**Citrix Nimbus**

Automated deployment & monitoring of Micro-services for CloudOps

Introduction:

With the advent of microservices, it has been increasingly difficult to manage, monitor instances spanning across multiple Availability zones and deploying new builds across these servers. There has been an increasing demand to come up with a modularized dashboard for monitoring and easy automated deployment for a group of microservices.

So, the ideal solution to the problem will be individual teams managing a set of microservices that they own, and have a portal to do so. This will also remove the ownership on one team to manage all deployments, and will have dedicated owners for the set of microservices that they own.

Problem Statement:

Cloud Ops team has been burning midnight oil to get the new builds across the instances, yet waiting on the instances to respond back with the status.

Here are a few points:

* There is no single point of reference to monitor the server stats and deploy the new builds.
* With more and more cloud workspaces coming up, it could be more of a learning curve for the CloudOps guys learn. If an automated script could do this for them, it would really come handy.
* Nagois client sends back very few stats. There has always been an urge for more device level information.
* Since, every deployment is individualistic in nature and a very tedious task, where there is always a possibility that something can go wrong.
* We have to rely always on the cloud service provider to get us stats on the DB performance.
* Responding to alerts could be difficult, as the CloudOps team does not necessarily know what could have possibly gone wrong with the server. If automated scripts based on monitoring results could fix the issues. For e.g.: if there is low disk space, automated scripts could automatically alert the concerned team and as well as clear up some space without any manual intervention.
* Log management is really difficult in case of microservices, since there are multiple servers running on different AZs. And in order to understand what went wrong, the CloudOps person manually collects all the logs files individually. Wouldn’t it be great if an automated script could do this for us.

There will be more and more demand to host multiple microservices in the future. The problems could be more widespread and diverse. And without a single interface to manage, it would be even more difficult for the CloudOps team.

Proposed Solution:

So, we need a one stop solution for:

* **Configuration Management**: configure new servers in the cloud with all the desired rules and logic in place.
* **Analytics**: Server statistics includes HDD, heap/cache memory monitoring, memstats, requests/responses, up-time. Databases can also be monitored on all of the above metrics, which means less or no dependency on the Cloud DB provider.
* **Alerts**: to configure alerts and respond to them using single unified interface. An automated way to achieve this would be much more efficient than an individual doing the task manually.
* **Deployments**: All new deployments have to be done using the single user interface, which would generate automated scripts to do the job for the team seamlessly. And since the executed steps are autosaved into playbooks, there is much less room for error. It can also be shared across teams.
* **Modularization**: Since every team gets to monitor, manage and deploy the mcroservices that they own, there is much less dependency on the Cloud Ops team.
* **Secured**: Because of the modularization, every team has its own set of owners for the portal with Role based access, the system is much more secure.
* **Logs Management**: While troubleshooting an issue, the first thing developer looks for are logs. So, the solution must have easy retrieval of logs from the servers.

Architecture:

Here is an architecture sketch of the proposed solution to address the problem statement above. 

Design Details:

The main components are as below :

**Ansible:**

Ansible is an open source, widely accepted product for app deployment, configuration management and orchestration remotely using ssh protocols. It uses no agents and no additional custom security infrastructure, so it's easy to deploy - and most importantly, it uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

Ansible can talk to any cloud services (Azure/AWS/Rackspace/Google) hosted in multiple Avalability Zones using SSH keys.

**Prometheus:**

Prometheus is an open-source systems monitoring and alerting toolkit. Prometheus agent has to run on the instances. Ubuntu 16 comes preinstalled with Prometheus agents.

Its main features are:

· a multi-dimensional data model (time series identified by metric name and key/value pairs)

· a flexible query language to leverage this dimensionality

· inbuilt time series collection happens via a pull model over HTTP

· pushing time series is supported via an intermediary gateway

· exposes a set of REST API to access the timeseries data within a certain interval

**Ansible REST Server :**

This server has to be developed that exposes REST APIs to injest the supplied playbooks, run adhoc tasks intuitively, automate the process of deployment. This would automatically fetch the relevant instance information from the shared DB.

**Shared DB:**

A shared Database will be created that maintains the hostnames and the configurations for the instances. Only Ansible has write access to this DB, which means a new instance can only be added to the DB via the Ansible REST APIs, whereas, the Prometheus server had read-only access on this DB.

This guarantees a single source of truth avoiding concurrent writes.

**CloudOps Dashboard:**

This hosts the CloudOps UI, a singular unified portal with proper authentication and authorization for automated deployments, managing configurations, instance monitoring, managing triggers.

This dashboard is the sole consumer of the REST APIs for Anisble and Prometheus. In addition to this, since all the added instances and the configurations are saved in a local shared DB, there is much less room for error.

The whole project will be simple to set-up and deploy. The plan is to facilitate individual teams to manage a set of micro-services by themselves.