Containers and Clusters for Edge Cloud Architectures

Claus Pahl
Irish Centre for Cloud Computing and Commerce IC4, Ireland
University of Bozen-Bolzano, Italy
Motivation

- Cloud technology is moving
  - distribution across multi-clouds
  - inclusion of devices – IoT / edge cloud / fog computing

- Lightweight virtualisation
  - for smaller, virtualised devices to host application/platform services
  - Containerisation as a lightweight virtualisation solution.

- Containers relevant for Platform-as-a-Service (PaaS) clouds
  - application packaging and orchestration.
  - this can help to manage and orchestrate applications as containers

Agenda:
- review edge cloud requirements
- discuss suitability of container and cluster technology
Agenda

- Edge Cloud
- Virtualisation and Containers
- PaaS Clouds and Containers
- Clusters and Distribution in the Cloud
- Container-based Edge Cloud
- Edge Cloud Management
- Use Cases
Edge Cloud – Architectural Requirements

- **Challenges:**
  - Virtualisation and interoperable application packaging
  - Distributed delivery and orchestration of infrastructure and application services
Edge Cloud – Architectural Requirements

- Classify distributed clouds into three architectural models:
  - **Multi-datacentre clouds** with multiple, tightly coupled data centers under control of the same provider.
  - **Loosely coupled multi-service clouds** combine services from different cloud providers.
  - **Decentralized edge clouds** utilize edge resources to provide data / compute resources in a highly dispersed manner.

- Needs infrastructure and application services to be placed at source of data.
Edge Cloud – Architectural Requirements

- **Infrastructure Support:**
  - Location awareness, computation placement, local replication/recovery
  - Packaging, deployment, orchestration
  - Data transfer between virtualised resources
Edge Cloud – Architectural Requirements

- Development support for these architectures
  - supported through orchestration based on topology patterns + orchestration plans
  - reflecting common and reference architectures

- Application packaging through containerisation:
  - Containers to distribute service and applications to the edge
  - Docker has been used to do this

- Programmability:
  - Orchestration support through topology specification
    - TOSCA topology patterns
  - Service orchestration needs to cover whole life-cycle
    - deploy, patch, shutdown
  - Operations are mapped to cloud infrastructure management
    - TOSCA engine runs on top of edge cloud infrastructure
Virtualisation and Containers

- VM instances: full guest OS images – large files
- Space and time constraints
Containers

- Packages, self-contained, ready-to-deploy set of parts of applications
- In the form of binaries and libraries to run applications
Virtualisation and Containers

- Recent Linux distributions - Linux container project LXC
  - kernel mechanisms to isolate processes on shared operating system
  - Mechanisms: namespaces and cgroups

- Namespace isolation
  - allows groups of processes to be separated
  - different namespaces for process isolation, access to inter-process communication, mount-points, for isolating kernel and version identifiers

- cgroups (control groups)
  - manage and limit resource access for process groups
  - enables better isolation between isolated applications on a host
  - restricts containers in multi-tenant host environments
  - cgroups allow sharing hardware resources between containers
    - if required, setting up limits and constraints
Virtualisation and Containers

- **Boot process:**
  - traditional Linux boot: kernel mounts root FS as read-only, then switches rootfs volume to read-write mode
  - Docker mounts the rootfs as read-only, but instead of changing FS to read-write mode, it uses a union mount to add a writable file system on top

- **Mounting (union mount):**
  - allows multiple read-only FS to be stacked on top of each other
  - can create new images by building on top of base images
  - each of these FS layers is a separate image loaded by the container engine for execution.

- **Container:**
  - only the top layer (container) is writable
  - container can have state and is executable - directory for everything
Virtualisation and Containers

Container-based Application Architecture – Scenarios:

- **Container solution:**
  - Repositories
  - API: create, define, compose, distribution
  - Storage and network functions:
    - shared volumes, links for data transfer
PaaS Clouds and Containerisation

- **PaaS:**
  - Built farms
  - Routing layers
  - Schedulers to dispatch workloads
PaaS Clouds and Containerisation

- **Evolution of PaaS:**
  - **first PaaS generation:**
    - classical fixed proprietary platforms
    - such as Azure or Heroku.
  - **second PaaS generation:**
    - open-source solutions such as Cloud Foundry or OpenShift
    - allow users to run their own PaaS (on-premise or in the cloud)
    - already with a built-in support of containers.
      - Openshift moves from own container model to Docker model
      - Cloud Foundry does as well through its internal Diego solution
  - **third PaaS generation:**
    - Dawn, Deis, Flynn, Octohost and Tsuru,
    - built on Docker from scratch
    - deployable on own servers or on public IaaS clouds
    - Clustered, distributed architecture management
PaaS Clouds and Containerisation

- **Microservices architectural style**
  - developing a single application as a suite of small services
  - each running in its own process and lightweight communication

- **Microservices are**
  - independently deployable
  - supported by automated deployment and orchestration

- **They require**
  - ability to deploy often and independently at arbitrary schedules

- **Microservice dev/arch concerns are PaaS concerns**
  - Containerisation provides ideal mechanism for flexible deployment schedules and orchestration needs
  - particularly, if these are to be PaaS-provisioned
Clustering and Distribution

- Cluster architecture:
  - Multiple clusters in multiple clouds
Clustering and Distribution

Container clusters

- Features:
  - Failover
  - Load balancing
  - Scalability

- API:
  - Platform service mgmt
  - Lifecycle mgmt
  - Cluster head node
Clustering and Distribution

- Requirements for a lightweight virtualised cluster architecture:
  - Hosting containerised services
  - Providing secure communication between these services
  - Auto-scalability and load balancing support
  - Distributed and scalable service discovery and orchestration
  - Transfer/migration of service deployments between clusters

- Tools:
  - Mesos and Kubernetes
Clustering and Orchestration

- Cluster architecture:
  - Interoperable orchestration
Clustering and Distribution

- TOSCA supports a number of features:
  - interoperable description of application & infrastructure services
    - here implemented as containers hosted on nodes in an edge cloud,
  - relationships between parts of the service
    - here service compositions and links as relationships,
  - operational behaviour of the services in an orchestration plan
    - such as deploy, patch or shutdown
TOSCA for Container Orchestration

- **Needed:** a TOSCA-based modelling language
  - To describe the features of a container in abstract terms
  - to compose multiple containers to build an application
  - to orchestrate the deployment and management of multi-container applications in distributed clusters

- **Specifically:**
  - Need: manage applications over multiple and heterogeneous clouds,
  - Solution: services have to be described and orchestrated in a standardized fashion
TOSCA Service Templates for Containers

[ joint work with A. Brogi, J. Soldani @ University of Pisa ]
Docker Orchestration Example
Container-based Edge Cloud Deployment

- Cluster architecture for edge cloud scenarios:
  - Cloud deployment on resource-constrained devices
PaaS & Container Ecosystem

Strata of the Container Ecosystem

Layer 7: Workflow
- OpenShift
- DEIS

Layer 6: Orchestration
- Kubernetes
- Marathon

Layer 5: Scheduling
- Mesos
- Omega

Layer 4: Container Engine
- Docker
- Rocket
- OpenShift

Layer 3: Operating System
- Ubuntu
- Red Hat Linux
- CoreOS

Layer 2: Virtual Infrastructure
- vSphere
- Amazon Web Services
- EC2

Layer 1: Physical Infrastructure
- Raw compute
- Network
- Storage
Beyond PaaS: Devices for the Edge Cloud

- **Driver:**
  - Bring computation to the edge
  - Infrastructure + application services placed at source of data

- **Assumption:**
  - Resource-constrained devices
  - Capable of carrying out some remote calculations

- **Solution**
  - **Hardware:** Raspberry Pi (or similar)
  - **Software/application packaging:** Docker
  - **Cluster management:** Kubernetes
  - **Orchestration:** TOSCA
Raspberry Pi and Linux

- **Raspberry Pi 2 Model B** - second generation Raspberry Pi
  - 900MHz quad-core ARM Cortex-A7 CPU
  - 1GB RAM

  Replaced the original Raspberry Pi 1 Model B+ in Feb’15.

- **Processor: ARMv7 + 1GB**
  - can run the full range of ARM GNU/Linux distributions,
  - support of Raspbian - a free operating system based on Debian optimized for Raspberry Pi hardware
Docker on Raspberry Pi

From [https://blog.docker.com/2015/09/update-raspberry-pi-dockercon-challenge/]

- Describes demo running 500 Docker containers on a Raspberry Pi 2 device

- As of now (?), the current record stands at **2334 web servers** running in containers on a single Raspberry Pi 2.
Sample configuration

Resources:
- A RPi 2 has 1Gb of RAM (about 975 Mb available)
- Memory footprint of single web server outside a container is 0.3 Mb
- Can use ~700 Mb (2300 instances) for “real” processes,
- Leaves 300 Mb for the system and the Docker engine

Specs for set-up:
- Raspberry Pi 2 (4x core, 1 GByte memory)
- Docker 1.8.1 (stock version, without any optimisations)
- Linux: Debian Wheezy (HypriotOS) with Kernel 3.18.11 (used for DockerCon demo)
- Web server: Docker Image “hypriot/rpi-nano-httpd:minimal” (available on Docker Hub)
Raspberry Pi – Cloud and Cluster

University of Glasgow:
- [https://raspberrypicloud.wordpress.com/blog/](https://raspberrypicloud.wordpress.com/blog/)
- Glasgow Raspberry Pi Cloud (PiCloud)
  - scale model of data centre (DC) composed of clusters of Raspberry Pi
  - The Pi Cloud emulates all layers of a cloud stack
    - ranging from resource virtualisation to network behaviour

University of Bozen-Bolzano:
- The UniBZ Raspberry Pi cluster creates a small DC infrastructure
The University of Bozen-Bolzano (BZ)
Raspberry Pi Cluster

- **Aim:**
  - small-scale cloud data centre for teaching/research purposes.

- **Focus:**
  - particularly interested in mobility
  - i.e., how to move clusters to locations where they are needed
  - e.g. in difficult terrain or in emergency circumstances

- **Architecture:**
  - 300 nodes in star topology
BZ Raspberry Pi Cluster

**Architecture:** 300 nodes
- *Network topology:* star
- *Rack:* bespoke
- *Power Supply:* ...
- *OS:* Debian 7

**Cluster management**
- Own solution
- Low-level cluster configuration, monitoring, and maintenance:
  - boot master
  - register RPis
- could use Kubernetes in the future...
BZ RPi Cluster Management

- Cloud platform
  - include centralized management of:
    - resource pool
    - usage monitoring
    - automated service provision
    - online access to acquired resources
  - focus resource management:
    - subcluster organization
    - storage
Cluster orchestration

- The whole cluster is split into subclusters:
  - allowing different users to run processes in parallel on different parts of the infrastructure

- Orchestration:
  - Static orchestration:
    - At the moment a subcluster is composed of a predefined set of RPIs.
  - Dynamic orchestration:
    - It is possible to move an RPi from one subcluster to another one
    - We are working on a solution to make this switch dynamically:
      - i.e., switch while the system is running
      - without burdening the system’s resources too heavily
Cluster storage

- **SD cards slow**: use a network storage system
  - to improve the performance of the overall system
  - to make a common filesystem for the cluster available

**Implementation:**

- a four-bay Network Attached Storage (NAS) from QNAP Systems, allows us to replace the original firmware with a custom Debian image.
- This NAS forms part of the master node
- Inside the NAS:
  - every subcluster has a dedicated volume managed by LVM (logical vol mngr)
  - which is shared by all the R Pis belonging to that subcluster
- The R Pis mount a volume locally via Network File System (NFS) v.4
  - we used NFS rather than iSCSI (Internet Small Computer System Interface)
  - allows sharing of same volume between different nodes,
  - thus making inter-node communication via file system possible
Kubernetes on Raspberry Pi 2’s


- Kubernetes is a powerful orchestration tool for containerised applications across multiple hosts.

- Glasgow Cluster:
  - Fully running implementation of Kubernetes on Raspberry Pi 2
  - Min config:
    - 2 Raspberry Pi 2s
    - Two SD cards loaded with Arch Linux | ARM
Edge Cluster Management Implementation

- Architecture
- Data and Software Management
Technology Stack

Application Layer

Developer Portal

TCP Sockets

REST API

Applications

Data Management

Web Sockets

HTTP

Software Management

Storm

Orchestration

Couchbase

docker

Service Layer

Device Layer

PaaS

Orchestration

kubernetes

REST API

Edge

/

IoT

Stomp

MQTT

REST API

TCP Sockets

Web Sockets

HTTP

iOS

Android

Windows

Raspberry Pi

Arduino
Entities

- Edge Object (EO) – Cloud Object (CO) – Data/Software Processing (DSP)
Management Architecture
Configuration based on ServIoTicy

DSP
Aggregate

Filter (rssi < -70dBm)
Configuration based on ServIoTicy

**Central Cloud**

- **DSP**
  - Aggregate
  - Filter (rssi < -70dBm)

- **Discoverd BLE devices**
  - WO
  - EO

- **Discoverd BLE devices**
  - WO
  - EO

- **Discoverd BLE devices**
  - WO
  - EO
Use Cases - Requirements
Use Cases

- **Smart City**
  - Traffic management
    - Cameras
    - Sensor coordination
    - Space occupancy

- **Smart Area**
  - Tourism
    - Sensor management: temperature, people counters
  - Agriculture
    - Sensors and actuators: sun/precipitation -> irrigation

- **Technology:**
  - Web objects
    - HTTP-enabled devices
    - connect to edge clouds DC or cloudlets (e.g. containers on Rasbp Pi)
Requirements for use cases

Why Docker/Kubernetes on Raspberry Pi:
- Need full data centre (DC) capabilities
- Need easy orchestration
- Need portability / interoperability

Meets requirements of Edge Cloud architectures
Conclusions
Conclusions

- Edge clouds move the focus from heavy-weight data centre clouds to more lightweight virtualised resources
  - Use emerging container technology and container cluster management for edge clouds
  - Some PaaS have started to address limitations in the context of programming (orchestration) and DevOps for clusters.
- Observation: cloud management platforms are still at an earlier stage than the container platforms they build on
- Container technology has the potential to
  - advance PaaS technology towards distributed heterogeneous clouds
  - through lightweightness and interoperability
Thank you!

Claus.Pahl@unibz.it