SMARTER:
Experiences with Cloud Native on the Edge

Alexandre Ferreira
Arm Research

Eric Van Hensbergen
Arm Research

Chris Adeniyi-Jones
Arm Research

Edmund Grimely-Evans
Arm Research

Josh Minor
Arm Research

Mark Nutter
Arm Research

Luis E. Peña
Arm Research

Kanak Agarwal
Arm Research

Jon Hermes
Arm Research

Abstract

The decreasing cost and power consumption of intelligent, interconnected, and interactive devices at the edge of the internet are creating massive opportunities to instrument our cities, factories, farms, and environment to improve efficiency, safety and productivity. Developing, debugging, deploying and securing software for the estimated trillion connected devices present substantial challenges. As part of the SMARTER (Secure Municipal, Agricultural, Rural, and Telco Edge Research) project, Arm has been exploring the use of cloud-native technology and methodologies in edge environments to evaluate their effectiveness at addressing these problems at scale.

1 Introduction

As the number of internet of things (IoT) endpoints grow, it will be less practical to send all data produced to the cloud for real-time decision making and/or control functions. Edge computing has emerged as a dominant industry trend due to practical constraints such as latency, bandwidth, privacy, security, and robustness to intermittent connectivity. For the purposes of this paper, we define edge computing as operations on IoT device endpoints and the gateways [11] that connect those devices to the broader internet. In many cases, edge computing of this form will reside on constrained hardware [20], but, increasingly, even constrained hardware is interconnected, intelligent, and capable of doing more than one task. The industry is starting to converge on a model where this general purpose compute capability is leveraged by performing filtering, analytics, and aggregation closer to the source of the data. These devices range in compute power from low-end systems like the Raspberry Pi [9], to systems with integrated ML acceleration such as Google's Coral-Dev [5], to embedded cores coupled with GPUs like Nvidia's Xavier [8].

At the same time, the management and orchestration of the applications become daunting challenges as applications get spread over a highly distributed and heterogeneous infrastructure. In this paper, we introduce the SMARTER project, which attempts to establish an edge computing platform that brings cloud-native application development, deployment and management technologies to the intelligent IoT edge. In the next section, we will go into more detail on differences between the edge and cloud environments. Section 3 will describe the basic design principles and architecture of our solution. Our approach to managing application network connectivity is described in section 4, with section 5 describing how we manage access to other peripherals such as sensors. Section 6 will describe our approach to monitoring and debug of applications. Finally, we conclude by summarizing related work in section 7 and discussing key issues and trade-offs in section 8.

2 Edge versus Cloud

Beyond hardware constraints, there are a number of key differences between edge and cloud as operational environments. Edge nodes and devices are purpose-built with different cost constraints, resulting in many different configurations deployed over multiple generations of underlying hardware components. Nodes will differ in hardware resources such as CPU architecture, micro-architecture, core count, memory, storage, connectivity (latency and bandwidth), peripherals, and accelerators. Additionally, edge nodes and gateways are more likely to have dynamic frequency scaling (either because of battery conservation or thermal throttling). This high degree of hardware heterogeneity has implications on deployment, where multiple versions of an application may be required to support device differences.

The edge has a higher data storage and transmission cost compared to the data center. Few edge devices are likely to have high-bandwidth network connections, and transferring data to and from thousands of edge gateways will be expensive. Virtual machine and container images magnify the data movement cost, amounting to close to a complete distribution download per application due to packaging. While layered

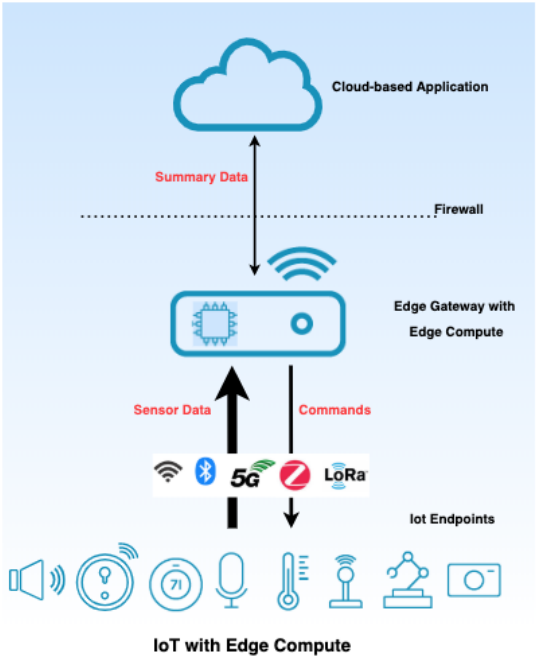

Figure 1: Edge Gateway Use Case

container images are intended to reduce this overhead, third-party packaging of applications makes underlying layer reuse unlikely. For example, we had a prototype health-care application which used a total of 17 Docker images, occupying approximately 2.3GB of storage. Deploying this application to thousands of nodes over metered cellular networking would not have been practical.

The cloud-native model, characterized by continuous integration and deployment, has become the preferred development and deployment model for modern applications. It leverages the uniform and seemingly unlimited resources of the cloud to speed application life cycle while improving test coverage and operational efficiency. The aforementioned heterogeneous nature of edge deployment makes this somewhat trickier to manage, although layering emulation environments and target-specific test beds can address many of the differences between cloud and edge application life cycle. Further, recent advances in multi-architecture targeting of container build environments [12] can improve application reach across different classes of devices. Edge deployment has its own quirks (such as bandwidth conservation, lack of location independence and scale-out limitations), but many of the same cloud-native deployment benefits such as canary updates, rolling updates, and fast rollback on failure are still possible.

3 Architecture

Applications in edge environments will have components that run either on the cloud, the intelligent gateway, or the embedded device. SMARTER is primarily responsible for the pieces running on the intelligent gateway. The gateway is intended to support multiple endpoint embedded devices, multiple applications, and, potentially, multiple users or tenants. As such, we use containerization as a fundamental element both to provide user isolation and to meet application dependencies. Containerization also enables some the benefits around cloud-native development and deployment environments mentioned in the previous section.

We leverage Kubernetes [4] (k8s) as the control plane [2], composed of a k8s master running in the cloud with each gateway node running the kubelet node agent with Docker [16] as the container runtime engine. Instead of a cluster of homogeneous nodes able to run any service as is typical in a cloud k8s environment, each gateway node is treated as an independent entity. Gateway nodes are not able to communicate directly with each other except through interactions with cloud-based resources. All communication originates from the gateway node to the cloud, reducing the potential attack surface for security intrusions on the endpoints. The nature of the k8s protocol provides an open-standard, authenticated and encrypted interface, and the declarative model employed by k8s provisioning is resilient to intermittent connectivity and node restart.

Our design philosophy for SMARTER was based on a few principles. The first principle, which we borrow from k8s itself, is that all functionality should run as containers and be remotely deployable. This requirement enables easy upgrades from within the platform and the selective deployment of only the containers that are needed to support the applications on that node. The second principle is to enhance k8s functionality instead of replacing it; reuse existing k8s features, if possible, instead of creating new ones. The third principle is simplicity: do not create or require concepts if they do not apply to IoT: the edge is different from the cloud and some differences are inevitable.

Three components from k8s are used to create the low level SMARTER application deployment model: *DaemonSets*, *Labels* and *LabelSelectors*. An application in SMARTER is composed of a set of containers. Depending on the environment, some containers may not always be required or two different types may be mutually exclusive (accelerated and not accelerated for example). Each container is described by a single DaemonSet, where the DaemonSet contains a template for the container including the location of the image, resource requirements, security and a LabelSelector for that container. The DaemonSet also provides features like rolling updates, restarts in case of container failures, and statistics collection.

Labels are key/value pairs associated with many objects in k8s and can also be associated with nodes. They can be

created by the user or automatically created. Kubelet, for instance, creates labels that contain the node's architecture and operating system. The LabelSelector is a set of logical operations that, when applied to the Labels on a object, returns true or false. In the case of DaemonSets, the LabelSelector deploys a container if the result is true for the Labels on a given node. These three concepts allow SMARTER to decouple the application definition (creation of the templates, versioning) and management (statistics from the node deployment and management) from deployment, since the deployment of the application in a node can be gated by the use of one or more Labels. As more nodes are added to a cluster, they will automatically pick up the appropriate application template.

Security is a very important aspect of edge computing [15]. SMARTER splits security into infrastructure security and application security. Infrastructure security preserves the integrity of the computing platform and the communications with the managing platform in the cloud. It encompasses all the aspects of hardware security, firmware and OS integrity up to node management (kubelet and container runtime). SMARTER assumes that this security is provided by the underlining platform. However, other projects at Arm Research have been focusing on providing standard, high-security foundations for IoT devices and gateways. Secure node identity and attestation to the cloud are essential features to enable resilient and security aware deployments of IoT infrastructure. This is also part of the path for establishing data provenance, which is emerging as a fundamental requirement for applications that can have large social or financial impact.

Application security encompasses provenance, isolation, identity, resource management, privacy, identification, authorization, configuration and secret management. SMARTER provides capabilities that address some of these requirements in a limited way, but there still are a lot of opportunities for further improvement. Container isolation is not considered secure enough for certain use cases; resource partitioning is not provided for some resources (*e.g.* memory bandwidth and cache partitioning). GPU and other accelerators provide interesting challenges: since they were not designed to be shared or partitioned, they exhibit limitations when used in an infrastructure that is oriented towards multi-tenancy of applications. SMARTER is intended to support mixed secure environments and to scale the security requirements according to the use case as well as regional policy.

4 Connectivity

SMARTER sees each gateway as independent. By not requiring each gateway to be accessible to each other, SMARTER can provide a network configuration that is independent of the external network configuration. The only requirement is that the gateway is able to connect to the k8s master on the cloud. This approach facilitates AI-at-edge and edge-to-cloud analytics. Each gateway can produce local actionable insights,

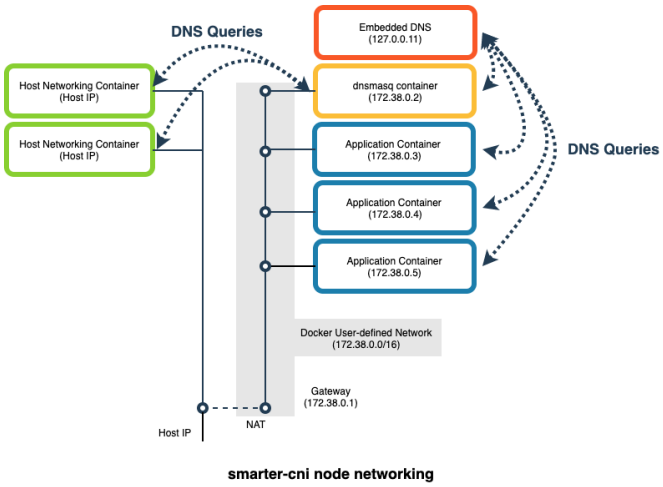

Figure 2: SMARTER-CNI network

or share metadata with the cloud for larger multi node applications [17]. In order for applications running on the same gateway to be able to communicate with each other, we implement a custom edge container network interface (CNI) [18]. The CNI is used to manage the allocation and deallocation of network resources to containers as they are created and deleted.

The design of the edge CNI was dictated by a few high-level requirements. The internal communication of the gateway should not be affected by the external connectivity. The applications in a gateway are not directly visible to other gateways, so discovery (DNS naming policies primarily) can be restricted to a single gateway, keeping the internal configuration of the gateway not externally visible. The networking configuration for a gateway using an edge CNI can be viewed in two ways:

External Each physical network interface (Ethernet, Wi-Fi, cellular, etc.) on the gateway is managed by the network that it is connected to. The system makes no assumptions about the IP addresses or DNS names provided. It is expected that at least one interface provides access to the Internet so that the node can connect to the cloud-based k8s master. We assume that the external interfaces of the node will be externally configured and secured by boot-time platform configuration in the firmware.

Internal SMARTER uses a Docker user-defined network to which all applications are connected via virtual interfaces (only applications that use host networking do not have a virtual interface). Each deployed application has an interface allocated from this user-defined network, receiving an allocated address from within the range of the internal subnet.

A local DNS is used as the service discovery element on our CNI and removes the need of using k8s Service objects [3].

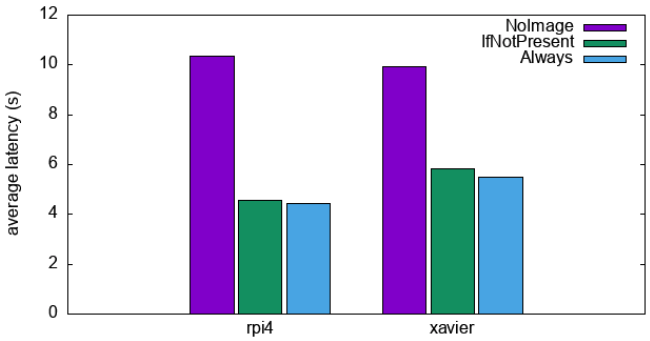

Figure 3: Application Deployment Latency

Docker provides an automatically enabled, embedded DNS resolver (`127.0.0.11`) for user-defined networks. When an application is started on the gateway, our CNI captures the k8s pod name and creates a DNS record in the embedded DNS server. It is this mechanism that enables applications running on the same node to discover each other's IP addresses via DNS lookups using their application names. Each gateway also runs a containerized `dnsmasq` connected to the user-defined network with a static address. Applications using host networking are configured to look up DNS entries via this `dnsmasq` and can, therefore, also discover IP addresses via DNS lookups of application names (which would not normally be possible as host-networked pods cannot access the embedded DNS resolver directly).

With our current connectivity model, application deployment involves a party requesting the creation or deletion of k8s resources via the cluster master's API. Once the action is registered, the updated state is broadcast to the entire cluster, where appropriate actions are taken by each node's kubelet to achieve the desired state (*e.g.* by creating or destroying pods). Because the k8s master resides in the cloud, not co-located with the edge gateways, there are concerns over system response latency in these widely-distributed systems. To investigate response latency, we measured the time taken from a user on a home network requesting to run a new application on an edge node to when the application is deployed and has transmitted data to the cloud. In our test, the k8s master ran on an AWS EC2 instance with 2 vCPUs and 4GB of RAM.

The test results are broken into 3 different k8s application deployment scenarios: one where the Docker image of the target application is not available to the node and it must be pulled (`NoImage`), one where the image is present and there is no attempt made to pull the image (`IfNotPresent`), and another where the image is present but there still is an attempt made to pull the image (`Always`). We observed that, for a roughly 50MB application image, both the Raspberry Pi 4 and Nvidia Xavier tested take around 10 seconds when having to pull the image from a remote image repository. On the other hand, it takes between 4 and 6 seconds to bring the application up if the application image is resident on the device (`IfNotPresent` and `Always` pull policies). As the size of the application image grows, the time spent pulling new images on the edge device will grow larger, however this is generally is only a one time cost.

5 Device Manager

Being designed for the cloud, k8s only manages CPU, memory, storage and network as virtual resources. IoT systems have a much broader range of peripherals, built for monitoring and interacting with the physical environment, with complexity ranging from a simple thermostat to very intricate industrial process control (*e.g.* chemical plants). This is accomplished via a variety of sensors and actuators, which are either connected to the gateway physically (*e.g.*GPIO, USB, I2C, SPI) or wirelessly (*e.g.* Bluetooth, Wi-Fi). Regardless of the way the sensor or actuator is connected, it is typically represented as a device which applications interact with through the operating system.

Even though container runtimes allow direct access to device drivers, containers running under k8s are not expected to do so. Existing IoT applications directly interface to sensors through the Linux kernel via device drivers (for serial ports and USB devices), synthetic file systems (like `sysfs` or `procfs`), multimedia services (such as V4L or PulseAudio), or through one of the system bus methodologies. Controlled access to these devices is essential to secure a container-based IoT solution. We built the SMARTER device-manager to govern direct access and sharing to devices on the host in a secure way.

The SMARTER device-manager leverages the device plugin API [6] provided by k8s. This API allows the kubelet to be informed of which resources are available on the node and to execute the necessary operations to make that resource usable by a container. In its current incarnation, the SMARTER device-manager scans the `/dev/` directory of the host and uses a set of rules to determine which devices to advertise as resources and how those resources should be accounted. Some resources are not shareable, so only a single container can have access to each of them, others can be shared, but may have a limitation on how many simultaneous applications are allowed to access it concurrently.

The SMARTER device-manager is a container by itself, and it is deployed by k8s in the same fashion as the applications, allowing easy upgrade and configuration management. Of course, any devices would already need to be supported by the kernel running on the device, unless using user-space drivers.

Some devices, particularly those normally associated with client devices, do not provide strong isolation between devices. For example, under Linux, access can be controlled to a Bluetooth wireless controller, but does not provide fine grained control over the devices it connects. So far, we have managed this with separate DaemonSet containers managing

fine grained access to such buses and exposing interfaces over the internal private network. This approach works for multiplexing access to devices which are not capable of virtualizing themselves such as microphones or cameras, but comes with some performance overhead that must be considered as part of the trade-offs of the system.

6 Monitoring and Debug

There are many pre-existing solutions for logging and performance monitoring of cloud-native applications. The limitations of the edge impact the applicability of many of these solutions to either being used in a limited form or requiring certain modifications. For example, many methods rely on the cloud being able to contact and pull data from the node, but communication on the edge must always push data from the gateway to the cloud. Similarly, the aforementioned constraints on bandwidth and storage may suggest different configurations and optimizations in a cloud environment. In our deployments, we have focused on anomaly detection on the edge prior to pushing telemetry to the cloud versus a constant stream of metrics and logs.

While cloud-native development often precludes physical access to the actual system the application is deployed to, developers are usually able to `ssh` directly to development, test, and production systems. SMARTER's expected deployment model prevents direct connection to edge nodes, and we need to accommodate situations which may require remote field debugging of the application

A special application, a privileged *debug container*, can be deployed to the edge device. This container connects to a *comms* service deployed on an accessible cloud server. The developer can then indirectly access the edge device through the cloud service. The result is to provide the developer with something that closely resembles `ssh` or `docker exec` to the edge device, but that works despite firewalls not allowing a direct connection.

It seems to be easier and more secure to use three `ssh` connections: one from the edge device to the `comms` service, one from the developer's machine to the `comms` service, and a third one that goes from the developer's machine through a tunnel created by the other two connections all the way to the edge device. The use of `ssh` and public key authentication allows a robust security model: all the communication is encrypted and access can be given per node and per developer.

The `comms` service may be shared by multiple developers and multiple connections and the only requirement is that is has to be accessible by the node and developer. The current version has a REST API for setting up accounts and uploading SSH keys.

There are facilities to make an account on the `comms` service expire at a particular time in the future, and the developer can run a particular command remotely by specifying it on the command line versus always using an interactive session. Unfortunately, the current debug and monitoring solutions are not multi-tenant-safe as they give access to the entire gateway and not to a specific application. This is something we hope to address in future versions of the infrastructure.

7 Related Work

The current status of the IoT edge computing oriented frameworks can be split into three main classes:

k8s-based frameworks that enhance k8s with additional capabilities either by creating architectural blueprints or by adding additional components.

k8s-derived frameworks that use either portions of k8s, provide a compatible API either at server level (k8s API server) or slave (Kubelet API).

non-k8s frameworks that utilize other abstractions (Docker compose) or are proprietary.

Akraino [24] falls into the k8s-based class and provides blueprints for specific uses cases, the common thread is supporting network functions in a telecommunications environment. The closest to SMARTER is the ELIoT blueprint [21] which has similar requirements, usage model and components and a k8s deployment model. ELIoT supports containers and uses k8s, split into k8s master on the cloud and and k8s slave (kubelet) on the remote nodes.

KubeEdge [23] is in the k8s-derived class. It is an open source project that uses the k8s API but replaces the cloud and edge portions with derived code and communicates between them using a proprietary protocol. It is prescriptive in the addressing of the device management for industrial IoT use cases (specifically supporting ModBUS and Bluetooth) but does not provide a generic interface to Linux device drivers, and its interface is very oriented to set/get/status devices. The CNI and networking is oriented towards using the internal communication channels of KubeEdge (MQTT and go channels) where our CNI is oriented towards keeping the flexibility of cloud-style microservices communication - not enforcing a single model (like MQTT).

AWS IoT Greengrass [14] provides support for containers at the edge but uses a Docker compose model and it is not k8s API compatible. It is not open-source and uses a proprietary API for orchestration. It is designed to manage embedded AWS IoT Greengrass devices. Docker compose provide similar capabilities as our CNI and device management without explicit management (resource management for devices as an example). Microsoft Azure IoT [13] provides support for containers at the edge and a k8s API at the cloud. It uses the virtual kubelet concept so it can be managed by a k8s master. The solution is partially open source.

K3s [22] is a k8s derived framework oriented towards efficiency and ease of installation. It has similar capabilities as

the standard k8s and similar limitations for IoT, the enhancements provided by SMARTER for CNI and device management are compatible and can be applied to K3s.

Resources

The cloud-side management components of the SMARTER project are being productized by Arm as part of its Pelion Edge Compute service [19]. All SMARTER client-side components are available as open-source, with a K3s-based example provided on our GitLab site at `https://gitlab.com/arm-research/smarter` and can be used with a generic k8s master.

8 Discussion

Kubernetes provides a rich set of abstractions supporting various aspects of application deployment. Our restricted use of these abstractions to only accommodate those which make sense for edge deployments (*e.g* DaemonSets) allowed us to use vanilla implementations of k8s, but came at the cost of some of the benefits of the more advanced features. It also prevents us from easily using tools and standard recipes for more complex application deployments such as Helm [7]. An alternative is to provide different semantics for these abstractions in edge deployments. While our edge deployment model is somewhat flat, it is becoming apparent that having small clusters at the edge is desirable for reliability or alternatively having hierarchical offload mechanisms and deployments may be beneficial – these may suggest a different set of design decisions. It is unclear to us the best trade-off here or what the best approach to augmenting k8s for different deployment models is so we would value the opinion of others in the community.

We use containers as a base application unit of isolation and deployment. In our experience, many applications at the edge are simple ingest, minimal processing, and up-link to the cloud. These tend to be written as simple scripts that do not require complex dependencies in terms of support software. The high overhead of containers for such simple scripts led us to look at function-as-a-service (FaaS), such as AWS IoT Greengrass [14]. However, FaaS models share many of the similarities and differences that cloud-native and edge-native deployments share. We have started implementing FaaS services on top of k8s and have been looking at FaaS runtime isolation models using WebAssembly [10], but are interested in others' experience with FaaS models on the Edge.

Microcontroller-based devices (sensors) are not directly managed by SMARTER because they provide very limited or no support for dynamic deployable code and for the main requirement of updating firmware and security credentials. These devices are part of the data plane of SMARTER serving as data sources and control elements with SMARTER providing larger computing capabilities (ML), data aggregation, databases, gateway services and other functions. A number of companies including Arm [1], Microsoft [13], Amazon [14], and others provide commercial or open source solutions for managing microcontroller-based devices that are compatible with SMARTER. Another possibility, if we go further down the FaaS path, is to find a way to extend FaaS to microcontroller devices from the gateway.

Most existing models of edge compute focus on an appliance view of the resource, where it is dedicated to a single purpose (or set of purposes), essentially predetermined and loaded onto the device at manufacture time. In several of the use cases and market segments we have discussed with partners, it is actually desirable to build edge compute as distributed infrastructure and allow multiple applications and even multiple tenants utilize the resources. Our use of k8s facilitated multiple applications and even dynamic deployment, but it fell short of actually providing a true multi-tenant infrastructure. Similarly, given the distributed nature of edge compute, one could easily envision a distributed environment with multiple operators of edge infrastructure which could publish portions of their resources for shared use by multiple consumers. Providing this capability would require a broker which would facilitate the publishing, discovery, and subscription of resources. It would require better isolation primitives and better monitoring of system resources to facilitate fine grained metering. We are interested in hearing about resource and data brokerage infrastructures being developed by others.

References

[1] Arm. Arm Pelion. https://blog.pelion.com/post/gateway-to-unified-iot.

[2] Sujoy Basu, Sven Graupner, Jim Pruyne, and Sharad Singhal. Control plane integration for cloud services. In *Proceedings of the 11th International Middleware Conference Industrial Track*, Middleware Industrial Track '10, page 29–34, New York, NY, USA, 2010. Association for Computing Machinery.

[3] Eric A Brewer. Kubernetes and the path to cloud native. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 167–167, 2015.

[4] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Queue*, 14(1):70–93, 2016.

[5] Stephen Cass. Taking AI to the edge: Google's TPU now comes in a maker-friendly package. *IEEE Spectrum*, 56(5):16–17, 2019.

[6] CNCF. Kubernetes device plugin. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/.

[7] Jessica Deen. DevOps with Kubernetes and Helm. 2018.

[8] Michael Ditty, Ashish Karandikar, and David Reed. Nvidia's Xavier SoC. In *Hot Chips: A Symposium on High Performance Chips*, 2018.

[9] Sheikh Ferdoush and Xinrong Li. Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications. *Procedia Computer Science*, 34:103–110, 2014.

[10] Phani Kishore Gadepalli, Gregor Peach, Ludmila Cherkasova, Rob Aitken, and Gabriel Parmer. Challenges and opportunities for efficient serverless computing at the edge. *SRDS*, 2019.

[11] S. Guoqiang, C. Yanming, Z. Chao, and Z. Yanxu. Design and implementation of a smart IoT gateway. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 720–723, Aug 2013.

[12] Eric Van Hensbergen. Continuous cross-architecture integration with GitLab. https://community.arm.com/developer/research/b/articles/posts/continuous-cross-architecture-integration-with-gitlab.

[13] Scott Klein. *IoT Solutions in Microsoft's Azure IoT Suite*. Springer, 2017.

[14] Agus Kurniawan. *Learning AWS IoT: Effectively manage connected devices on the AWS cloud using services such as AWS Greengrass, AWS button, predictive analytics and machine learning*. Packt Publishing Ltd, 2018.

[15] Milosch Meriac. Security manifesto. https://pages.arm.com/iot-security-manifesto.html, 2017.

[16] Dirk Merkel. Docker: Lightweight Linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[17] Nvidia. Multi-camera large-scale intelligent video analytics with DeepStream SDK. https://devblogs.nvidia.com/multi-camera-large-scale-iva-deepstream-sdk/.

[18] Youngki Park, Hyunsik Yang, and Younghan Kim. Performance analysis of CNI (container networking interface) based container network. In *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 248–250. IEEE, 2018.

[19] Deepak Poornachandra. The gateway to unified IoT. https://blog.mbed.com/post/gateway-to-unified-iot, 2019.

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[21] Alessandro Sivieri, Luca Mottola, and Gianpaolo Cugola. Building internet of things software with ELIoT. *Computer Communications*, 89:141–153, 2016.

[22] Caroline Tarbett. Why K3s is the future of Kubernetes at the edge. https://rancher.com/blog/2019/why-k3s-is-the-future-of-k8s-at-the-edge, 2019.

[23] Ying Xiong, Yulin Sun, Li Xing, and Ying Huang. Extend cloud to edge with KubeEdge. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 373–377. IEEE, 2018.

[24] Wenhui Zhang, John Craig, Kandemir Mahmut, Yizheng Jiao, Robert P Eby, Deepak Kataria, David Plunkett, Robin Chen, Zhe Huang, Oliver Spatscheck, et al. Network slicing, personalized intrusion detection for internet of things gateway.