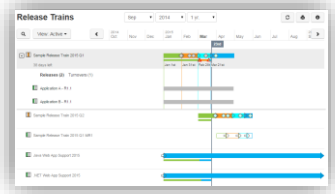
Microservices, Containers, Clusters and PaaS

What the uninitiated really need to know!

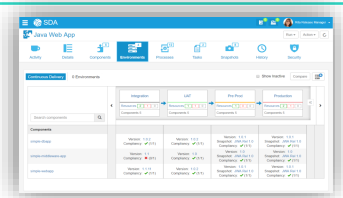
Kevin A. Lee – kevin.lee@microfocus.com

Senior Solutions Architect

DevOps and Release Management Webinars (UK)



- **Implementing Release Management in a DevOps World**



- **Continuous Delivery for All**: Automation for greenfield AND heritage applications

- **Microservices, Containers, Clusters and PaaS**: what the uninitiated really need to know!

Disclaimer!

- This presentation is aimed at those who want to (or need to) learn about the practicalities of this topic, but at a high-medium level.
- If you are already a container guru, you might know most of this stuff already…

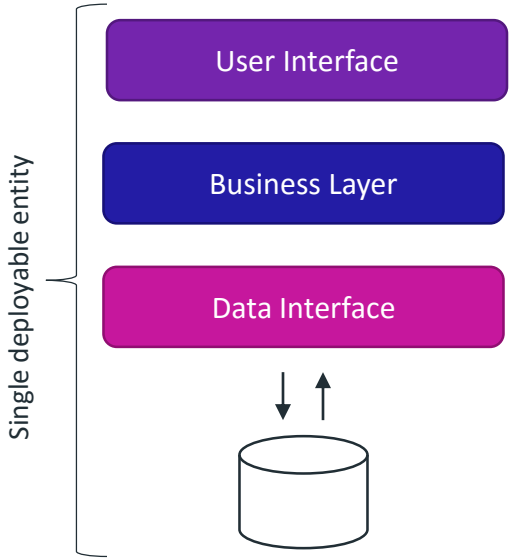
Agenda

- From Monoliths to Microservices…
- Deploying Microservices in Containers with Docker
- Containers in Production
- Micro Focus Solutions
- Demonstration
- Q&A

From Monoliths to Microservices

Monolithic Applications

- Monoliths are typically implemented using a classic “three-tier” architecture.
- Although there may be many “components” that are developed separately they are deployed together in a single process.
  - e.g. Java WAR/EAR file
- Scalability is usually achieved by replicating the whole monolith on multiple servers.



Monoliths = Siloed Functional Teams

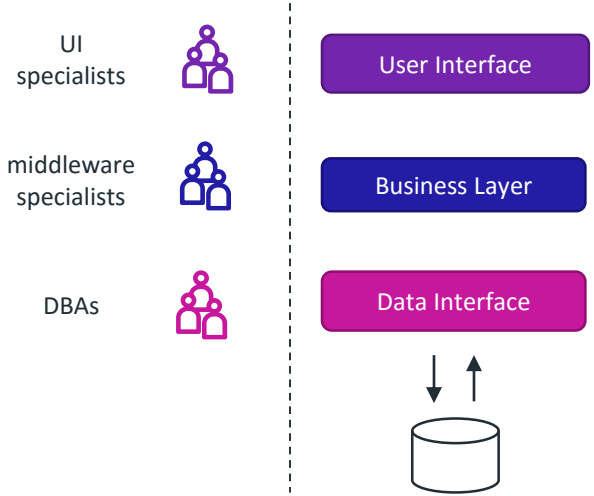
- Because components are NOT aligned to business functionality, there is a tendency to create siloes.
- For example: to develop a new feature, three cross functional teams might be required:

  - UI, Middleware, DBAs

- This enforces Conways Law:

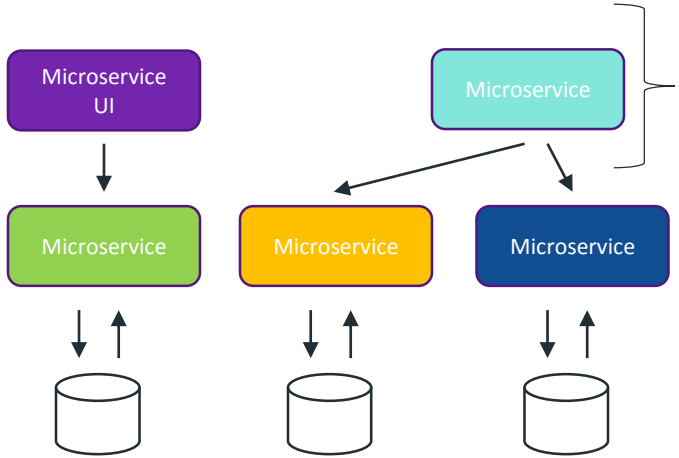
  - *"Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure"*

- And is Anti-DevOps!



Microservice Applications

- Components are developed (as services) AND deployed separately.
- Focus of each team (building a service) is on maintaining the business functionality given by that service.
- Each team requires cross functional skills.
- Scalability achieved by distributing services across servers, replicating as needed.
- Similar in principle to SOA but more “fine-grained” with decentralized governance…



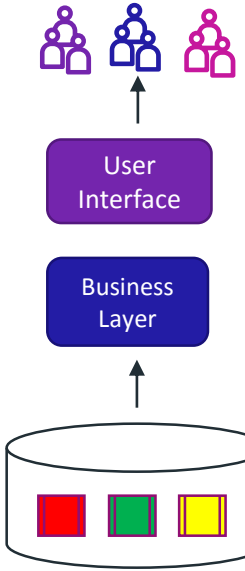
Independent entities with cross communication through apis

Microservices = Decentralized Governance

- Monolithic

Standard set of technology and frameworks

Decisions made by key architects and DBAs



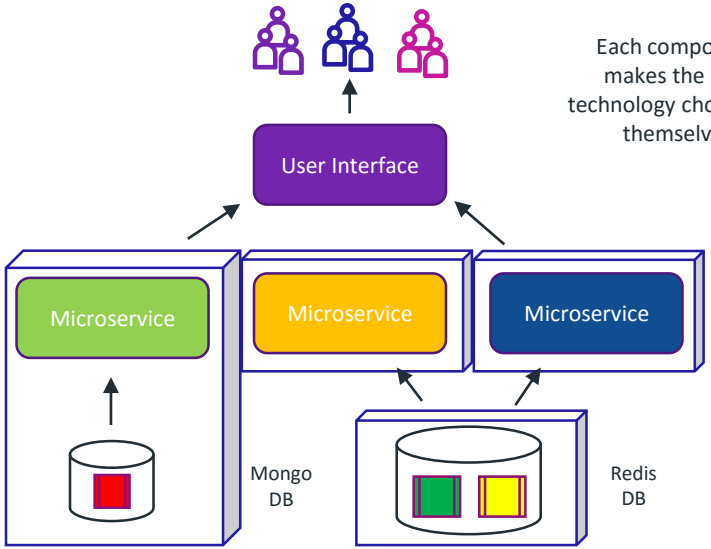
JSF

WebSphere MQ

Oracle DB

- Microservices

Each component makes the best technology choices for themselves



Spring, Apache …

Mongo DB

Redis DB

Microservices: Pros and Cons

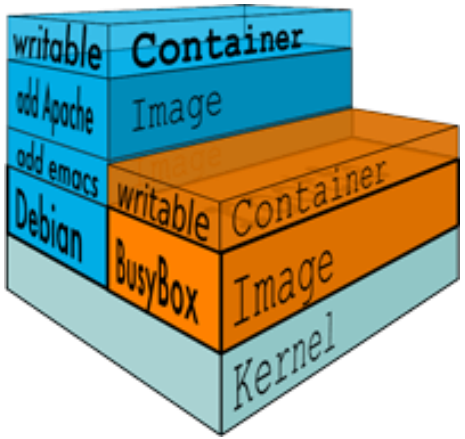
- Ease of deployment
- Ease of enhancement
- Easier to understand
- Resilience
- Scalability
- Freedom to choose technology
- Attracts the best talent… ?

- Increased resource use? (at least initially)
- Increased network communication
- Requires additional skills and capabilities to secure, test and monitor
- Requires disciplined Configuration Management capabilities
- Requires strong DevOps skills (versioning, automation, CI/CD)

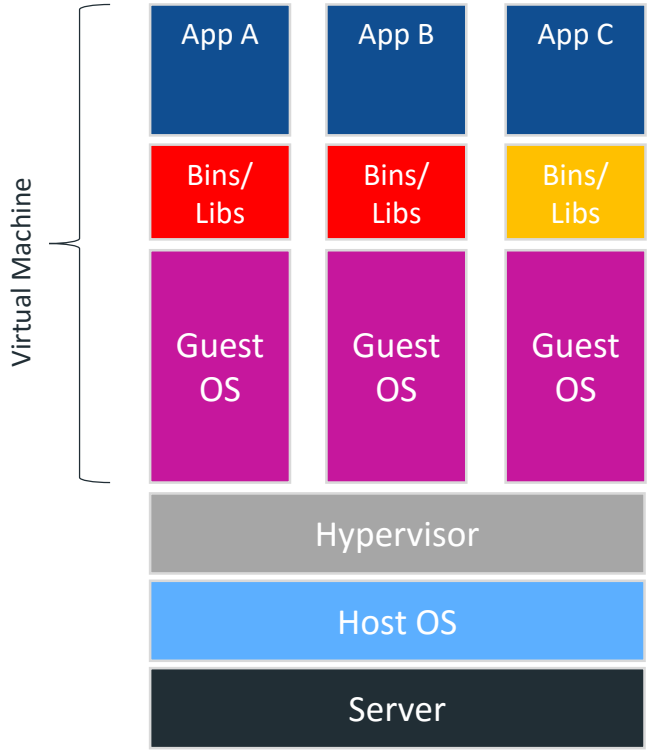
Deploying Microservices in Containers … with Docker

Containerization

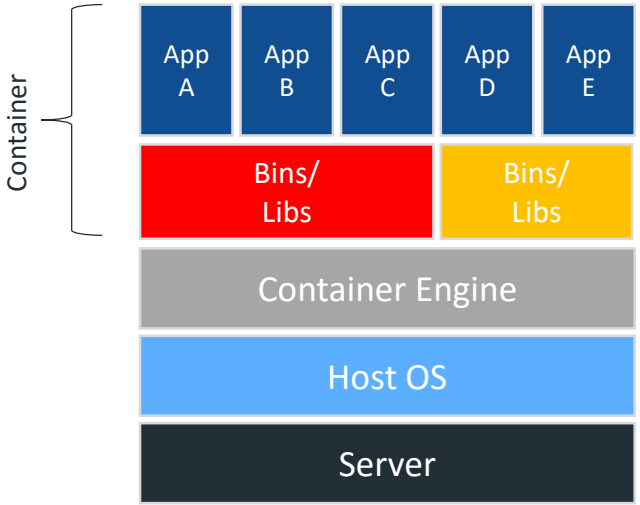
- A container wraps up a piece of software in a complete “filesystem” - it contains everything it needs to run:
  - code, o/s runtime, system tools, system libraries – anything you can install on a server
- This guarantees that it will always run the same, regardless of the environment it is running in.
- Containers are built and run from base images.



Virtual Machines versus Containers

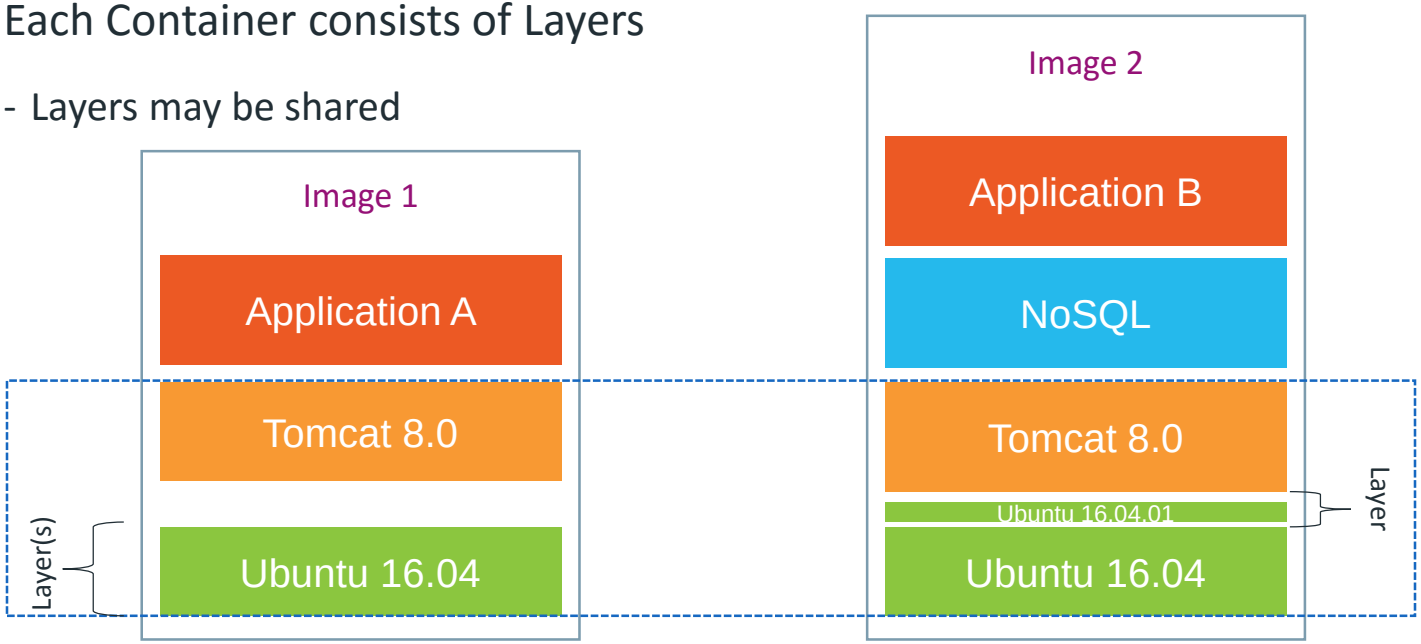


Containers are isolated, But share OS and, where appropriate, Bins/Libs



Containers and Layers

- Each Container consists of Layers
  - Layers may be shared



  - Layers are built on each other – like a Git “hash”: b0facfa253f5

Docker

- The leading containerization technology (but not the only one).
- Some terminology:
  - **Image** – ordered collection of filesystem changes to provide some capability
  - **Container** – a runtime instance of an image
  - **Engine** - the container runtime which creates and runs Docker containers
  - **Dockerfile** – file that contains the commands to build an image
- The Docker Engine runs natively on Linux, UNIX and now Windows*
- Developers share their images with others via a **Docker Registry**:
  - Online in the cloud using **Docker Hub**
  - On-premise using a **Docker Private Registry**

*Windows Server 2016

Example Dockerfile

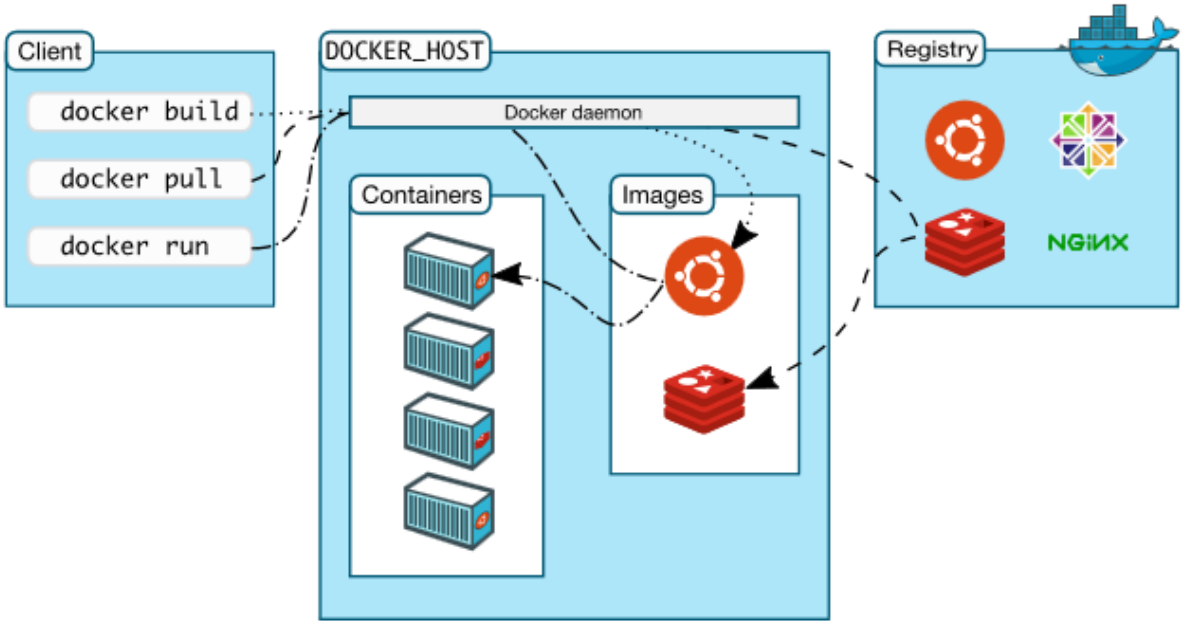
```
01:  # Container for Web User Interface
02:  LABEL description="Container for Web User Interface"
     Vendor="Micro Focus" Version="1.0"
03:  LABEL maintainer "dev@devops.local"
04:  FROM frolvlad/alpine-oraclejdk8:slim
05:  ADD target/web-ui-1.0-SNAPSHOT.jar app.jar
06:  RUN sh -c 'touch /app.jar'
07:  ENV JAVA_OPTS=""
08:  ENV HTTP_PORT=""
09:  ENV EUREKA_HOST=""
10:  ENV EUREKA_PORT=""
11:  ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar
     /app.jar -p $HTTP_PORT -eh $EUREKA_HOST -ep $EUREKA_PORT" ]
```

Line 4: the base image to build new image from

Line 5: files to add into the new image

Line 11: what to do when container based on this image is started

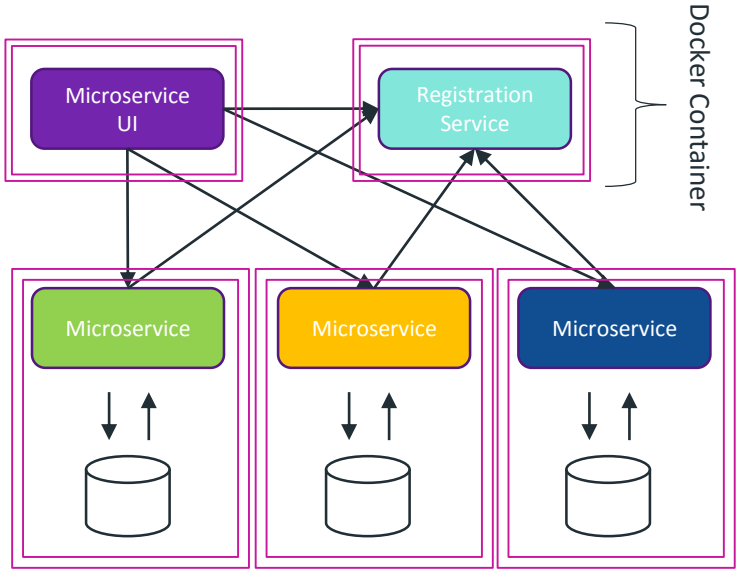
Docker Example



Docker client sends requests to docker demon which talks HTTP(S) to the registry to push/pull images as needed.

Docker and Microservices

- Docker is not just for Microservices but the paradigm works very well as each microservice:
  - is deployed in its own container
  - has the same/similar Bins/Libs, so many Layers are shared
- A Registration Service (in another container) is used to allow the containers to discover each other and communicate through REST APIs.
- For deployment to an environment, one or more containers are simply “pulled” from a Docker Registry and “run”…



Independent entities with cross communication through apis

Containers in Production

Docker Challenges

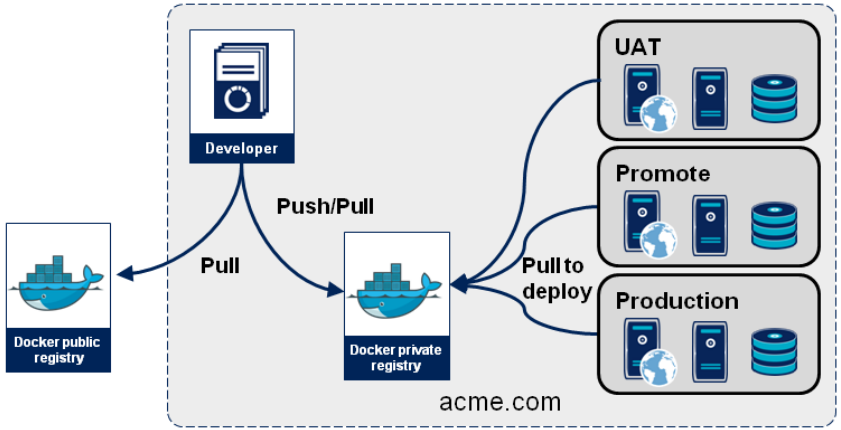
A single Microservices application might consist of 10s/100s of containers.

In Production we need to think about:

- **Container Security**:
  - where are our images stored? have they been they approved?
  - are there any security issues with base images/layers?
- **Container Configuration**:
  - how do we pass configuration to our containers for different environments?
  - how do we capture/parse all the logs? what do we backup?
- **Container Development Lifecycle**
  - how do package code securely and repeatedly into containers for production deployment
  - how do ensure traceability from code to image to container
- **Container Deployment and Scaling**:
  - how do we rollout new versions or provision complete new environments – one by one, as a set?
  - how do we scale up/down for usage?
  - and what is the real performance compared to monoltihs?

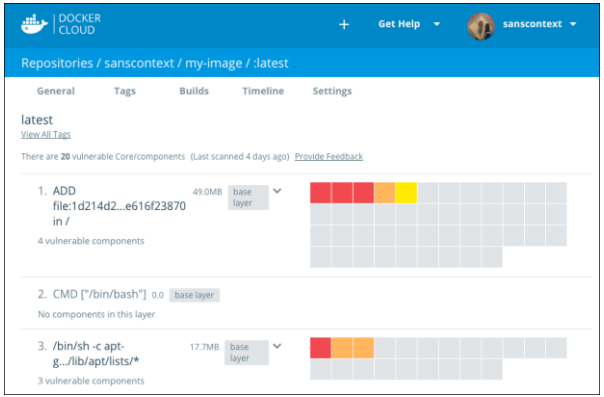
Container Security: Docker Private Registry

- Public Registry Challenges:
  - Lack of governance.
  - Can you trust the images?
  - Performance/Availability.
- Private Registry Benefits:
  - Local/Shareable.
  - Traceability/Consistency.
  - Governance (Image Promotion)
  - High Availability.



Container Security: Docker Image Scanning

- Scan images in private repositories to verify that they are free from known security vulnerabilities or exposures
- Report the results of the scan for each image tag
- Based on known CVE identifiers
- Available in open source products (clair), commercial products (aqua) as well as Docker Cloud



aqua

clair
A Container Image Security Analyzer by CoreOS

Container Configuration

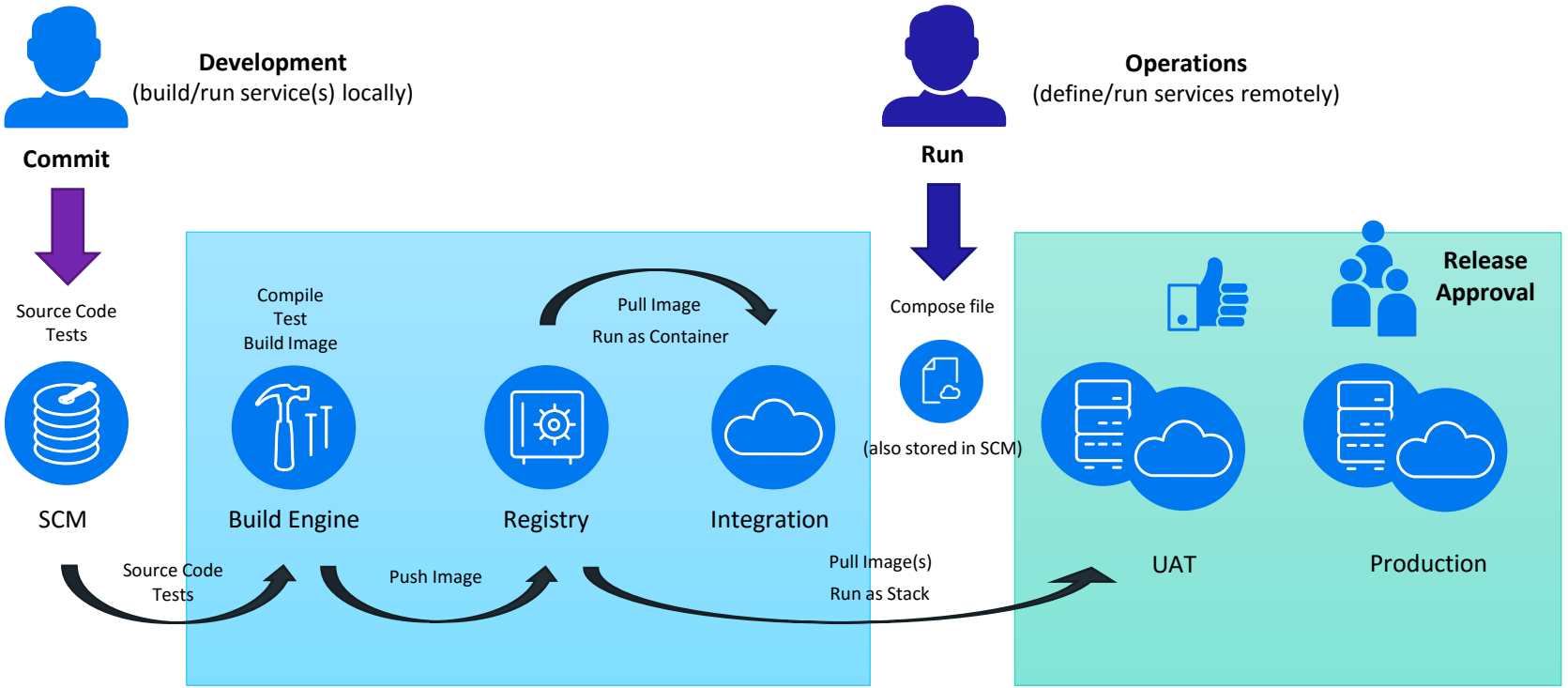
- How do we deploy a containers to different environments - with different configurations (e.g. ports/passwords/file locations)?

Worst

Best

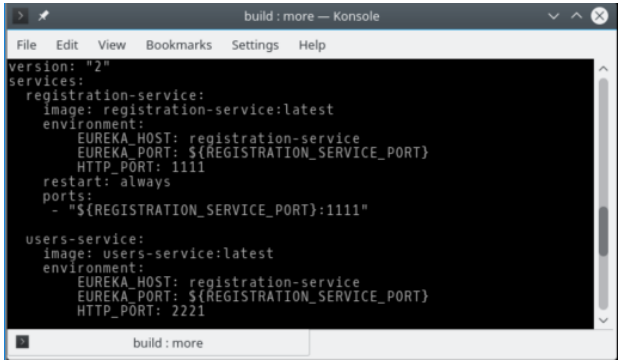
1. Create different containers for different environments
2. Pass environment variables to containers at startup
3. Mount the “configuration” directory to the host
4. Use a distributed service discovery and configuration tool (Consul, etcd, Zookeeper)

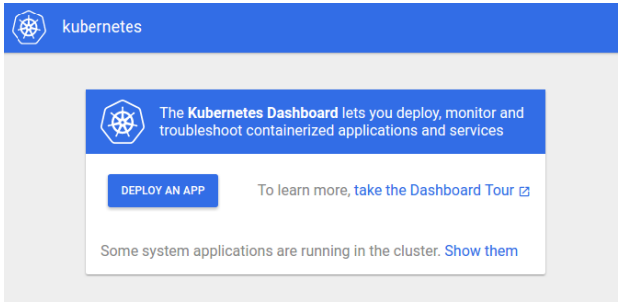
Example Container Development Lifecycle



Container Deployment

- Using Docker Pull & Run
  - Deploy individual container through “docker pull” and “run” commands.
- Using Docker Compose
  - Deploy multiple containers (for each) application through “docker-compose” cli.
  - “docker-compose” format also used by clustering tools.
- Through Docker Cluster/Server API Commands
  - each Cluster solution has its own API for:
    - Deployment/Rollback
    - Rolling deployments/Canary deployments





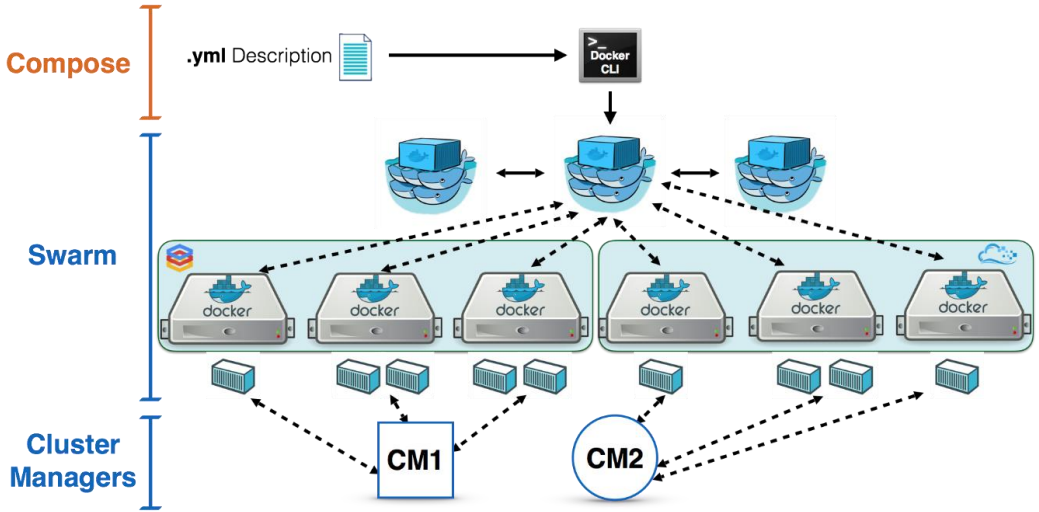
Docker Clustering and Container Management

- Clustering:
  - Abstracts collection of physical/virtual resources.
  - Supports scaling/load balancing/fault tolerance.
  - You do not need to know about underlying hardware.
- Container Management Tools:
  - Docker Swarm, Kubernetes, Apache Mesos.
- Cloud Container Services:
  - Amazon EC2 Container Service.
  - Azure Container Service.
  - OpenStack.
  - …

We have containers running as a service:

- If a **container fails** another is created.
- If a **node fails** or is **shutdown**, any containers running on it are replaced by new containers on other nodes.

Docker Compose/Swarm Example




```
version: "3"
services:
  registration-service:

…

 users-service:
    image: users-service:latest
    environment:
        EUREKA_HOST: registration-service
        EUREKA_PORT: ${REGISTRATION_SERVICE_PORT}
        HTTP_PORT: 2221
    restart: always
    ports:
     - "${USERS_SERVICE_PORT}:2221"
    links:
     - registration-service
…
```

Example docker-compose.yml

Micro Focus Solutions

Everything we do is based on a simple idea–

the fastest way to get results from new technology is to build on what you have.

In essence, **bridging the old and the new.**

Micro Focus ADM – Portfolio Summary

PLAN

Project, Portfolio and Requirements

- Project & Portfolio Mgmt
- Atlas
- Caliber
- Dimensions RM
- Rhythm

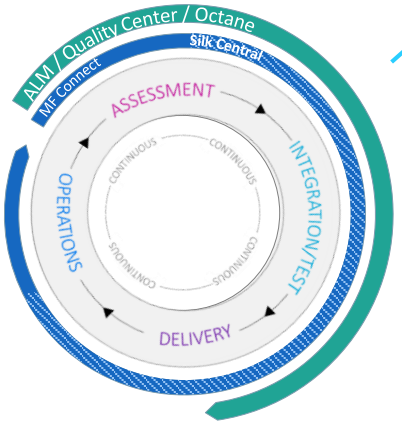
OPERATE

Application and User Monitoring

- AppPulse
- Silk Performance Manager

RELEASE/DEPLOY

- Release Control
- Deployment Automation



BUILD

Software Change & Configuration Mgmt

- AccuRev
- Dimensions CM
- Star Team
- PVCS

TEST

Functional Test

- UFT
- BPT
- Sprinter
- StormRunner Functional
- Silk Test
- Silk WebDriver

Performance Test

- LoadRunner
- Performance Center
- StormRunner Load
- Silk Performer

Digital Lab

- Mobile Center
- Service Virtualization
- Network Virtualization

Micro Focus Container Support

Build

- **Dimensions**
  - Enforce secure container development lifecycle, automating continuous inspection through changeset visualization and expert chains.
  - Docker lifecyle plugins
  - Provides on-premise secure Docker Private Registry

Test

- **Fortify**
  - Scan and identify application security vulnerabilities throughout development (before and after containerization)
- **StormRunner**
  - Agile load and performance testing for developers and dev testers to test and optimize Docker applications.
  - Use Docker for on-premise load generators

Deploy

- **Deployment Automation**
  - Container configuration, approval and deployment through environment lifecycle
  - Docker Registry source plugin
  - Docker lifecycle plugins
  - Kubernetes, Openshift … plugins

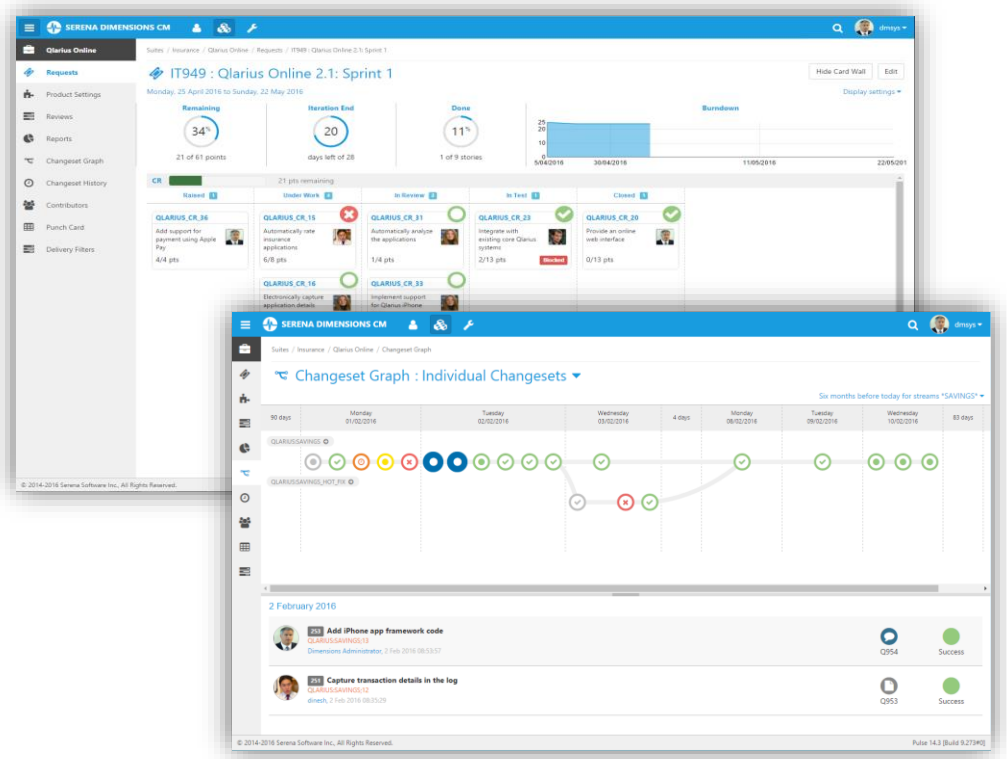
Pulse (Dimensions CM)

Key Capabilities

- Graphical changeset graph
- Automated Expert Toolchains
  - (Docker/Fortify/DA)
- Agile Request Management
- Git/Subversion APIs

Value Benefits

- Optimized developer experience
- Rapid branching/merging
- Automated Continuous Inspection
- Open API



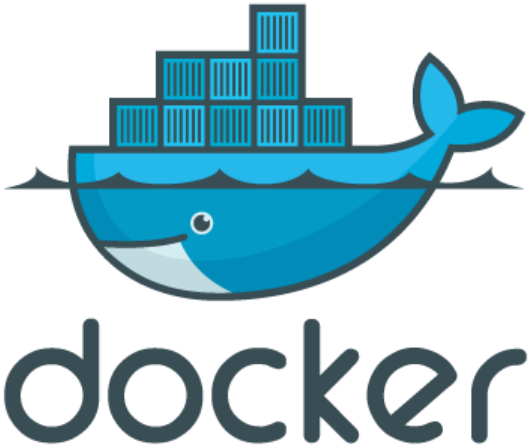
Docker Registry (Dimensions CM)

Key Capabilities

- Enterprise strength Docker Private Registry
- Versioning & approvals of images
- Traceability from change request to image
- Granular access control for images

Value Benefits

- Single source for all your artefacts
- Scalable and fault tolerant
- Secure and control what images are deployed
- Full traceability



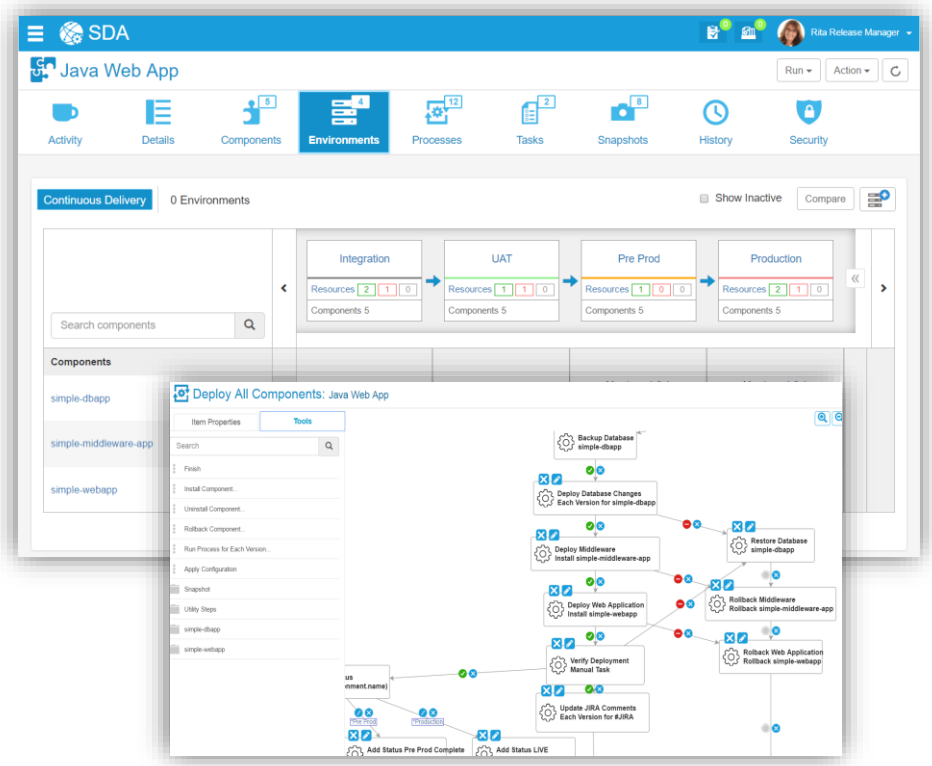
Deployment Automation

Key Capabilities

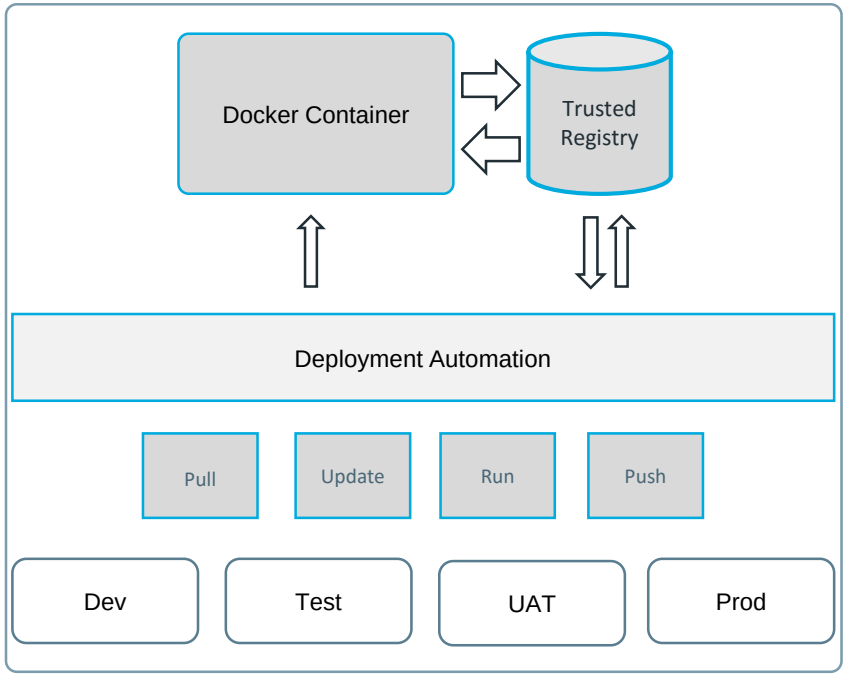
- Automation Workflow Modelling
- Deployment Pipelines
- Environment Inventory
- Approvals, Statuses & Gates
- Release Snapshots

Value Benefits

- Collaborate across teams and environments
- Visualise deployment progress and environment inventory
- Leverage existing investments



Docker Deployment (Deployment Automation)



Full support for Public and Private Trusted Registries

DA **components** are mapped to **images** in Registry

DA monitors Registry for new **tags** and creates equivalent **component versions** in DA

Images are pulled into target environments (Dev / Test / UAT / Prod) following **pipelines** defined in Deployment Automation and "run" as containers

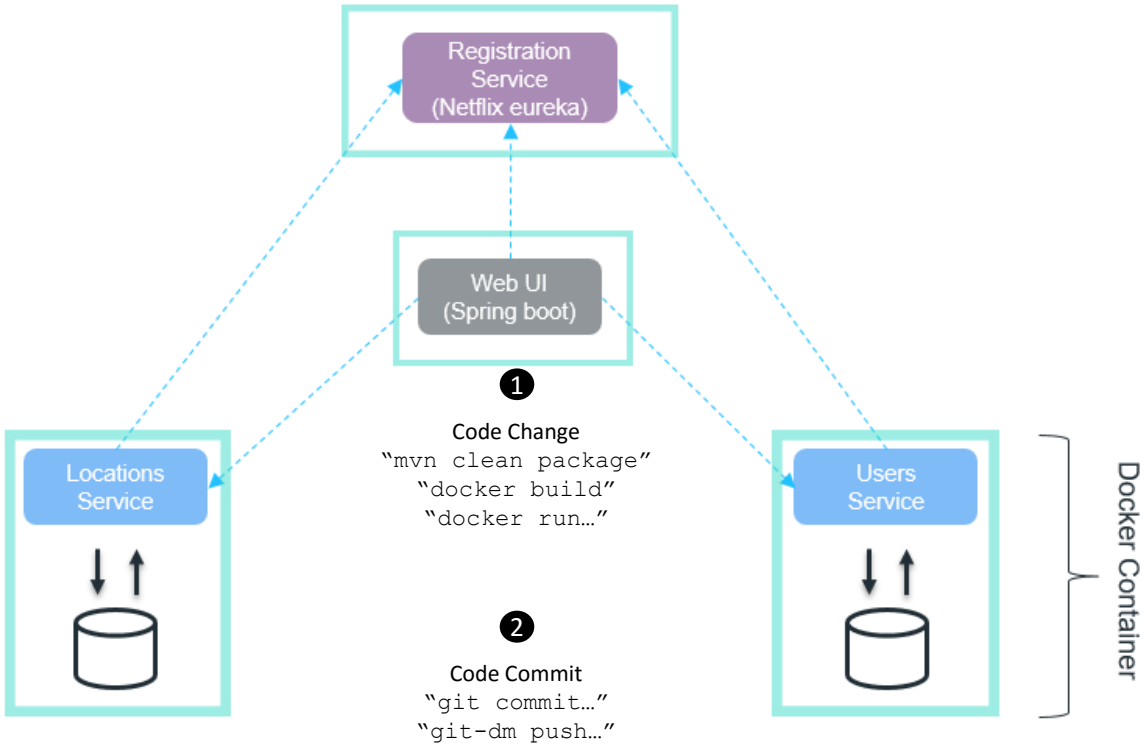
DA **snapshots** can be created to deploy multiple containers

Full support is provided for running individual containers and/or multiple containers through compose files

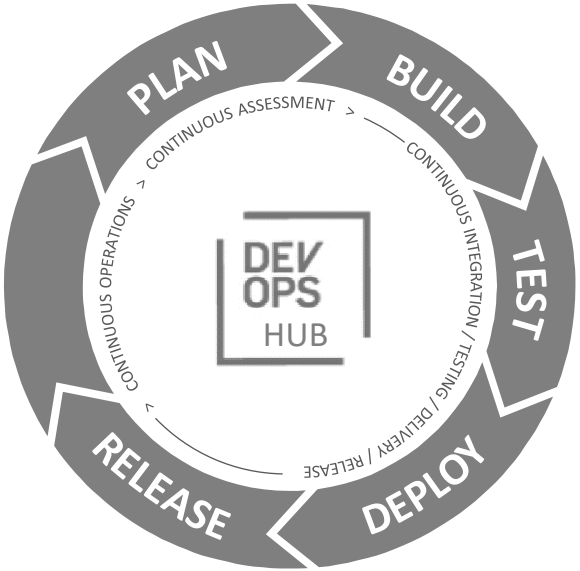
Containers can be updated and pushed into the Registry as new images via Automation if required

Demonstration

Demonstration Scenario (Application)



Demonstration Scenario



Build

- Code change to Web-UI microservice
- Build/Test locally (as Docker container)

- Commit to local **Git** repo
- Push to remote **Dim CM** repo (git-dm) – ***"changeset" created..***
- Automated *Continuous Integration* by **Pulse** expert chain**:**
  - Code Inspection
  - Maven Java Build
  - Build Docker Image (with "changeset" as tag)
  - Deploy to Int using **Deployment Automation** (with "changeset" as version)
- (Optional) Peer Review in **Pulse**

Test

- Automated regression test by **Silk Test, StormRunner** etc

Deploy

- Create snapshot in **DA** for all image versions (changesets)
- Deploy all services using **DA** Docker Compose plugin

Release

- *Deploy all services to Production using **DA** Docker Swarm plugin*
- *…*

Supporting Collateral

- **Containerization vs. virtualization in cloud computing:**
  - http://www.serena.com/index.php/en/campaign/release-management-mesos-and-containerization/
- **Practical Guide to DevOps and Continuous Delivery:**
  - http://www.serena.com/index.php/en/campaign/practical-guide-enterprise-devops-and-continuous-delivery/

Q&A

Kevin A. Lee
kevin.lee@microfocus.com
+44 (0)7799 072507

Kevin A. Lee

Email: kevin.lee@microfocus.com

Tel: (+44) 07799 072507

Thank You!!