International Journal of Management, IT & Engineering
Vol. 13 Issue 03, March 2023,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

Reduce IT Operations Cost by Modernized Automation

Sudhansu Sekhar Behera
Enterprise Architect & Program Manager
Tata Consultancy Services, USA

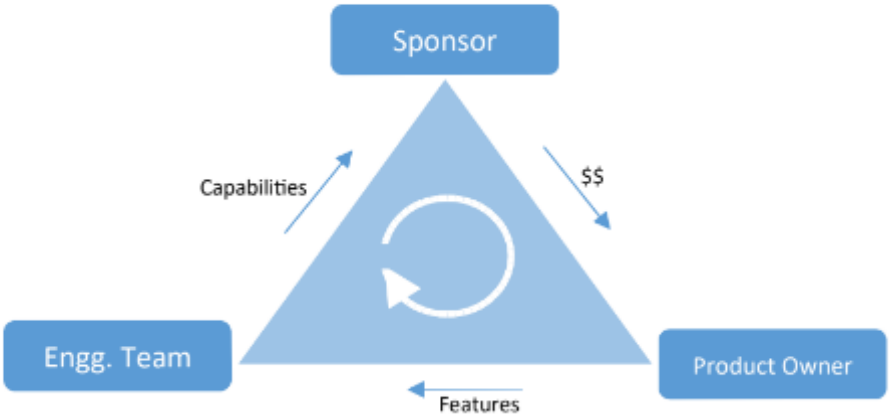
ABSTRACT

Purpose of this research work is to demonstrate the use of IT capabilities (along with coding samples) to reduce IT Operations Cost (a.k.a. OpEx) for major industries. Although the technologies explained in this paper are with Azure, Python, Kubernetes and Unix Shell script, once read the article, it will give a broader insight into enabling the Operation Automation regardless of the technologies used by any Software Application. This research concludes dramatic results in OpEx reduction and beyond.
This paper talks about reducing the "eyes-on-glass" monitoring effort, alerting the stakeholders for a possible catastrophe and thereby automating an executive notification. This research also includes an example scenario on restoring a catastrophe without human intervention.
Keywords: OpEx Reduction, Operational Expense, Monitoring and Alert, IT Operation Process Automation

INTRODUCTION

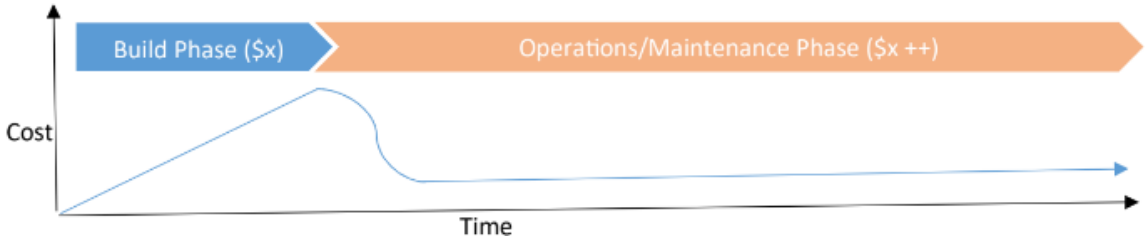
In an attempt to capture the market demand and to enhance the User & Customer experiences, major industries embody new features and capabilities using modern technologies in Cloud and beyond.
While the Product Owners pump in new features, the Engineering/Build team enjoy delivering the features in Agile methodologies under the blessings of Sponsor (as depicted in Figure-1) without predicting the Operational Costs on a longer run. Since there's no linear Cost equation or correlation between IT development and Operations costs, any expert estimation sounds far-fetched until after the hindsight.

Figure -1



While the sponsors take conscious decisions during the build phase, the aftermath of Operations cost (depicted in Figure-2) implications are oftentimes squirrelled away.

Figure-2



Since the Operations Cost consideration is a major deterrent for funding large transformational programs or new program initiatives, it is imperative to explore options to automate manual IT Operations and there by substantially reduce the OpEx of the organizations as a whole.

METHODOLOGY

This research primarily focuses on reducing below (not limited to) IT Operations cost by automation:

- Eyes-on-Glass monitoring transactional failures
- System failures including performance degradation, downtime and unavailability
- KPI and Control Reporting
- Alerting stakeholders
- Application Restoration (Healing)

Methods applied in this paper considers a use case of an organization that has built applications on Azure Cloud PaaS platform equipped with Kubernetes, transmitting data into and from Cosmos database.

Eyes-on-Glass Monitoring

IT Production/Operations Support teams typically comprise of engineers deployed 24x7 constantly and literally keeping an eye in front of computer screens gauging the transaction flows, specially watching out for red flags, or Customer complaints, business notifications around inabilities performing certain business transactions, Customer service telephone calls and/or email notifications.
Modeling section of this paper talks about avoiding/minimizing such notifications/Customer complaints by proactively restoring such situations even before they come to the notice of business users or Customers.

Systemic failures

Generally during peak business hours due to heavy traffic, computing systems' performance degrades, or the intensive use of computing resources cause CPUs shoot beyond 100% usage that brings the systems to a dead-lock situation.
During non-peak business hours, certain upstream or downstream systems perform maintenance activities such as system upgrade or patch upgrade causing downtime.
Such situations necessitate Operations engineers manually intimate the business users to wait until restoration.

KPI & Control Reporting

It's the job of Production Support teams to regularly (frequency can be hourly or daily) report the Key Performance Indicators (KPI) to Business Stakeholders to find possible pitfalls in applications and subsequently address the scope of improvement.
Specific Controls are also fed by Operations engineers to measure major bottlenecks e.g. waiting time during UI navigation, processing delay, number of retries and rage-clicks manifesting users' frustration.
Due to lack of utilities or tools, traditionally Operations engineers perform these activities manually which can be easily automated by simple solutions (*refer Modeling Section*)

Alerting

Monstrous applications comprise of countless moving parts such as Micro-services, APIs, API Gateways, Database tables/Collections, Message Queues along layers of integrating mechanism to connect with upstream and downstream systems including third party vendor systems.
Most commonly Production Support Engineers falls at bay because of the fact that it's humanely impossible to closely look at each of the moving parts and momentarily restore them.
It is possible to automate the alerting mechanism not only to notify the operations engineers, but also tag the severity of the alerts depending up on the degree and volume of failures such as informing, alarming, or catastrophic failures.

Application Restoration

Almost all the applications, in today's world, are housed in clustered servers to evenly distribute the inbound/outbound traffic across multiple computing units.

Upon failure of certain cluster nodes, most frequently, Operations Support engineers bring down the severed cluster to divert the traffic to healthy nodes. Once the problematic nodes calm down, they reboot the nodes to bring them back to life for serving the transactions as usual.
Such is the case for even containerized systems where multiple instances of java objects or micro-services serve the same purpose. In such cases, the effected instances (also known as PODs or containers) are brought down to divert the traffic.
While traditionally the Operations support engineers manually perform these activities, they can be automated for restoration without manual intervention.

MODELING AND ANALYSIS

The regular activities that came out of this research methodology are considered the potential candidates for modeling automation. However, there are unforeseen catastrophes or failures which essentially need human intervention and restoration which are excluded in the analysis.

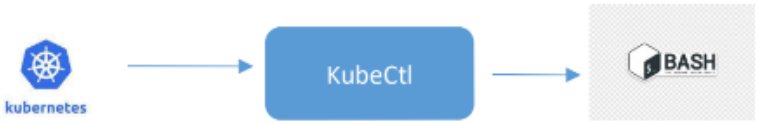
Failure Alert and Self-Heal

Applications conventionally throw specific error codes into the container/server logs in the event of transactional failures. Additionally, Azure Kubernetes Servers also logs system unavailability error codes along with Azure Application Insights has out-of-the-box "smart detection" feature to flag downtime and critical failures at server/hardware/network level.
Alerts can be established by feeding Kusto Queries (KQL) results into Azure Alert utilities as shown in below example, where it lists the problematic POD instances for a given error.

```
Kusto Query Sample
traces
| where cloud_roleName has "<your micro-service name>"
| where message has "<your Error code>"
| summarize count() by bin(timestamp, 30m), cloud_RoleInstance
```

Above failure diagnosis can be centrally orchestrated by the most powerful "KubeCtl" utility as shown in Figure-3.

Figure-3



The basic pseudo code in Kubectl can be designed as below where it kills the POD instance that throws error.

```
While traverse through all the POD-A instances for a given Micro-service {
do {
if { POD-A instance throws ERRCODE-1 }
restart all the POD-A instances
else  if { POD-A instance throws ERRCODE-2 }
        kill this particular POD-A instance
else if { POD-A instance throws ERRCODE-3 }
        do something else
else if { POD-A instance throws ERRCODE-4 }
        do a rolling restart of POD-A
else if { POD instance throws ERRCODE-5 }
        do a rolling restart of POD-B and then rolling restart of POD-A
}
}
```

Below sample code assumes the kubernetes PODs are configured as horizontal auto-scaling to maintain minimum number of POD instances. Once the POD instance is killed, a new instance gets created on it's own due to auto manifestation as defined in horizontal auto-scale configuration.

KubeCtl Code Sample for self-healing

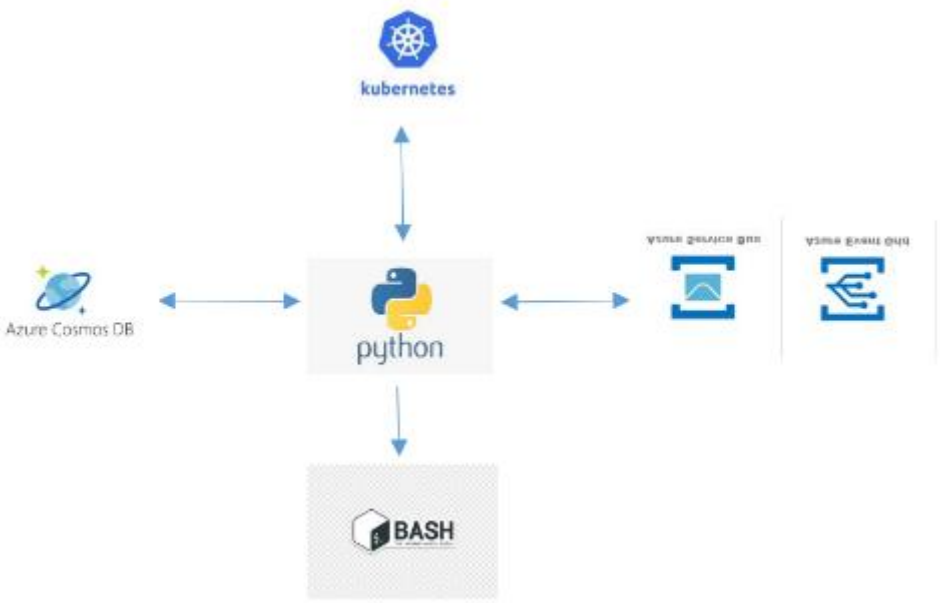
```
Kubectl get po -n <namespace> | grep <microservice name> | awk {'print $1'} > podlist
mapfile -t pod_names < podlist
for pod in "${pod_names[@]}"; do
  container="$kubectl get pods $pod -o jsonpath='{.spec.containers[*].name}' -n <namespace> | awk -F'
' '{print $2}' | uniq -u)"
  echo $container >> container.log
  kubectl logs $pod -c $container -n <namespace> | grep <ERRORCODE string>" > $pod.log
  if [ -s $pod.log ]; then
     kubectl delete po $pod -n <namespace>
     mailx -s "$pod deleted: For <ERRORCODE> <Your eMail here> < $pod.log
  fi
  done
```

KPI and Control Reporting

Python has abundance of libraries that can connect with any system that are deployed in recent age. In this model, the python utilities connect with Azure Kubernetes clusters, connect to Cosmos for data feed and also look at message counts and dead letters that fall in Azure Service Bus. While figure-4 depicts a few Azure components, they can be extended further to much more such as Azure Databricks or API Manager or Azure SQL DB and many more.

Shell scripts or Bash scripts can be then designed to invoke python programs and subsequently scheduled CRON jobs to periodically report the data points to required email distribution lists. The data points can further be enhanced programmatically to produce an executive level report for producing summarized tables.

Figure-4



Below python code sample fetches data from Azure Cosmos database collection and print into console.

A separate shell script can be written to call below python script for data manipulation, transformation, summarization and emailing to specific user groups for reporting.

Python Code Sample

```
...
import json
from azure.cosmos import CosmosClient
...
def establish_cosmos_connection():
   client = CosmosClient(url, credential=key)
   database = client.get_database_client(database_name)
   return database.get_container_client(container_name)
...
def get_data():
   cosmos_query = '<your Cosmos Query here>'
   container = establish_cosmos_connection()
   container_list = list(container.query_items(query=cosmos_query, enable_cross_partition_query=True))
   for item in container_list:
      data = json.dumps(item, indent=True)
      data = json.loads(data)
      print(data.get('value'))
if __name__ == '__main__':
   get_data()
```

RESULTS AND DISCUSSION

This research has taken two Transformation Programs (TP) as subject of study where, TP-B is 10 times larger than TP-A. For assessing a comparative analysis, TP-B is enabled automation of IT Operations while TP-A continued supporting in a conventional way.

	TP-A	TP-B
Component Size	7 Units	80 Units
IT Operations Effort (Persons)Without Automating Project B	40	315
IT Operations Effort (Persons) After Automating Project B	40	35
Customer Complaints	High Escalations	Negligible
Ops Engineer Job Satisfaction	Very Poor	Excellent
Major Incidents Reported (in a month)	12	0
No of Command Center calls (in a month)	18	2

It was observed for period of 8 months after automating mundane and routine tasks which gave a dramatic results – not only a substantial reduction of team size and cost, but the program saw psychological changes in engineers in terms of job satisfaction and attrition. Even the program leadership witnessed significant decline in customer escalations and Command Center calls.

On top of that, since the operations support could focus on mission critical problem incidents, the restoration delay is seen reduced substantially from several days to few hours.

Upon empowering the operations support engineers with latest tech skills such as KubeCtl, Python and shell scripting, the team developed many utilities on their own during their free time which further reduced the SLA (Service Level Agreement) delays.

Proactive monitoring helped team flag issues up front before they came to the notice of Business stakeholders. Timely engagement of third party vendors and emphasizing on issue avoidance in future also added to the advantages of Operations automation.

CONCLUSION

Having witnessed noticeable and tangible outcome after the research work, it is concluded that automation of IT Operations brings following benefits to organizations.

- Substantial Operational Cost reduction by about 70-80%
- Reduction in Customer Complaints

- Job Satisfaction and Behavioral improvements in Operations Engineers
- Reduction in occurrence of incidents
- Reduction in attrition
- Reduction in Command Center calls, thereby reduced governance Cost
- Increased focus on mission critical failures
- Empower Operations team for more automation of mundane tasks

RECOMMENDATION

Since Operational Cost consideration has a detrimental effect on deliberating new Program initiatives and transformational programs, it's advisable to embrace recommended automation methods during and after completion of build phases.

Here are few key considerations that will effectively improve the IT Operations with reduced ongoing cost on a longer run.

- Make frequently handled operating procedures as potential candidates for automation
- Consider a separate budget for operations automation during or after build
- Include below functions during the Requirement Phase and/or User Story Elaboration
    - KPI Reporting
    - Alert Mechanism
    - Auto-Monitoring
    - Self-Heal
    - Creation of Business Metrics dashboard
- Post Build phase, ring-fence key Devops personnel for Ops Automation activities until all the benefits (*discussed in Conclusion*) are realized

REFERENCES

1. CapEx vs OpEx: Captial Expenditures & Operational Expenses explained - https://www.bmc.com/blogs/capex-vs-opex/
2. ABC Operating expenses - https://www.macrotrends.net/stocks/charts/AMZN/amazon/operating-expenses
3. XYZ Operating Expenses - https://www.macrotrends.net/stocks/charts/BBY/best-buy/operating-expenses
4. 5 IT Operations Cost Traps and How to Avoid Them - https://www.infoq.com/articles/operations-traps-avoid/
5. A process-oriented perspective of IS success: Examining the impact of IS on operational cost - Omega Volume 34, Issue 5, October 2006
6. Kubernetes Kusto Query Language (KQL) - https://learn.microsoft.com/en-us/azure/data-explorer/kusto/query/
7. Kubernetes Control Cheat Seat - https://kubernetes.io/docs/reference/kubectl/cheatsheet/