resource. Essentially, these containers are seemingly independent instances of computing resources (like separate VM instances) that are managed as such, but which are actually created by dividing a larger machine up so that you can make more efficient use out of this machine. Containers and Virtual Machines are very similar in their respective uses, however there a benefits to each. A Virtual Machine has the benefit of flexibility. By utilizing a Virtual Machine, one could choose to use any Operating System they wanted to use regardless of that of the host. The benefit to using Containers, however, is that they utilize the Operating System of their host, thus negating the overhead and resource waste created by implementing a separate Operating System on top of another Operating System. By using Containers and minimizing or eliminating this waste of computer resources, we reduce the amount of computing resources we require drastically and can save on operating costs. The only requirement to make use of Containers in the cloud is that you must be able to create your service in the type of environment provided by the cloud service providers, but as they offer very typical Operating System choices this is easily done in most cases.

Kubernetes then organizes these containers into pods, which are groups of containers that work together to achieve some goal. Pods are the smallest deployable units as viewed by Kubernetes [5]. They can contain from one to any number of containers, however all containers within a pod need to exist on the same physical or virtual host. Pods are also seen as “immutable”, meaning that once a pod has been created by Kubernetes it remains unchanged for its lifetime. Kubernetes creates pods based on need. It uses a replication controller and its desired state value to determine how many pods should exist any time. If this desired state is not meant, the number of active pods is either increased or decreased. Every pod has a template that it is created from, so in the case that Kubernetes must create more pods, like at startup or when traffic increases enough, it refers to the template of a pod that is needed and creates a pod from that template.

Different types of pods can then work together to create the service we are trying to achieve. As an example, consider 2 types of pods, one is the front end to the web service and the other is the backend. These can work together to act as a fully operational service.

Kubernetes utilizes a Replicated State Machine approach to providing web services. This means that Kubernetes creates a certain number of copies of a service and directs users wishing to access the service to the least utilized instance of the service. The number of copies of a service is called the desired state and is a setting within Kubernetes. When something happens to change the number of available services, such as one instance of the cloud that was available to Kubernetes previously failing or being revoked, Kubernetes will see some pod has failed and will bring an identical one up in its place. To create an identical pod, Kubernetes references the label attached to the pod that has failed and accesses a template of that pod to create the next instance. The desired state is the parameter being managed by the first level of management, which means Kubernetes will bring up and close down resources as it is instructed by the first level of management.

**It is my hypothesis that by gauging the traffic of a service built on unreliable cloud resources we can manage the amount of cloud instances needed to provide 99.99% reliability of a service for the minimum cost available.**

[1] <https://s3.amazonaws.com/startupcompass-public/StartupGenomeReport2_Why_Startups_Fail_v2.pdf>

[2] <https://aws.amazon.com/ec2/spot/>

[3] <https://www.ctl.io/developers/blog/post/what-is-kubernetes-and-how-to-use-it/>

[4] <https://cloud.google.com/container-engine/docs/pods/>

[5] <https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/pods_and_services.html>

[6] <https://cloud.google.com/compute/pricing>

[7]