On the Public Cloud Deployment of Cloud-Native Mobile Core Systems

Yuto Takano[1], Andrew E. Ferguson[1], Mahesh K. Marina[1]

[1]The University of Edinburgh

THE UNIVERSITY of EDINBURGH informatics

Overview

Leveraging the public cloud for the Core Network (CN) remains uncommon amongst mobile network operators, despite the economical and operational benefits. The drive to "cloud-native", through advancements in mobile core architecture research and standard evolution have made the case stronger. We believe that there are four types of challenges inhibiting this adoption: **(1)** jurisdictional regulations, **(2)** 3GPP & SLA compliance, **(3)** Cost Transparency, and **(4)** Safety and Control. In addition, all of the aforementioned challenges apart from (1) require in-depth observability, which would be hard to achieve without direct control of the hardware. While some past works have analyzed limited aspects of these challenges, our findings are the first to provide a holistic analysis of the public cloud in comparison to on-site mobile core deployments. Our findings show the technical and economical feasibility, and lay the ground for further adoption of the public cloud.

Challenges

Jurisdiction
Data protection in sovereign clouds

(not covered in our work)

Costs
Migration cost
Service cost
Personnel cost

Safety / Control
Debugging
Privileged monitoring
Software isolation

3GPP / SLA
Latency / Reliability
Protocol compliance

Observability

Evaluation Methodology

1. Representative Public Cloud: **AWS**
2. Representative CN Software: **CoreKube on Kubernetes**
   Chosen for being cloud-native, modular, open-source, thus representative of the future of CN software.

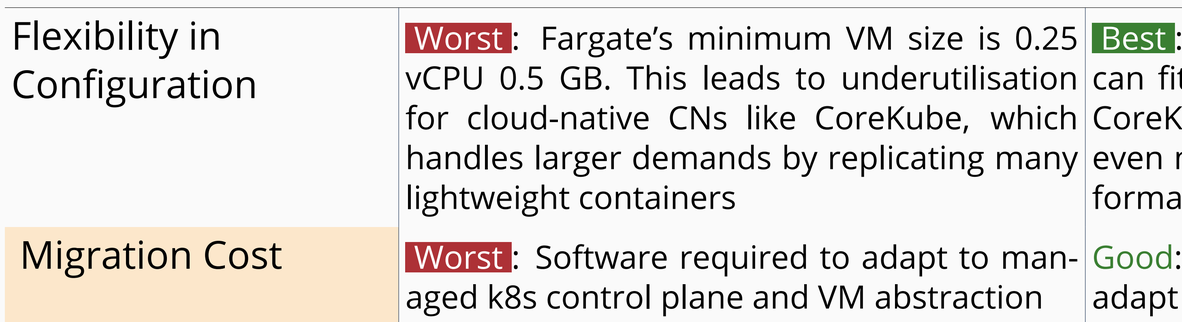
Deployment Methods

1. **Elastic Kubernetes Service (EKS)** with **Fargate** instances
   EKS is AWS's managed Kubernetes control-plane. Fargate is AWS's compute engine technology for cloud-native software, which runs each container in its own ephemeral micro VM. The equivalent on GCP and Azure are 'Cloud Run' and 'Container Apps'. EKS costs $0.1/hour, and Fargate costs $0.0133/vCPU/hour for compute and $0.00147/GB/hour for memory.
2. **EKS** with **Elastic Compute Cloud (EC2)** instances
   EKS can be configured to use standard EC2 VMs for container scheduling too. EC2 costs vary largely, but for CPU-bound software like the CN, machines like `t3.medium` at $0.0418 for 2 vCPU and 4 GB memory is enough.
3. **Self-managed Kubernetes** on **EC2** instances.
   One can also host their own Kubernetes control-plane within a EC2 VM too, and manage its configuration entirely themselves.

Qualitative Evaluation

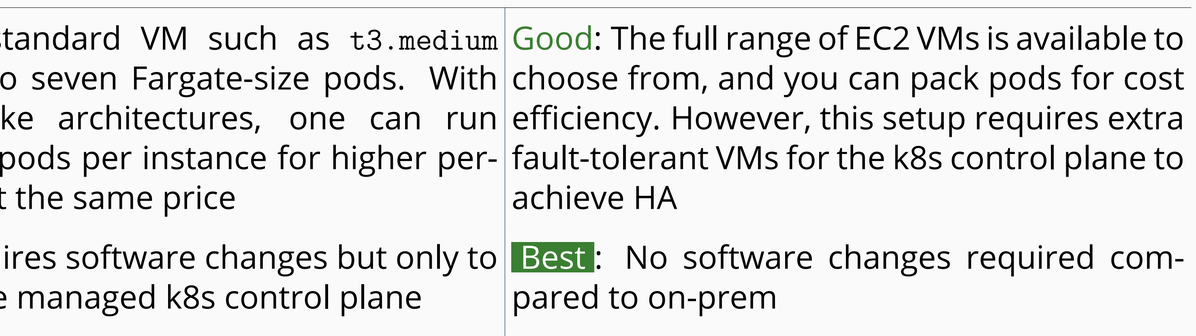
	EKS with Fargate (unsuited)	EKS with EC2 (feasible)	Self-hosted K8S on EC2 (suboptimal)
Flexibility in Configuration	Worst: Fargate's minimum VM size is 0.25 vCPU 0.5 GB. This leads to underutilisation for cloud-native CNs like CoreKube, which handles larger demands by replicating many lightweight containers	Best: A standard VM such as `t3.medium` can fit six to seven Fargate-size pods. With CoreKube-like architectures, one can run even more pods per instance for higher performance at the same price	Good: The full range of EC2 VMs is available to choose from, and you can pack pods for cost efficiency. However, this setup requires extra fault-tolerant VMs for the k8s control plane to achieve HA
Migration Cost	Worst: Software required to adapt to managed k8s control plane and VM abstraction	Good: Requires software changes but only to adapt to the managed k8s control plane	Best: No software changes required compared to on-prem
Service Cost	Good: EKS cost, plus fixed pricing for Fargate compute and memory	Best: EKS cost, plus EC2 VM cost. EC2 Spots achieve better price-per-pod than Fargate	Good: Only EC2 VM cost. But high availability control-plane puts costs about equal as EKS
Personnel Cost	Best: Everything is managed by AWS, little maintenance manpower required	Good; Operators only need to manage workload nodes	Worst: Operators need knowledge and skills to manage control plane and workload nodes
Debug Privileges	Worst: Without root access on the micro VMs hosting Fargate pods, it is very difficult to trace syscalls or debug issues	Best: Privileged access on VMs for control, while removing the noise that is the k8s control plane	Good: Control over everything, which can lead to more unnecessary sources of error
Software Isolation	Best: Micro-VMs isolate each container	Bad: Isolation is entirely up the operator	Bad: Isolation is entirely up to the operator
Protocol Compliance	Worst: Requires a proxy since Fargate does not support SCTP port 38412 (NGAP)	Best: Ports work as-is as long as you allow it in the firewall	Best: Ports work as-is as long as you allow it in the firewall

Cost Efficiency Validation



The compute (vCPU) usage per worker pod under a load of 100 connected UEs shows that Fargate instances are not efficient in their resource use.
Note that the EC2 results apply for both EKS and Self-Managed control plane deployments.

SLA Conformance Validation



The control-plane message processing latency shows Fargate as being unacceptably slow and inconsistent w.r.t the allowed end-to-end latency.
Note that the EC2 results apply for both EKS and Self-Managed control plane deployments.

Future Work

1. **Observability Solution.** Underpinning all the challenges featured in our work is 'Observability', the ability to monitor the state of each and automatically or manually act. Our future work is to propose a software solution to improve observability in cloud-native CN software specifically for public cloud use cases.
2. **Dataplane Feasibility.** The dataplane is subject to stricter latency and reliability requirements than the control plane. Further work is required to determine the feasibility of the public cloud for the dataplane, and otherwise design architectures to overcome any challenges.

Contact