|  |  |
| --- | --- |
| **Description** | An innovative cloud application provisioning tool that provides advanced mission resource visibility, ease of use, and up to 50% cost savings in the AWS C2S environment. |
| **Submitter Name** | Oscar Ganteaume |
| **Submitter Company** | Axios |
| **Focus Item** | September Focus Item |

# Needs

Cloud Services Providers deliver powerful fee-for-service capabilities which enable users to easily access computing resources. The complexity required to efficiently utilize those resources however, can inhibit a developer from implementing their project in a straight forward and cost effective manner. AWS’ overly complicated auto-scaling feature, for example, typically isn’t used because of the complexity in implementing:

* Custom Threshold Metrics
* Service Request Roles
* Launch Configurations
* Auto Scaling Groups
* Alarm Creation
* Policy Creation

More commonly, a developer will statically over provision resources to guarantee functionality.

Zephyrus’ primarily design goal is to abstract and simplify dynamic resource allocation and load balancing by hiding AWS’ inherent complexity. The added benefit of elegantly applying dynamic resource control, results in significant AWS cost reductions.

# Approach

Axios, an LGS Innovations company, supports a number of IC and DoD customers to include the Ground Enterprise Division (GED) of the NRO in the areas of cloud based mission frameworks, cloud infrastructure services, and signal processing applications. This synopsis will describe a mission prototype called Zephyrus that was developed by Axios under IRAD, and later used in support of our GED Seven Hills program. As a result of our tests, we believe Zephyrus will have a positive cost & performance impact for the NRO if matured under Iron Patriot contract.

Zephyrus was developed under internal IRAD to explore cloud agnostic resource allocation, workload load balancing, and cost containment in anticipation of the governments cloud processing initiatives. In addition to greatly simplifying C2S resource provisioning, saving valuable labor hours, Axios also successfully implemented dynamic load balancing to efficiently distribute mission processing loads. Our Zephyrus financial results also show up to 50% cost savings in the C2S environment over a stock C2S environment.

The Zephyrus solution described below will walk you through the key features of the tool and how they can be of benefit to the overall NRO mission. If the Zephyrus features and benefits resonate with the Iron Patriot program office, Axios proposes taking this IRAD prototype and developing it into an enabling mission application intended for widespread C2S cost savings and use.

Taking into account the current shortcomings of the AWS services, Zephyrus’ primarily design goal is to facilitate dynamic resource allocation while hiding its inherent complexity, dynamically “right-sizing” workloads & providing visibility in a cloud agnostic environment.

Zephyrus has the ability to start virtual machines only when needed and terminating them when no longer needed. It offers load balancing and auto-scaling features, but also provides a method for customizing the algorithms to do so if warranted by mission circumstances.

Zephyrus will also allocate a virtual machine to any session that was not able to launch a task due to lack or resources. The VM would be provided by a free agent if available or newly created, to expedite the provisioning process.

Zephyrus monitors the states of all the virtual machines running within the system grouping them by sessions and terminating those that are no longer needed. As mentioned previously, Zephyrus allows full customization of the conditions in which virtual machines are initialized or terminated. This continuous monitoring and task orchestration approach is what enables Zephyrus to save provisioning costs while isolating the application from a cloud service specific configuration.

Zephyrus features a web-based visualization application that allows you to monitor the status of the managed applications, and allows you to quickly determine the health of each session using a near real-time color-coded table. The user can also visualize the health of each virtual machine associated to a session by clicking on the session name.

If the operator is interested in seeing specific tasks then s/he can select a single virtual machine. Zephyrus loads all the information containing the tasks running on the particular virtual machine. Selecting a particular task displays detailed status information such as start time, run time duration, and where it’s running, et cetera.

# Benefits

**[Oscar: we need ease of use benefits here.]**

Zephyrus 's primarily goal is to facilitate the use of cloud services by hiding its inherent complexity. After an initial configuration in which a Virtual Machine (VM) image is defined and adapted for a specific cloud service provider, users will be able to create, modify, remove, and launch tasks without having to know how they are executed in the cloud. To accomplish this goal, Zephyrus provides a web application with configuration, tasking, and visualization pages. Each page is designed to minimize cloud services knowledge requirements. The configuration page allows administrators to configure the system’s cluster behavior. Users can monitor the system’s health information through the visualization page.

With support from our GED Seven Hills program, Axios tested the Zephyrus concept on a multi-user mission application and compared operational costs. We ran two different scenarios and show the cost comparison in the table below. The first scenario provides resources for up to 60 simultaneous users assuming a typical mission usage scenario, and consumption of On-Demand Instances. The second scenario assumes there is some a priori knowledge of the production environment, which would allow the user to purchase Reserved Instances at a reduced cost in comparison to On-Demand Instances. In both cases we can see a cost saving of almost 50% when the Zephyrus framework manages the cloud resources.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Service | AWS | Zephyrus/AWS | AWS | Zephyrus/AWS |
| Scenario | Up to 60 users (On-Demand Instances) | Up to 60 user (On-Demand Instances) | Up to 60 users (Reserved Instances) | Up to 60 users (Reserved Instances) |
|  |  |  |  |  |
| ECS | $9,876.84 | $3,567.25 | $6,449.74 | $3,035.56 |
| EBS |  |  | $515.29 | $403.37 |
| RDS | $0.00 | $556.32 | $0.00 | $365.38 |
| Other costs | $0.00 | $0.00 | $0.00 | $0.00 |
| Monthly Total | $9,876.84 | $4,123.57 | $6,965.03 | $3,804.31 |

# Competition

Describe what the alternatives are to the proposed solution.

**[Oscar: we need a paragraph on competition. I presume it’s the hard to use Amazon tools themselves.]**

One of the key features of Zephyrus is its ability to dynamically allocate and deallocate resources as needed. Implementing similar behavior in AWS is a non-intuitive, cumbersome task to do. In order to succeed the developer needs to first install software packages that will be useful and applicable only when working with AWS. Zephyrus on the other hand, uses a cloud service agnostic custom based approach.

Zephyrus provides some default algorithms to determine when to start or stop a virtual machine. It also allows developer to create and use their own based on their own specific needs. The AWS equivalent to this methodology is the use of Custom Metrics. The following section summarizes what a developer would have to do in order to achieve similar auto scaling capabilities:

## Create Custom Metrics

The first step is to develop some code that will run on each virtual machine to collect resource information, create a custom metric item, and submit it to AWS. The generated metrics are now available to AWS and ready for consumption. How often those metrics are sent is determined by the AWS type or resolution, which in turns translate in costs.

## Create a Role

Creating Custom Metrics is relatively straight forward activity as it does not involved modifying an extensive number of configuration parameters compared with some of the remaining steps. A role is defined an Identity and Access Management (IAM) entity that defines a set of permissions for making AWS service requests. Creating a role requires the administrator to create a new policy that would apply to EC2 services. Once the new policy is generated the developer would need to add new CloudWatch services to allow for putting metric data, get metric statistics, and list metrics to the role. The final role should have 3 and 2 CloudWatch and EC2 actions respectively.

## Create a launch configuration

A launch configuration contains all the details about the type of virtual machine to launch. It includes setting up the same type of information administrators would have to provide to Zephyrus to ensure the appropriate virtual machines are used. At this point Zephyrus would have been properly configured to perform auto scaling and load balancing. The steps to create a new launch configuration requires following a set of steps similar to creating a new role. The creation of a new launch configuration option is available under EC2 services in the AWS web page. The process requires the administrator to add the role previously created and enable CloudWatch detailed monitoring. A script would be required to send metrics continuously and can be specified under the User Data section. After that the administrator will need to set additional parameters such as storage and Security Group details.

## Create an auto scaling group

The auto-scaling group expects a launch configuration with all the parameters set regarding what kind of virtual machines to launch. Auto Scaling is the actual service that starts or stops the virtual machine based on alarms. Alarms is what is triggered by the custom metrics tying the service and the virtual machines. Each Auto Scaling group requires a Launch Configuration such as the one described above. Auto Scaling services can be found under the EC2 section in the AWS console. After the developer clicks on create a new Auto Scaling Group, the page will ask for a set of configuration parameters. The administrator will need to set Virtual Private Cloud (VPC) and the subnet as desired. Under the advanced details the administrator will need to set the Health Check Grace Period and to Enable CloudWatch detailed monitoring. The next set of steps allows for setting the initial size of the group. The configuration above just tells AWS to make sure one EC2 instance is available at any time for this Auto Scaling group. The policies regarding when to start or stop a virtual machine is specified on each alarm tied to this Auto Scaling group. The process of how to create alarms is provided in the next section.

## Creating alarms

To be able to scale up and down we need to specify the conditions as of when to do so for each type of scaling. This is accomplished by creating two alarms: one for scaling up and one for scaling down. Zephyrus comes with already built-in algorithms to scale up and down which saves users from having to create their own. Out of the box Zephyrus gives users the chance to scale based on resource utilization or number of tasks running on each virtual machine. Each alarm is configured to read and interpret the metrics and to send an alarm based on that. The alarm is used to trigger either adding a newly created Virtual Machine to the session or removing it from it.

## Add Policies

So far we have created a configuration that tells AWS which type of virtual machine to start or stop and we have also set some alarms to determine when to do so. Policies are a way to join alarms to Auto Scaling groups. Because we have two alarms we need to create two policies, one to scale up and one to scale down, and attach them to the Auto Scaling group created previously. To add a policy the administrator would need to click on the Create a simple scaling policy in the Scaling Group generated previously. To scale up the administrator would execute the policy when the scaling up alarm is triggered. The policy will require to be configured to add a new instance as the action. To scale down the administrator would need to follow a similar set of steps, but removing an instance when the alarm is triggered.

At this point the administrator should have successfully configured auto scaling using custom metrics to run only under Amazon Web Services. A very important point to mention is that this configuration does not provide any form of load balancing between virtual machines. Zephyrus provides load balancing based on resource loading, first fit, and number of tasks without any additional configuration. It also provides users the opportunity to define a custom load balancing as explained before.

The biggest challenge a developer will have discovering how to properly setup the most basic configuration for auto scaling was the need to switch from one service to another. The interdependencies between them and the need to know the order in which components need to be created made it difficult to follow. For instance, you need to create the Auto Scaling Group without policies, which is done under the EC2 Service dashboard. This requires the assumption or knowledge that the policies will be added later. The following step is to create the alarms which requires having an existing auto-scaling group even though is not fully configured. The alarms are part of the CloudWatch Service dashboard and not the EC2 Service. Alarms are needed to create policies that need to be added to the auto-scaling group. This step can take a lot of guessing and trial and error because the final step is to go back to the EC2 Service dashboard to create the policies attached to the alarms. Once the policies are setup and tied to the alarms we can go back to the auto-scaling group configuration and add the policies to it.

AWS auto-scaling approach works well in certain situations, but that is not always the case. The AWS approach pushes the responsibility to make sure all the tasks sent to the application are executed properly to the application itself. Because it does not take into consideration which tasks or services are running on each virtual machine, it could potentially terminate an EC2 instance before it finishes executing a task. This mean that the application itself would have to add resilience to minimize negative effects on the service it provides. A simple example demonstrating this would be using the CPU utilization as the metric to stop a virtual machine such as the one configured previously. AWS would stop the instance as soon as the CPU falls below a threshold for a pre-determined period of time regardless what the instance is doing. If there is a job waiting for something to happen, it would never finish its work if the instance is terminated. The Zephyrus framework takes into account the number of tasks running in each virtual machine before terminating it.

Regardless of how simple or complex configuring load balancing and auto scaling is for a particular cloud service, moving a system to a different provider could have a big impact on the application itself.